FREE eBook

LEARNING xpath

Free unaffiliated eBook created from
Stack Overflow contributors.

#xpath

Table of Contents

About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: xpath

It is an unofficial and free xpath ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official xpath.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapter 1: Getting started with xpath

Remarks

XPath is a language for addressing parts of an XML document.

It is used in XSLT and is a subset of XQuery. Libraries are available for most other programming languages as well.

XPath is an international standard with specifications published by W3C:

- XPath 1.0: XML Path Language (XPath), Version 1.0
- XPath 2.0: XML Path Language (XPath) 2.0 (Second Edition)
- XPath 3.0: XML Path Language (XPath) 3.0

Versions

Version	Release Date
1.0	1999-12-16
2.0	2007-01-23
3.0	2014-04-08
3.1 (W3C Candidate Recommendation)	2015-12-17

Examples

Sample XML (without namespaces)

Here is some sample XML against which example XPaths can be written:

```
<r>
  <e a="1"/>
  <f a="2" b="1">Text 1</f>
  <f/>
  <g>
    <i c="2">Text 2</i>
    Text 3
    <j>Text 4</j>
  </g>
</r>
```

Select text

For the sample XML (without namespaces):

This XPath,

```
/r/f/text()
```

will select the text node with this string value:

```
"Text 1"
```

And this XPath,

```
string(/r/f)
```

will return the string value of `f`, which is also:

```
"Text 1"
```

Select an element

For the sample XML (without namespaces):

This XPath,

```
/r/e
```

will select this element:

```
<e a="1"/>
```

Common HTML Operations

If the input HTML DOM is

```
<html>
    <body>
        <a>link</a>
        <div class='container' id='divone'>
            <p class='common' id='enclosedone'>Element One</p>
            <p class='common' id='enclosedtwo'>Element Two</p>
        </div>
    </body>
</html>
```

Find an element with a specific id in the entire page

```
//*[@id='divone'] # Returns <div class='container' id='divone'>
```

Find an element with a specific id in a particular path

```
/html/body/div/p[@id='enclosedone'] # Returns <p class='common' id='enclosedone'>Element
One</p>
```

Select an element with a particular id & class

```
//p[@id='enclosedone' and @class='common'] # Returns <p class='common'
id='enclosedone'>Element One</p>
```

Select the text of a particular element

```
//*[@id='enclosedone']/text() # Returns Element One
```

Testing Xpaths in browser console

A quick way to test your xpath is in your browser developer tool console.

Format is

```
   $x('//insert xpath here')
```

$ - specifies it is a selector.

x - specifies it is using xpaths

Example:

```
$x("//button[text() ='Submit']")
```

When this command is entered it will return all occurrences of elements that are buttons with text equal to Submit.

Read Getting started with xpath online: https://riptutorial.com/xpath/topic/883/getting-started-with-xpath

Chapter 2: Check if a node is present

Syntax

- boolean(path_to_node)

Remarks

Boolean function has other uses

1. Check if a string is empty
2. Check if the argument is not a number (NaN) or is 0

Examples

Does the animal have tusks?

XML

```
<Animal>
    <legs>4</legs>
    <eyes>2</eyes>
    <horns>2</horns>
    <tail>1</tail>
</Animal>
```

XPATH

```
boolean(/Animal/tusks)
```

OUTPUT

```
false
```

Does the animal have horns?

XPATH

```
<Animal>
    <legs>4</legs>
    <eyes>2</eyes>
    <horns>2</horns>
    <tail>1</tail>
</Animal>
```

XPATH

```
boolean(/Animal/horns)
```

OUTPUT

```
true
```

Are there plants in the house?

XML

```
<House>
    <LivingRoom>
        <plant name="rose"/>
    </LivingRoom>
    <TerraceGarden>
        <plant name="passion fruit"/>
        <plant name="lily"/>
        <plant name="golden duranta"/>
    </TerraceGarden>
</House>
```

XPATH

```
boolean(/House//plant)
```

OUTPUT

```
true
```

Read Check if a node is present online: https://riptutorial.com/xpath/topic/7432/check-if-a-node-is-present

Chapter 3: Check if a node's text is empty

Syntax

- boolean(path_to_node/text())
- string(path_to_node) != ''

Remarks

Boolean function has other uses

1. Check if a node is present
2. Check if the argument is not a number (NaN) or is 0

String function is used to return the string value of a node.

