CHAPTER 1

Little Journey into XPath

XPath is the de factor standard language to represent queries to identify nodes in an xml structure. In this chapter we will go through the main concepts and show some of the way we can access nodes in a xml document. All the expressions can be executed on the spot so do not hesitate to experiment with them.

1.1 Getting started

You should load the XML parser and XPath library as follows:

```
Gofer it
   smalltalkhubUser: 'PharoExtras' project: 'XMLParserHTML';
   configurationOf: 'XMLParserHTML';
   loadStable.
```

```
Gofer it
   smalltalkhubUser: 'PharoExtras' project: 'XPath';
   configurationOf: 'XPath';
   loadStable.
```

1.2 An example

As an example we will take the possible representation of Magic cards. Here is for example how we can represent Arcane Lighthouse that you can see at http://gatherer.wizards.com/Pages/Card/Details.aspx?multiverseid=389430 and is shown in Figure 1-1.

```
  <?xml version="1.0" encoding="UTF-8"?>
```

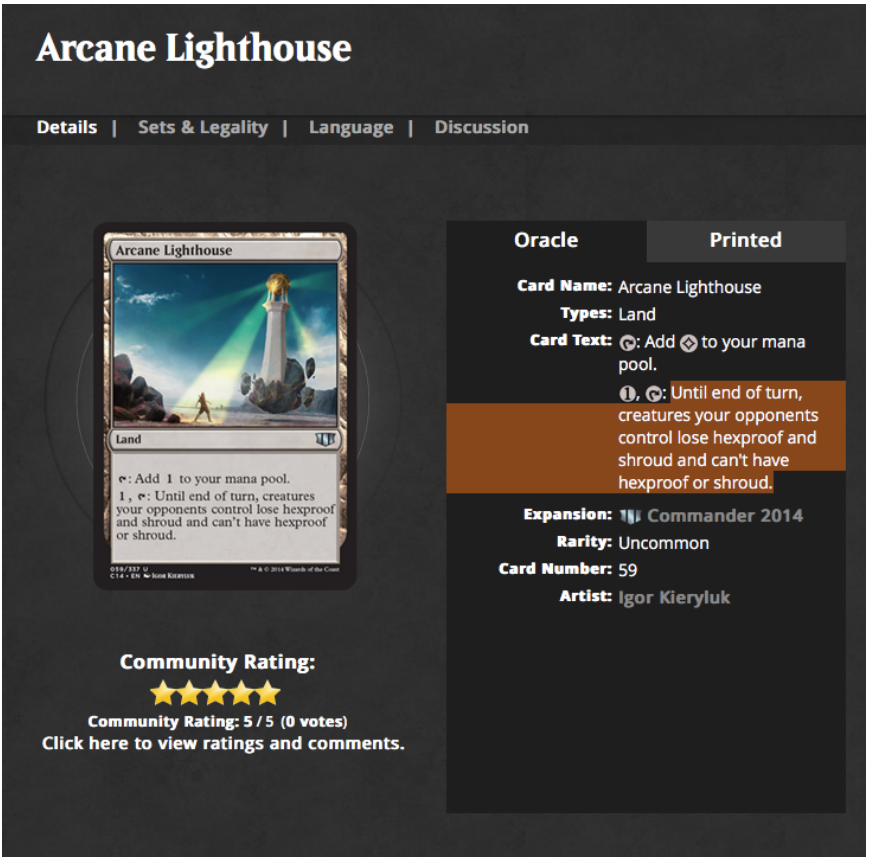

Figure 1-1 http://gatherer.wizards.com/Pages/Card/Details.aspx?multiverseid=389430.

```
<cardset>
  <card>
    <cardname lang="en">Arcane Lighthouse</cardname>
    <types>Land</types>
    <year>2014</year>
  <rarity>Uncommon</rarity>
  <expansion>Commander 2014</expansion>
    <cardtext>Tap: Add 1 uncolor to you mana pool.
  1 uncolor + Tap: Until end of turn, creatures your opponents
   control lose hexproof and shroud and can't have
   hexproof or shroud.</cardtext>
  </card>
</cardset>
```

