Current Node vs Context Node

The current node and the context node are different things. Most of the time they evaluate to the same thing, but sometimes they don't.

`.` actually refers to the context node rather than the current node, but every time we've used it until now, the context node was the same thing as the current node.

The function called `current()` refers to the current node.

The Current Node

When the XSLT processor instantiates a template for some node in the document tree, that node becomes the current node for the entire execution of that template.

The system maintains a stack of current nodes, which starts out empty. When a template is instantiated for a node, that node is added to the stack of current nodes. When a template is finished, that current node is removed from the stack.

`current()` always refers to the top current node reference on the stack.

Example Usage

```
[…]
<xsl:template match="person">
    <p>
        I found a person called <xsl:value-of
select="current()"/>
    </p>
</xsl:template>
[…]
```

The Context Node

When an XPath expression is being evaluated, the current node remains fixed, but the context node chages.

At the start of evaluating an XPath expression, the context node is the same as the current node. (So `<xsl:value-of select="."/>` is the same as `<xsl:value-of select="current()"/>`).

The Changing Context Node

```
[…]
<xsl:apply-templates
select=".//person[./corp/@hair='white']"/>
[…]
```

The first dot refers to the current node, as the context node is the same as the current node when evaluation of an XPath expression starts.

The second dot refers to the person node we are currently considering while compiling the node list. This is because the predicate is checked against each person node.

Example

```
[…]
<people>
    <person>
        <name>Celia</name>
        <corp eyes="blue" hair="red"/>
    </person>
    <person>
        <name>Tom</name>
        <corp eyes="blue" hair="white"/>
    </person>
    <person>
        <name>Mick</name>
        <corp eyes="black" hair="white"/>
    </person>
</people>
```

In the example, the first dot always refers to the root node, while the second dot applies to each of the person elements in turn. The current node, however, is unchanging until a new template is called.

Example 2

```
[…]
<nation name="Irish" typicalHairColour="red">
    <person>
        <name>Celia</name>
        <corp eyes="blue" hair="red"/>
    </person>
    <person>
        <name>Tom</name>
        <corp eyes="green" hair="red"/>
    </person>
    <person>
        <name>Mick</name>
        <corp eyes="black" hair="black"/>
    </person>
</nation>
[…]
```

We want to check how many people in the nation have the typical hair colour. Here's the stylesheet:

```
[…]
<xsl:template match="nation">
    <xsl:value-of
select="count(./person[./corp/@hair=current()/@typicalHairCo
lour])"/>
</xsl:template>
[…]
```

The `value-of` instruction here is counting the number of person nodes in the nation element whose hair attribute of their corp child is the same as the typicalHairColour attribute of their nation parent. It's checking how many people in the Irish nation have red hair.

Since two children have hair colour red, which is the typical hair colour of the nation element, this `value-of` will give 2.

Here the `current()` node is the Irish `nation` node, while this template has been called to process that `nation` node.

`for-each` Loops and `current()`

With a `for-each` statement, the meaning of `current()` changes.

1. Inside the open `for-each` tag (e.g. in the "select" attribute):
   - `current()` refers to the node matched for the current template.
2. In between the open and close `for-each` tags (in the body of the loop):
   - `current()` refers to the node in the nodelist which is looked at in this loop iteration.

Example

```
[…]
<xsl:template match="/">
    <xsl:for-each
select="current()//person[./corp/@hair="white"]">
        <p>The name of this white-haired person is:</p>
        <p>
            <value-of select="current()/name"/>
        </p>
    </xsl:for-each>
</xsl:template>
[…]
```

In this example:

- Within the "select" attribute of the `for-each` statement, `current()` refers to the root node, as that's the node matched by template.
  - These are the instructions for compiling the nodelist for the `for-each` statement.
- Within the body of the `for-each` loop, `current()` refers to the node in the `for-each` nodelist that we are currently processing.
  - In this example, the nodelist will contain only people with white hair, so `current()` inside the `for-each` loop refers to the white-haired person we're currently looking at.

Example 2

```
<xls:template match="/">
[…]
<xsl:for-each select="current()/nations/nation">
    <xsl:for-each
select="current()/person[./corp/@hair=current()/@typicalHair
Colour]">

        <xsl:value-of select="current()/name"/>
    </xsl:for-each>
</xsl:for-each>
</xsl:template>
```

1. The first usage of `current()` is in the "select" attribute of the *outer* `for-each` loop. It refers to the root node, as the root node is the node that matched this template.
2. The second usage of `current()` is in the "select" attribute of the *inner* `for-each` loop. It refers to the nation node we're currently processing, as we're inside the outer `for-each` loop.
3. The third usage of `current()` is in the "select" attribute of the `value-of`

statement. It refers to the person node we're currently processing, as we're inside the inner `for-each` loop.

Using `current()` for Cross-Referenced Sorting

```
<xsl:for-each select=".//country">
<xsl:sort
select="/world/languages/language[./@id=current()/@language]
/@name"/>
[…]
</xsl:for-each>
```

Using `.` Only When We Need To

The above code can also be written as:

```
<xsl:for-each select="//country">
<xsl:sort
select="/world/languages/language[@id=current()/@language]/@
name"/>
[…]
</xsl:for-each>
```