Examples

Check if Deborah has a master and its text value is not empty

XML

```
<Deborah>
    <address>Dark world</address>
    <master>Babadi</master>
    <ID>#0</ID>
    <colour>red</colour>
    <side>evil</side>
</Deborah>
```

XPATH

```
boolean(/Deborah/master/text())
```

OR

```
string(/Deborah/master) != ''
```

OUTPUT

```
true
```

Check if Dobby has a master and its text value is not empty

XML

```
<Dobby>
    <address>Hogwartz</address>
    <master></master>
    <colour>wheatish</colour>
    <side>all good</side>
</Dobby>
```

XPATH

```
boolean(/Dobby/master/text())
```

OR

```
string(/Dobby/master) != ''
```

OUTPUT

```
false
```

Read Check if a node's text is empty online: https://riptutorial.com/xpath/topic/7445/check-if-a-node-s-text-is-empty

Chapter 4: Find nodes that have a specific attribute

Syntax

1. Inside a specific node
    - /path to/element[@attribute_name]
2. Anywhere in the document
    - //*[@attribute_name]
3. Inside a specific node with some value
    - /path to/element[@attribute_name='search value']
    - /path to/element[@attribute_name="search value"]
4. Anywhere in the document with some value
    - //*[@attribute_name='search string']
    - //*[@attribute_name="search string"]

Parameters

Selector	function
@attribute_name	It selects the attribute value for a node, if present

Remarks

Using [@attribute_name] we can select nodes that have the attribute irrespective of the value.

We can use any of the functions or combination of the functions such as starts-with and lowercase, for example, with this selector to suit our needs.

Examples

Find nodes with a specific attribute

XML

```
<Galaxy>
    <name>Milky Way</name>
    <CelestialObject name="Earth" type="planet"/>
    <CelestialObject name="Sun" type="star"/>
</Galaxy>
```

XPATH

```
/Galaxy/*[@name]
```

or

```
//*[@name]
```

OUTPUT

```
<CelestialObject name="Earth" type="planet" />
<CelestialObject name="Sun" type="star" />
```

Find nodes with a specific attribute value

XML

```
<Galaxy>
    <name>Milky Way</name>
    <CelestialObject name="Earth" type="planet"/>
    <CelestialObject name="Sun" type="star"/>
</Galaxy>
```

XPATH

```
/Galaxy/*[@name='Sun']
```

or

```
//*[@name='Sun']
```

OUTPUT

```
<CelestialObject name="Sun" type="star" />
```

Find nodes by substring matching of an attribute's value

XML

```
<Galaxy>
    <name>Milky Way</name>
    <CelestialObject name="Earth" type="planet"/>
    <CelestialObject name="Sun" type="star"/>
</Galaxy>
```

XPATH

```
/Galaxy/*[contains(@name,'Ear')]
```

or

```
//*[contains(@name,'Ear')]
```

Double quotes can also be used in place of single quotes:

```
/Galaxy/*[contains(@name, "Ear")]
```

OUTPUT

```
<CelestialObject name="Earth" type="planet" />
```

Find nodes by substring matching of an attribute's value (case-insensitive)

XML

```
<Galaxy>
    <name>Milky Way</name>
    <CelestialObject name="Earth" type="planet"/>
    <CelestialObject name="Sun" type="star"/>
</Galaxy>
```

XPATH

```
/Galaxy/*[contains(lower-case(@name),'ear')]
```

or

```
//*[contains(lower-case(@name),'ear')]
```

or, with the string in double quotes:

```
//*[contains(lower-case(@name), "ear")]
```