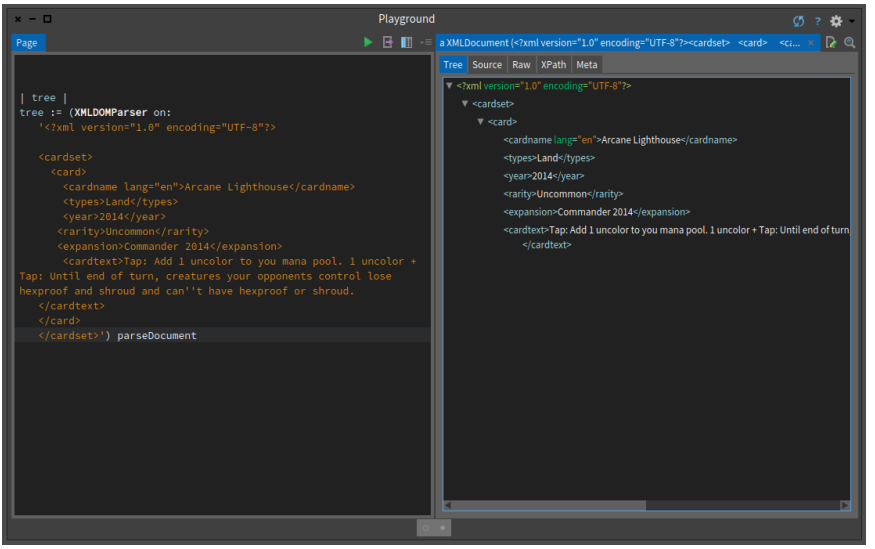

Figure 1-2 Grabbing and playing with a tree.

1.3 Accessing a tree object

In Pharo it is always powerful to get an object and interact with it. So let us do that now using the `XMLDOMParser`. Note that the escaped the `'` with an extra quote as in `can''t`.

```
| tree |
tree := (XMLDOMParser on:
'<?xml version="1.0" encoding="UTF-8"?>

<cardset>
  <card>
    <cardname lang="en">Arcane Lighthouse</cardname>
    <types>Land</types>
    <year>2014</year>
    <rarity>Uncommon</rarity>
    <expansion>Commander 2014</expansion>
    <cardtext>Tap: Add 1 uncolor to you mana pool.
  1 uncolor + Tap: Until end of turn, creatures your opponents
   control lose hexproof and shroud and can''t have
   hexproof or shroud.</cardtext>
  </card>
</cardset>') parseDocument
```

1.4 Nodes and atomic values

The following elements are nodes:

```
<cardset> (root element node)

<cardname lang="en">Arcane Lighthouse</cardname> (element node)

lang="en" (attribute node)
```

Atomic values are nodes with no children or parent. Here are some examples of atomic values:

```
Arcane Lighthouse

"en"
```

1.5 Basic tree relationships

Since we are talking about trees, nodes can have multiple relationships with each other: parent, child and siblings. Let us set some simple vocabulary.

- **Parent.** Each element and attribute has one parent. In the Arcane Lighthouse example, the card element is the parent of the cardname, types, year, rarity, expansion and cardtext.
- **Children.** Element nodes may have zero, one or more children. cardname, types, year, rarity, expansion and cardtext nodes are all children of the card element
- **Siblings.** Siblings are nodes that have the same parent. cardname, types, year, rarity, expansion and cardtext nodes are all siblings.
- **Ancestors.** A node's parent, parent's parent, etc. Ancestors of the cardname element are the card element and the cardset nodes.
- **Descendants** A node's children, children's children, etc. Descendants of the cardset element are the card,cardname, types, year, rarity, expansion and cardtext elements.

