Implementing Closures in XSLT/XPath 3.0

Roger L. Costello
November 2012

Closures are really cool and really powerful. In this paper I describe what closures are, why they are important, and how to implement them in XSLT 3.0 and XPath 3.0.

Functions as first-class values

Before learning about closures it is important to understand *higher-order functions*. In almost all programming languages you can create a function and pass to it an integer or a string (or some other data type). You can create functions that return an integer or a string. And you can assign to a variable an integer or a string. Thus, integers and strings are *first-class values*. For a long time many programming languages did not allow you to pass a function to a function nor create a function that returned a function nor assign a function to a variable. Thus functions were treated as *second-class values*. But many modern programming languages have elevated functions to first-class values. In fact, XSLT 3.0 and XPath 3.0 have done so. XSLT 3.0 and XPath 3.0 now allow you to pass functions to a function, create functions that return a function, and assign a function to a variable.

Functions that manipulate functions are called *higher-order functions*.

Let's see how to implement higher-order functions using XSLT 3.0 and XPath 3.0.

Example: the following variable holds the altitude of an aircraft in feet:

```
<xsl:variable name="altitude-in-feet" select="1200" />
```

I would like to output this value in other units such as meters and yards. So I created an anonymous function that is invoked by passing to it a value and a function:

```
function(
            $value as xs:decimal,
            $f as function(item()*) as item()*
        )
```

It returns a decimal:

```
function(
                $value as xs:decimal,
                $f as function(item()*) as item()*
        )
        as xs:decimal
```

What does it do? Answer: it applies $f to $value:

```
function(
                $value as xs:decimal,
                $f as function(item()*) as item()*
        )
        as xs:decimal
  {$f($value)}
```

We have created a higher-order function.

Cool!

Now let's assign it to a variable:

```
<xsl:variable name="unit-converter" select="function(
                                                $value as xs:decimal,
                                                $f as function(item()*) as item()*
                                            )
                                            as xs:decimal
                                        {$f($value)}" />
```

So $unit-converter is a function that takes two arguments:

(1) a decimal value

(2) a function

and it applies the function to the value.

Let's create another function. This is a simple function that converts feet to meters:

```
<xsl:variable name="feet-to-meters" select="function(
                                                $a as xs:decimal
                                            )
                                            as xs:decimal
                                        {$a * 0.3048}" />
```

It takes a decimal argument and multiplies it by 0.3048.

Now let's convert $altitude-in-feet to meters and output the result:

```
<xsl:value-of select="$unit-converter($altitude-in-feet, $feet-to-meters)" />
```

$unit-converter is invoked with the value 1200 and a function that converts the value from feet to meters.

Okay, we can convert to meters; let's now create a simple function that converts feet to yards:

```
<xsl:variable name="feet-to-yards" select="function(
                                              $a as xs:decimal
                                            )
                                           as xs:decimal
                                         {$a * 3}" />
```

It takes a decimal argument and multiplies it by 3.

Now let's convert $altitude-in-feet to yards and output the result:

```
<xsl:value-of select="$unit-converter($altitude-in-feet, $feet-to-yards)" />
```

$unit-converter is invoked with the value 1200 and a function that converts the value from feet to yards.

Notice that the function $unit-converter can convert any value to any units by simply providing it the conversion function.

$unit-converter is a *higher-order function*.

Closures

Recall that a higher-order function can return a function. Suppose a function *f* is invoked and it returns a function *g*. And into that returned function *f* stores some of its local data. That returned function *g* is called a closure.

Definition: A closure is a function that was returned from another function and the returned function contains data from the returning function.

Let's take an example.

The following anonymous function is invoked by passing to it a greeting; it returns a function:

```
function(
            $greeting as xs:string
          )
         as function(xs:string) as xs:string
```

To make the returned function a closure we store data into the body of the returned function (in this case we store the value of $greeting into the returned function):

```
function(
            $greeting as xs:string
          )
         as function(xs:string) as xs:string
  {function($name as xs:string) as item() {$greeting || $name}}" />
```

Note: XPath 3.0 has a new operator, ||, for concatenating two strings.

Look at the body of the returned function (i.e., everything between the curly braces). Read it as: return a function that takes one argument, $name, and it concatenates $greeting with $name. The body of the returned function *up-references* $greeting. Thus, $greeting is kept alive after the function returns.

Wow!