OUTPUT

```
<CelestialObject name="Earth" type="planet" />
```

Find nodes by substring matching the start of an attribute's value

XML

```
<Galaxy>
    <name>Milky Way</name>
    <CelestialObject name="Earth" type="planet"/>
    <CelestialObject name="Sun" type="star"/>
</Galaxy>
```

XPATH

```
/Galaxy/*[starts-with(lower-case(@name),'ear')]
```

or

```
//*[starts-with(lower-case(@name),'ear')]
```

OUTPUT

```
<CelestialObject name="Earth" type="planet" />
```

Find nodes by substring matching the end of an attribute's value

XML

```
<Galaxy>
    <name>Milky Way</name>
    <CelestialObject name="Earth" type="planet"/>
    <CelestialObject name="Sun" type="star"/>
</Galaxy>
```

XPATH

```
/Galaxy/*[ends-with(lower-case(@type),'tar')]
```

or

```
//*[ends-with(lower-case(@type),'tar')]
```

OUTPUT

```
<CelestialObject name="Sun" type="star" />
```

Read Find nodes that have a specific attribute online: https://riptutorial.com/xpath/topic/3096/find-nodes-that-have-a-specific-attribute

Chapter 5: Finding elements containing specific attributes

Examples

Find all elements with a certain attribute

Imagine the following XML:

```
<root>
    <element foobar="hello_world" />
    <element example="this is one!" />
</root>
```

```
/root/element[@foobar]
```

and will return the `<element foobar="hello_world" />` element.

Find all elements with a certain attribute value

Imagine the following XML:

```
<root>
    <element foobar="hello_world" />
    <element example="this is one!" />
</root>
```

The following XPath expression:

```
/root/element[@foobar = 'hello_world']
```

will return the `<element foobar="hello_world" />` element.

double quotes can also be used:

```
/root/element[@foobar="hello_world"]
```

Read Finding elements containing specific attributes online:
https://riptutorial.com/xpath/topic/6488/finding-elements-containing-specific-attributes

Chapter 6: Finding elements containing specific text

Examples

Find all elements with certain text

Imagine the following XML:

```
<root>
    <element>hello</element>
    <another>
        hello
    </another>
    <example>Hello, <nested> I am an example </nested>.</example>
</root>
```

The following XPath expression:

```
//*[text() = 'hello']
```

will return the `<element>hello</element>` element, but not the `<another>` element. This is because the `<another>` element contains whitespace surrounding the `hello` text.

To retrieve both `<element>` and `<another>`, one could use:

```
//*[normalize-space(text()) = 'hello']
```

or

```
//*[normalize-space() = 'hello']
```

which will trim the surrounding whitespace before doing the comparison. Here we can see that the `text()` node specifier is optional when using `normalize-space`.

To find an element *containing* specific text, you can use the `contains` function. The following expression will return the `<example>` element:

```
//example[contains(text(), 'Hello')]
```

If you want to find text that spans multiple children/text nodes, then you can use `.` instead of `text()`. `.` refers to the entire text content of the element and it's children.

```
//example[. = 'Hello,  I am an example .']
```

To see the multiple text nodes, you can use:

```
//example//text()
```

which will return:

- "Hello, "
- " I am an example "
- "."

And to more clearly see the entire text content of an element, one can use the `string` function:

```
string(//example[1])
```

or just

```
string(//example)
```

> Hello, I am an example .

The latter works because, if a node-set is passed into functions like `string`, XPath 1.0 just looks at the first node (in document order) in that node-set, and ignores the rest.

so:

```
string(/root/*)
```

would return:

> hello

Read Finding elements containing specific text online:
https://riptutorial.com/xpath/topic/1903/finding-elements-containing-specific-text