1.6 A large example

Let us expand our example to have cover more cases.

```
| tree |
tree := (XMLDOMParser on:
'<?xml version="1.0" encoding="UTF-8"?>

<cardset>
```

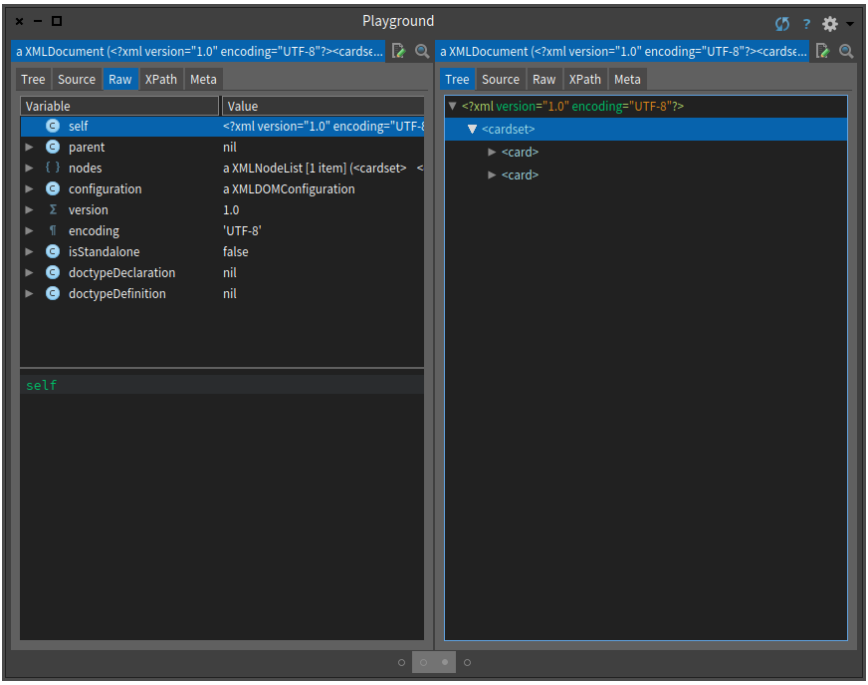

Figure 1-3 Select the raw tab and click on self in the inspector.

```
  <card>
    <cardname lang="en">Arcane Lighthouse</cardname>
    <types>Land</types>
    <year>2014</year>
    <rarity>Uncommon</rarity>
    <expansion>Commander 2014</expansion>
    <cardtext>Tap: Add 1 uncolor to you mana pool.
  1 uncolor + Tap: Until end of turn, creatures your opponents
   control lose hexproof and shroud and can''t have
   hexproof or shroud.</cardtext>
  </card>
  <card>
    <cardname lang="en">Desolate Lighthouse</cardname>
    <types>Land</types>
    <year>2013</year>
    <rarity>Rare</rarity>
    <expansion>Avacyn Restored</expansion>
    <cardtext>Tap: Add Colorless to your mana pool.
1BlueRed, Tap: Draw a card, then discard a card.</cardtext>
  </card>
</cardset>') parseDocument
```

Select the raw tab and click on self in the inspector (as shown in Figure 1-3). Now we are ready to learn XPath.

1.7 Node selection

The following table shows the XPath expressions.

Expression	Description
nodename	Selects all nodes with the name ”nodename”
/	Selects from the root node
//	Selects any node from the current node that match the selection
.	Selects the current node
..	Selects the parent of the current node
@	Selects attributes

In the following we expect that the variable `tree` is bound the full document tree we previously created parsing the xml string. In Pharo expressions selecting nodes returns set of nodes. Now let us play with the system to really see how it works.

Node tag name

nodename	Selects all nodes with the name ”nodename”
card	Selects all nodes with the name ”card”

Current and parent

.	Selects the current node
..	Selects the parent of the current node

The following expression shows that `.` (period) selects the current node.

```
(tree xpath: '.') first == tree
>>> true
```

Matching path based children nodes

The operator / selects from the root node.