Let's assign the whole thing to a variable:

```
<xsl:variable name="f" select="function(
                                    $greeting as xs:string
                                  )
                                  as function(xs:string) as xs:string
                               {function($name as xs:string) as item() {$greeting || $name}}" />
```

We invoke the function $f with a string:

```
$f('Hello: ')
```

As we've seen, $f returns this function:

```
function($name as xs:string) as item() {$greeting || $name}
```

That is a closure -- the body is up-referencing a value from $f.

That is so cool.

Let's assign the returned function to a variable:

```
<xsl:variable name="sayHello" select="$f('Hello: ')" />
```

Remember, $sayHello is a function and we invoke it by passing to it a name. So let's pass to it "John" and then output the result:

```
<xsl:value-of select="$sayHello('John')" />
```

Output: Hello: John

We invoked $sayHello with a simple name and it operated on it using data from $f. This was accomplished using a closure.

Here is the code in its entirety:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:xs="http://www.w3.org/2001/XMLSchema"
                version="3.0">

   <xsl:output method="text"/>

   <xsl:variable name="f" select="function(
                                        $greeting as xs:string
                                    )
                                    as function(xs:string) as xs:string
                            {function($name as xs:string) as item() {$greeting || $name}}" />

   <xsl:variable name="sayHello" select="$f('Hello: ')" />

   <xsl:template match="Name">
      <xsl:value-of select="$sayHello('John')" />
   </xsl:template>

</xsl:stylesheet>
```

In the example, the data inserted into the closure was a hardcoded string. Let's create a closure that contains data from an XML document. That's next.

How to create a function that has context, without passing it context

Suppose you would like to create a function that, given the title of a book, it returns the author.

For example, the function call:

```
author('The Society of Mind')
```

returns the author:

```
Marvin Minsky
```

Notice that the function, author(), was just provided a string and no context. How can the function obtain the author without any context? Recall that functions supposedly have no context and you must give it (through parameters) all the context it needs.

I show how to create functions that have context, but you don't have to give it the context.

It is accomplished using closures.

Here is the XML document that the function will operate on:

```
<Books>
  <Book>
    <Title>Six Great Ideas</Title>
    <Author>Mortimer J. Adler</Author>
  </Book>
  <Book>
    <Title>The Society of Mind</Title>
    <Author>Marvin Minsky</Author>
  </Book>
</Books>
```

I create a variable, author, that is defined to be an XPath 3.0 anonymous function:

```
<xsl:variable name="author" select="... define author as an anonymous function ... " />
```

Then I invoke the "author function" with a string representing the title of a book:

```
<xsl:value-of select="$author('The Society of Mind')" />
```

Output: Marvin Minsky

This is key:

> *Create a variable that is defined to be a function that returns a function and*
>
> *store context into the returned function.*

Let's step through this very carefully.

I create a variable, root, to which I pass the root element of the XML document (Books):

```
<xsl:variable name="root" select="function(
                                        $root_ as element(Books)
                                    )
```

The function returns a function:

```
<xsl:variable name="root" select="function(
                                        $root_ as element(Books)
                                    )
                                    as function(xs:string) as item()
```

The function that is returned takes an argument that is a string representing the title of a book and it returns the author of the book:

```
<xsl:variable name="root" select="function(
                                        $root_ as element(Books)
                                    )
                                    as function(xs:string) as item()
                        {function($title as xs:string) as item() {$root_/Book[Title eq $title]/Author}}" />
```

Observe that the returned function is up-referencing the <Books> element (which is in the variable $roots_).

The returned function is a closure.

Remember the author variable? It is assigned the returned function:

```
<xsl:variable name="author" select="$root(/Books)" />
```

Okay, let's put it all together:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                 xmlns:xs="http://www.w3.org/2001/XMLSchema"
                 version="3.0">

   <xsl:output method="text"/>

   <xsl:variable name="root" select="function(
                                          $root_ as element(Books)
                                        )
                                        as function(xs:string) as item()
                        {function($title as xs:string) as item() {$root_/Book[Title eq $title]/Author}}" />

   <xsl:variable name="author" select="$root(/Books)" />

   <xsl:template match="Books">
      <xsl:value-of select="$author('The Society of Mind')" />
   </xsl:template>

</xsl:stylesheet>
```

Acknowledgements

Thanks to the following people who contributed to this paper:

- Michael Kay
- Dimitre Novatchev
- Liam Quinn