Chapter 7: Get nodes relative to the current node

Syntax

1. All ancestors of a node
    - /path to the node/ancestor::node()
2. A specific ancestor of a node
    - /path to the node/ancestor::ancestor_name
3. Parent of a node
    - /path to the node/parent::node()
4. Following siblings of a node
    - /path to the node/following-sibling::node()
5. A specific sibling following a node
    - /path to the node/following-sibling::sibling_name
6. Preceding siblings of a node
    - /path to the node/preceding-sibling::node()
7. A specific sibling preceding a node
    - /path to the node/preceding-sibling::sibling_name
8. All immediate child nodes of a node
    - /path to the node/child::node()
9. A specific immediate child node of a node
    - /path to the node/child::chid_name
10. All the descendants of a node
    - /path to the node/descendant::node()
11. All specific descendants of a node
    - /path the to node/descendant::descendant_name

Parameters

Axis	selects
ancestor	all the ancestor nodes
parent	parent node
following-sibling	siblings following the node
preceding-sibling	siblings preceding the node
child	immediate children
descendant	all the descendant irrespective of the nesting level

Remarks

These axes can be used in combination with other functions to suit our needs.

Examples

Find My ancestors

XML

```
<GrandFather name="Bardock" gender="male" spouse="Gine">
    <Dad name="Goku" gender="male" spouse="Chi Chi">
        <Me name="Gohan" gender="male"/>
        <brother name="Goten" gender="male"/>
    </Dad>
</GrandFather>
```

XPATH

```
//Me/ancestor::node()
```

OUTPUT

```
<GrandFather name="Bardock" gender="male" spouse="Gine">
  <Dad name="Goku" gender="male" spouse="Chi Chi">
    <Me name="Gohan" gender="male" />
    <brother name="Goten" gender="male" />
  </Dad>
</GrandFather>
<Dad name="Goku" gender="male" spouse="Chi Chi">
  <Me name="Gohan" gender="male" />
  <brother name="Goten" gender="male" />
</Dad>
```

Find My Parent

XML

```
<GrandFather name="Bardock" gender="male" spouse="Gine">
    <Dad name="Goku" gender="male" spouse="Chi Chi">
        <Me name="Gohan" gender="male"/>
        <brother name="Goten" gender="male"/>
    </Dad>
</GrandFather>
```

XPATH

```
//Me/ancestor::Dad
```

or

```
//Me/parent::node()
```

OUTPUT

```
<Dad name="Goku" gender="male" spouse="Chi Chi">
  <Me name="Gohan" gender="male" />
  <brother name="Goten" gender="male" />
</Dad>
```

Find my grand father

XML

```
<GrandFather name="Bardock" gender="male" spouse="Gine">
  <Dad name="Goku" gender="male" spouse="Chi Chi">
    <Me name="Gohan" gender="male" />
    <brother name="Goten" gender="male" />
  </Dad>
</GrandFather>
```

XPATH

```
//Me/ancestor::GrandFather
```

or

```
//Me/parent::node()/parent::node()
```

OUTPUT

```
<GrandFather name="Bardock" gender="male" spouse="Gine">
  <Dad name="Goku" gender="male" spouse="Chi Chi">
    <Me name="Gohan" gender="male" />
    <brother name="Goten" gender="male" />
  </Dad>
</GrandFather>
```

Find my brother

XML

```
<GrandFather name="Bardock" gender="male" spouse="Gine">
  <Dad name="Goku" gender="male" spouse="Chi Chi">
    <brother name="Goten" gender="male" />
    <Me name="Gohan" gender="male" />
    <brother name="Goten" gender="male" />
  </Dad>
</GrandFather>
```

XPATH

```
//Me/following-sibling::brother
```

OUTPUT

```
<brother name="Goten" gender="male" />
```

Get all avatars before Parashurama

XML

```
<Dashavatar>
    <Avatar name="Matsya"/>
    <Avatar name="Kurma"/>
    <Avatar name="Varaha"/>
    <Avatar name="Narasimha"/>
    <Avatar name="Vamana"/>
    <Avatar name="Balabhadra"/>
    <Avatar name="Parashurama"/>
    <Avatar name="Rama"/>
    <Avatar name="Krishna"/>
    <Avatar name="Kalki"/>
</Dashavatar>
```