/	Selects from the root node
/cardset	Selects the root element cardset
cardset/card	Selects all the card nodes that are children of cardset

The following expression selects all the card nodes under cardset node.

```
path := XPath for: '/cardset/card'.
path in: tree.
```

It is equivalent to the following expression using the `xpath:` message

```
tree xpath: '/cardset/card'
```

Matching deep nodes

The // operation selects all the nodes matching the selection.

//	**Selects any node from the current node**
//year	Selects all year nodes in all the children of the current node
cardset//year	Selects all year nodes that are descendant of cardset

Let us try with another element such as the expansion of a card.

```
tree xpath: '//expansion'
>>>
a XPathNodeSet(<expansion>Commander 2014</expansion>
    <expansion>Avacyn Restored</expansion>)
```

In Pharo you can also send message to the node. So the previous expression can be expressed as follows using the message //:

```
tree // 'expansion'
>>>
a XPathNodeSet(<expansion>Commander 2014</expansion>
    <expansion>Avacyn Restored</expansion>)
```

Identifying attributes

@ matches attributes.

Expression	**Description**
@	Selects attributes
//@lang	Selects all attributes that are named lang

The following expression returns all the attributes whose name is `lang`.

```
(tree xpath: '//@lang')
>>>  a XPathNodeSet(lang=""en"" lang=""en"")
```

1.8 Predicates

Predicates are used to find a specific node or a node that contains a specific value. Predicates are always embedded in square brackets.

Let us study some examples:

First element

The following expression selects the first card child of the cardset element.

```
tree xpath: '/cardset/card[1]'
>>>
a XPathNodeSet(<card>
    <cardname lang=""en"">Arcane Lighthouse</cardname>
    <types>Land</types>
    <year>2014</year>
    <rarity>Uncommon</rarity>
    <expansion>Commander 2014</expansion>
    <cardtext>Tap: Add 1 uncolor to you mana pool.
  1 uncolor + Tap: Until end of turn, creatures your opponents
   control lose hexproof and shroud and can't have
   hexproof or shroud.</cardtext>
  </card>)
```

In the XPath Pharo implementation the message `??` can be used for position or block predicates.

the previous expression is equivalent to the following one

```
tree / 'cardset' /'card' ?? 1 .
```

Block or position predicates can be applied with `??` to axis node test arguments or to result node sets.

The following expression returns the first element of each 'card' descendant:

```
tree // 'card' / ('*' ?? 1)
>>> "a XPathNodeSet(<cardname lang=""en"">Arcane
    Lighthouse</cardname> <cardname lang=""en"">Desolate
    Lighthouse</cardname>)"
```

Other position functions

The following expression selects the last card node that is the child of the cardset node.

```
tree xpath: '/cardset/card[last()]'.
```

The following selects the last but one node, in our case since we only have two elements we get the first.

```
tree xpath: '/cardset/card[last()-1]'.
>>>
a XPathNodeSet(<card>
      <cardname lang=""en"">Arcane Lighthouse</cardname>
      <types>Land</types>
      <year>2014</year>
      <rarity>Uncommon</rarity>
      <expansion>Commander 2014</expansion>
```

```
      <cardtext>Tap: Add 1 uncolor to you mana pool.
    1 uncolor + Tap: Until end of turn, creatures your opponents
     control lose hexproof and shroud and can't have
     hexproof or shroud.</cardtext>
    </card>)
```

We can also use the position function and use it to identify nodes. The following selects the first two card nodes that are children of the cardset node.

```
(tree xpath: '/cardset/card[position()<3]') size = 2
>>> true
```

Selecting based on node value

In addition we can select nodes based on a value of a node. The following query selects all the card nodes (of the cardset) that have a year greater than 2014.

```
tree xpath: '/cardset/card[year>2013]'.
```

The following query selects all the cardname nodes of the card children of cardset that have a year greater than 2014.

```
/cardset/card[year>2013]/cardname
>>> a XPathNodeSet(<cardname lang=""en"">Arcane
    Lighthouse</cardname>)
```