XPATH

```
//Avatar[@name='Parashurama']/preceding-sibling::node()
```

OUTPUT

```
<Avatar name="Matsya"/>
<Avatar name="Kurma"/>
<Avatar name="Varaha"/>
<Avatar name="Narasimha"/>
<Avatar name="Vamana"/>
<Avatar name="Balabhadra"/>
```

Get all avatars after Parashurama

XML

```
<Dashavatar>
    <Avatar name="Matsya"/>
    <Avatar name="Kurma"/>
    <Avatar name="Varaha"/>
    <Avatar name="Narasimha"/>
    <Avatar name="Vamana"/>
    <Avatar name="Balabhadra"/>
    <Avatar name="Parashurama"/>
    <Avatar name="Rama"/>
    <Avatar name="Krishna"/>
    <Avatar name="Kalki"/>
</Dashavatar>
```

XPATH

```
//Avatar[@name='Parashurama']/following-sibling::node()
```

OUTPUT

```
<Avatar name="Rama" />
<Avatar name="Krishna" />
<Avatar name="Kalki" />
```

Get all avatars except the current one (Parusharama)

XML

```
<Dashavatar>
    <Avatar name="Matsya"/>
    <Avatar name="Kurma"/>
    <Avatar name="Varaha"/>
    <Avatar name="Narasimha"/>
    <Avatar name="Vamana"/>
    <Avatar name="Balabhadra"/>
    <Avatar name="Parashurama"/>
    <Avatar name="Rama"/>
    <Avatar name="Krishna"/>
    <Avatar name="Kalki"/>
</Dashavatar>
```

XPATH

```
//Avatar[@name='Parashurama']/following-sibling::Avatar |
//Avatar[@name='Parashurama']/preceding-sibling::Avatar
```

OUTPUT

```
<Avatar name="Matsya" />
<Avatar name="Kurma" />
<Avatar name="Varaha" />
<Avatar name="Narasimha" />
<Avatar name="Vamana" />
<Avatar name="Balabhadra" />
<Avatar name="Rama" />
<Avatar name="Krishna" />
<Avatar name="Kalki" />
```

Get all the details (child nodes) of House

XML

```
<House>
    <Rooms>10</Rooms>
    <People>4</People>
    <TVs>4</TVs>
```

```
    <Floors>2</Floors>
</House>
```

XPATH

```
/House/child::node()
```

OUTPUT

```
<Rooms>10</Rooms>
<People>4</People>
<TVs>4</TVs>
<Floors>2</Floors>
```

Get all rooms (immediate children named Room) in House

XML

```
<House>
    <numRooms>4</numRooms>
    <Room name="living"/>
    <Room name="master bedroom"/>
    <Room name="kids' bedroom"/>
    <Room name="kitchen"/>
</House>
```

XPATH

```
/House/child::Room
```

or

```
/House/*[local-name()='Room']
```

OUTPUT

```
<Room name="living" />
<Room name="master bedroom" />
<Room name="kids' bedroom" />
<Room name="kitchen" />
```

Get all rooms (irrespective of the position) in House

XML

```
<House>
    <numRooms>4</numRooms>
    <Floor number="1">
        <Room name="living"/>
        <Room name="kitchen"/>
```

```
    </Floor>
    <Floor number="2">
        <Room name="master bedroom"/>
        <Room name="kids' bedroom"/>
    </Floor>
</House>
```

XPATH

```
/House/descendant::Room
```

OUTPUT

```
<Room name="living" />
<Room name="kitchen" />
<Room name="master bedroom" />
<Room name="kids' bedroom" />
```

Read Get nodes relative to the current node online: https://riptutorial.com/xpath/topic/6495/get-nodes-relative-to-the-current-node

Chapter 8: Get the count of nodes

Syntax

- count(node-set)

Parameters

function	returns
count	total number of nodes in the node set

Remarks

We can use this in combination of other functions and axes to suit our needs.

Examples

How many children does Goku have?

XML

```
<Goku>
    <child name="Gohan"/>
    <child name="Goten"/>
</Goku>
```

XPATH

```
count(/Goku/child)
```

OUTPUT

```
2.0
```

How many plants are there in the house?

XML

```
<House>
    <LivingRoom>
        <plant name="rose"/>
    </LivingRoom>
    <TerraceGarden>
```

```
        <plant name="passion fruit"/>
        <plant name="lily"/>
        <plant name="golden duranta"/>
    </TerraceGarden>
</House>
```

XPATH

```
count(/House//plant)
```

OUTPUT

```
4.0
```

Read Get the count of nodes online: https://riptutorial.com/xpath/topic/4463/get-the-count-of-nodes

Chapter 9: Location paths and axes

Remarks

An XPath *location path* is a series of *location steps* separated by a `/` character:

```
step1/step2/step3
```

A *location step* contains an *axis*, a *node test*, and an optional list of *predicates*. The *axis* and the *node test* are separated by two colon characters `::`. The *predicates* are enclosed in square brackets:

```
axis::nodeTest[predicate1][predicate2]
```

The evaluation of a *location path* starts with a node set containing the *context node* given by the context of the expression, or the *root node*, if the location path starts with a `/`. At each step, each node *N* in the original node set is replaced with the set of nodes that

- can be reached from *N* following the given *axis*,
- matches the *node test*, and
- matches all *predicates*.

The result of a *location path* expression is the final node set obtained after processing all *location steps*.

Examples

Traversing child elements

Traversing from the root node to a descendant element using the `child` axis:

```
/child::html/child::body/child::div/child::span
```

Since the `child` axis is the default axis, this can be abbreviated to:

```
/html/body/div/span
```

Traversing all descendants

The `descendant` and `descendant-or-self` axes can be used to find all descendant elements of a node at any depth. In contrast, the `child` axis only traverses immediate children.

```
/child::html/descendant::span
/child::html/descendant-or-self::*
```

The double slash `//` is a shortcut for `/descendant-or-self::node()/`. So the following expressions are equivalent:

```
table//td
child::table/descendant-or-self::node()/child::td
child::table/descendant::td
table/descendant::td
```

Traversing ancestors

The `parent` axis contains only the parent of a node. The following expression selects the `html` element by taking a detour over the `body` element:

```
/child::html/child::body/parent::html
```

`..` is a shortcut for `parent::node()`

The `ancestor` and `ancestor-or-self` axes traverse all ancestors of a node. The following expression returns all `div` elements that are ancestors of the context node:

```
ancestor::div
```

The "self" axis

The `self` axis only contains the context node itself. The expression `.` is a shortcut for `self::node()` and always matches the context node. The `.` shortcut is useful for enumerating descendants of the context node. The following expressions are equivalent:

```
.//span
self::node()/descendant-or-self::node()/child::span
descendant::span
```

The `self` axis can be helpful in XPath 1.0 predicates. For example, to select all `h1`, `h2`, and `h3` children of the context node:

```
*[self::h1 or self::h2 or self::h3]
```

Traversing following and preceding nodes

The `following-sibling` and `preceding-sibling` axes contain the siblings before or after the context node, and the `following` and `preceding` axes contain all nodes in the document before or after the context node, but:

- None of these axes contain attribute or namespace nodes.
- The `following` axis doesn't contain any descendants.
- The `preceding` axis doesn't contain any ancestors.

Examples:

```
following::span[1]
following-sibling::*[last()]
```

Traversing attribute and namespace nodes

The `attribute` and `namespace` axes contain all attribute and namespace nodes of an element. The shortcut `@` stands for `attribute::`, so the following are equivalent:

```
child::div/attribute::class
div/@class
```

Read Location paths and axes online: https://riptutorial.com/xpath/topic/6171/location-paths-and-axes

Chapter 10: Namespaces

Remarks

XPath 1.0 doesn't have the concept of a default namespace.

Also, the namespace prefixes defined in the original XML document do not affect XPath - namespace prefixes have to be explicitly registered with the XPath provider, otherwise prefixes can't be used at all in the XPath expression.