Selecting nodes based on attribute value

We can also select nodes based on the existence or value of an attribute. The following expression returns the cardname that have the lang attribute and whose value is 'en'.

```
tree xpath: '//cardname[@lang]
>>> a XPathNodeSet(<cardname lang=""en"">Arcane
    Lighthouse</cardname> <cardname lang=""en"">Desolate
    Lighthouse</cardname>)
tree xpath: '//cardname[@lang='en']
```

Note that we can simply get the card from the name using '..'.

```
tree xpath: '//cardname[@lang='en']/..
>>>
```

1.9 Selecting Unknown Nodes

In addition we can use wildcard to select any node.

Wildcard	Description
	Matches any element node
@*	Matches any attribute node
node()	Matches any node of any kind

For example `//*` selects all elements in a document.

```
(tree xpath: '//*') size
>>> 15
```

While `//@*` selects all the attributes of any node.

```
tree xpath: '//@*'
>>> a XPathNodeSet(lang=""en"" lang=""en"")
```

For example `//cardname[@*]` selects all cardname elements which have at least one attribute of any kind.

```
tree xpath: '//cardname[@*]'
>>> a XPathNodeSet(<cardname lang=""en"">Arcane
    Lighthouse</cardname> <cardname lang=""en"">Desolate
    Lighthouse</cardname>)
```

The following expression selects all child nodes of cardset.

```
tree xpath: '/cardset/*'.
```

The following expression selects all the cardname of all the child nodes of cardset.

```
tree xpath: '/cardset/*/cardname'.
```

1.10 Handling multiple queries

By using the | operator in an XPath expression you can select several paths. The following expression selects both the cardname and year of card nodes located anywhere in the document.

```
tree xpath: '//card/cardname | //card//year'
>>> a XPathNodeSet(<cardname lang=""en"">Arcane
    Lighthouse</cardname> <year>2014</year>
<cardname lang=""en"">Desolate Lighthouse</cardname>
    <year>2013</year>)"
```

1.11 XPath axes

Xpath introduces another way to select nodes using *location step* following the syntax: `axisname::nodetest[predicate]`. Such expressions can be used in the steps of location paths (see below).

An axis defines a node-set relative to the current node. Here is a table of the available axes.

AxisName	Result
ancestor	Selects all current node ancestors
ancestor-or-self	... and the current node itself
attribute	Selects all current node attributes
child	Selects all current node children
descendant	Selects all current node descendants
descendant-or-self	... and the current node itself
following	Selects everything after the current node closing tag
following-sibling	Selects all siblings after the current node
namespace	Selects all current node namespace nodes
parent	Selects current node parent
preceding	Selects all nodes that appear before the current node except ancestors, attribute nodes and namespace nodes
preceding-sibling	Selects all siblings before the current node
self	Selects the current node

Paths

A location path can be absolute or relative. An absolute location path starts with a slash (/) (/step/step/...) and a relative location path does not (step/step/...). In both cases the location path consists of one or more location steps, each separated by a slash.

Each step is evaluated against the nodes in the current node-set. A location step, `axisname::nodetest[predicate]`, consists of:

- an axis (defines the tree-relationship between the selected nodes and the current node)
- a node-test (identifies a node within an axis)
- zero or more predicates (to further refine the selected node-set)

The following example access the year node of all the children of the cardset.

```
tree xpath: '/cardset/child::node()/year').
>>>a XPathNodeSet(<year>2014</year> <year>2013</year>)
```

The following expression gets the ancestor of the year node and selects the cardname.

```
(tree xpath: '/cardset/card/year') first  xpath:
    'ancestor::card/cardname'
>>> "a XPathNodeSet(<cardname lang=""en"">Arcane
    Lighthouse</cardname>)"
```

1.12 Conclusion

XPath is a powerful language and the Pharo XPath library developed and maintained by Monty Kamath is implementing the full standard 1.0. In addition coupled with life programming capabilities of Pharo it gives a really powerful to explore structured data.