Examples

Namespace aware functions

```
<root xmlns="http://test/">
    <element xmlns:example="http://foobar/">
        <example:hello_world attribute="another example" />
    </element>
</root>
```

The expression `/root` will return nothing, because there is no non-namespaced element called `root` at the root level of the document. However, The following *will* return the `<root xmlns="http://test/">` element.

```
/*[namespace-uri() = 'http://test/' and local-name() = 'root']
```

Read Namespaces online: https://riptutorial.com/xpath/topic/2324/namespaces

Chapter 11: Select nodes based on their children

Examples

Select nodes based on child count

Sample XML

```
 <Students>
    <Student>
        <Name>
            <First>Ashley</First>
            <Last>Smith</Last>
        </Name>
        <Grades>
            <Exam1>A</Exam1>
            <Exam2>B</Exam2>
            <Final>A</Final>
        </Grades>
    </Student>
    <Student>
        <Name>
            <First>Bill</First>
            <Last>Edwards</Last>
        </Name>
        <Grades>
            <Exam1>A</Exam1>
        </Grades>
    </Student>
</Students>
```

XPath

Select all students that have at least 2 grades recorded

```
//Student[count(./Grades/*) > 1]
```

Output

```
<Student>
    <Name>
        <First>Ashley</First>
        <Last>Smith</Last>
    </Name>
    <Grades>
        <Exam1>A</Exam1>
        <Exam2>B</Exam2>
        <Final>A</Final>
    </Grades>
</Student>
```

Select nodes based on specific child node

Sample XML

```
 <Students>
    <Student>
        <Name>
            <First>Ashley</First>
            <Last>Smith</Last>
        </Name>
        <Grades>
            <Exam1>A</Exam1>
            <Exam2>B</Exam2>
            <Final>A</Final>
        </Grades>
    </Student>
    <Student>
        <Name>
            <First>Bill</First>
            <Last>Edwards</Last>
        </Name>
        <Grades>
            <Exam1>A</Exam1>
        </Grades>
    </Student>
</Students>
```

XPath

Select all students that have a score for Exam2 recorded

```
//Student[./Grades/Exam2]
```

or

```
//Student[.//Exam2]
```

Output

```
<Student>
    <Name>
        <First>Ashley</First>
        <Last>Smith</Last>
    </Name>
    <Grades>
        <Exam1>A</Exam1>
        <Exam2>B</Exam2>
        <Final>A</Final>
    </Grades>
</Student>
```

Read Select nodes based on their children online: https://riptutorial.com/xpath/topic/6504/select-nodes-based-on-their-children

Chapter 12: Select nodes with names equal to or containing some string

Syntax

1. Inside a specific node:

   {path-to-parent}/name()='search string']

2. Anywhere in the document:

   //*[name()='search string']

Parameters

function	return value
local-name()	the node's name without prefix

Remarks

local-name() result does not include prefix (lookup name() XPATH function for it)

Examples

Search for nodes with name as Light, Device or Sensor

XML

```
<Galaxy>
    <Light>sun</Light>
    <Device>satellite</Device>
    <Sensor>human</Sensor>
    <Name>Milky Way</Name>
</Galaxy>
```

XPATH

```
/Galaxy/*[local-name()='Light' or local-name()='Device' or local-name()='Sensor']
```

or

```
//*[local-name()='Light' or local-name()='Device' or local-name()='Sensor']
```

OUTPUT

```
<Light>sun</Light>
<Device>satellite</Device>
<Sensor>human</Sensor>
```

Search for nodes that has name that contains Light

XML

```
<Data>
    <BioLight>
        <name>Firefly</name>
        <model>Insect</model>
    </BioLight>
    <ArtificialLight>
        <name>Fire</name>
        <model>Natural element</model>
        <source>flint</source>
    </ArtificialLight>
    <SolarLight>
        <name>Sun</name>
        <model>Star</model>
        <source>helium</source>
    </SolarLight>
</Data>
```

XPATH

```
/Data/*[contains(local-name(),"Light")]
```

or

```
//*[contains(local-name(),"Light")]
```

OUTPUT

```
<BioLight>
  <name>Firefly</name>
  <model>Insect</model>
</BioLight>
<ArtificialLight>
  <name>Fire</name>
  <model>Natural element</model>
  <source>flint</source>
</ArtificialLight>
<SolarLight>
  <name>Sun</name>
  <model>Star</model>
  <source>helium</source>
</SolarLight>
```

Search for nodes that has name that starts with Star

XML

```
<College>
    <FootBall>
        <Members>20</Members>
        <Coach>Archie Theron</Coach>
        <Name>Wild cats</Name>
        <StarFootballer>David Perry</StarFootballer>
    </FootBall>
    <Academics>
        <Members>100</Members>
        <Teacher>Tim Jose</Teacher>
        <Class>VII</Class>
        <StarPerformer>Lindsay Rowen</StarPerformer>
    </Academics>
</College>
```

XPATH

```
/College/*/*[starts-with(local-name(),"Star")]
```

or

```
//*[starts-with(local-name(),"Star")]
```

OUTPUT

```
<StarFootballer>David Perry</StarFootballer>
<StarPerformer>Lindsay Rowen</StarPerformer>
```

Search for nodes that has name that ends with Ball

XML

```
<College>
    <FootBall>
        <Members>20</Members>
        <Coach>Archie Theron</Coach>
        <Name>Wild cats</Name>
        <StarPlayer>David Perry</StarPlayer>
    </FootBall>
    <VolleyBall>
        <Members>24</Members>
        <Coach>Tim Jose</Coach>
        <Name>Avengers</Name>
        <StarPlayer>Lindsay Rowen</StarPlayer>
    </VolleyBall>
    <FoosBall>
        <Members>22</Members>
        <Coach>Rahul Mehra</Coach>
        <Name>Playerz</Name>
        <StarPlayer>Amanda Ren</StarPlayer>
    </FoosBall>
</College>
```

XPATH

```
/College/*[ends-with(local-name(),"Ball")]
```

or

```
//*[ends-with(local-name(),"Ball")]
```

OUTPUT

```
<FootBall>
  <Members>20</Members>
  <Coach>Archie Theron</Coach>
  <Name>Wild cats</Name>
  <StarPlayer>David Perry</StarPlayer>
</FootBall>
<VolleyBall>
  <Members>24</Members>
  <Coach>Tim Jose</Coach>
  <Name>Avengers</Name>
  <StarPlayer>Lindsay Rowen</StarPlayer>
</VolleyBall>
<FoosBall>
  <Members>22</Members>
  <Coach>Rahul Mehra</Coach>
  <Name>Playerz</Name>
  <StarPlayer>Amanda Ren</StarPlayer>
</FoosBall>
```

Search for nodes with name light (case insensitive)

XML

```
<Galaxy>
    <Light>sun</Light>
    <Device>satellite</Device>
    <Sensor>human</Sensor>
    <Name>Milky Way</Name>
</Galaxy>
```

XPATH

```
/Galaxy/*[lower-case(local-name())="light"]
```

or

```
//*[lower-case(local-name())="light"]
```

OUTPUT

```
<Light>sun</Light>
```

Read Select nodes with names equal to or containing some string online: https://riptutorial.com/xpath/topic/3095/select-nodes-with-names-equal-to-or-containing-some-string

Credits

S. No	Chapters	Contributors
1	Getting started with xpath	Community, hielsnoppe, kjhughes, Vinay, Wolfgang Schindler
2	Check if a node is present	suchitra nair
3	Check if a node's text is empty	suchitra nair
4	Find nodes that have a specific attribute	Keith Hall, miken32, suchitra nair
5	Finding elements containing specific attributes	Keith Hall
6	Finding elements containing specific text	Keith Hall
7	Get nodes relative to the current node	suchitra nair
8	Get the count of nodes	suchitra nair
9	Location paths and axes	nwellnhof
10	Namespaces	4444, Keith Hall
11	Select nodes based on their children	Matthew
12	Select nodes with names equal to or containing some string	Dimitre Novatchev, suchitra nair