4-29-2015

XSD: The Path From Excel to XML: The Basics: Mapping Elements and Attributes

Amelia Mowry
Wayne State University, amelia.mowry@wayne.edu

XSD: THE PATH FROM EXCEL TO XML

The basics: Mapping Elements and Attributes

By Amelia Mowry, M.S.I.S
Metadata & Discovery Services Librarian
Wayne State University
Detroit, Michigan
Amelia.mowry@wayne.edu
@TheMetaDiva

INTRODUCTION

Working with metadata often means moving it between various formats. One problematic move to make is from a spreadsheet to an XML file. This paper will describe the basics of using an XSD to map a Microsoft Excel spreadsheet to XML.

Moving metadata can be a messy process, especially when the data is older. Data often comes out of legacy systems as csv files, tab-delimited files, or Excel files. Moving data from these formats into XML can prove to be a challenge. I have also found it beneficial to do some initial data cleaning in spreadsheet form. This allows for the use of Microsoft Excel tools such as sorting, filtering, and vertical lookups to identify potential problems. Having metadata in a spreadsheet also allows for the use of other metadata tools, such as Open Refine and EMET.

This paper will describe the basics of using XSD to map data from an Excel spreadsheet to an XML file. It will cover setting up Excel for XML mapping and creating XSDs that map elements and attributes.

This paper is based on a workshop I gave at the 2014 Great Lakes THATCamp at Lawrence Tech in Southfield, Michigan in September 2014.

Important notes:

1. This process is not supported in Microsoft Office for Mac.
2. Spreadsheets can be mapped to XML in Open Office, but that process is based on XSLT.
3. The XML editor used in the examples is jEdit. It is free and can be downloaded from http://www.jedit.org

How Mapping Data in Excel Works: The Birds-Eye View

We'll start with a very high-level view of how an XSD words in this process. The fundamental problem with transforming a spreadsheet to XML is that spreadsheets are flat while XML is hierarchical. At the very basic level, the XSD is building a map for how the content in the spreadsheet fits into the hierarchical XML structure.

For example, the data in the title column of spreadsheet A goes into an XML element <mods:title>.

Title	Subject - Name
1958 Packard hawk, left side view, parked on drive	Packard Motor Car Company
1958 Packard hawk, three-quarter left side view, on display	Packard Motor Car Company
1958 Packard hawk, three-quarter front view	Packard Motor Car Company
1958 Packard sedan, left side view, parked on drive	Packard Motor Car Company
1958 Packard sedan, nine-tenths left side view, parked in drive\	Packard Motor Car Company
1958 Packard sedan, nine-tenths left side view, parked on drive	Packard Motor Car Company

Spreadsheet Example A

However, <mods:title> is itself contained in the element <mods:titleInfo> which is in the element, <mods:mods>, which is in turn in the root element <mods:modsCollection>.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<mods:modsCollection xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        <mods:mods version="3.4">
                <mods:titleInfo>
                        <mods:title>1958 Packard hawk, left side view, parked on drive</mods:title>
                </mods:titleInfo>
                <mods:subject>
                        <mods:name type="corporate" authority="naf">
                                <mods:namePart>Packard Motor Car Company</mods:namePart>
                        </mods:name>
                </mods:subject>
```

XML Example A

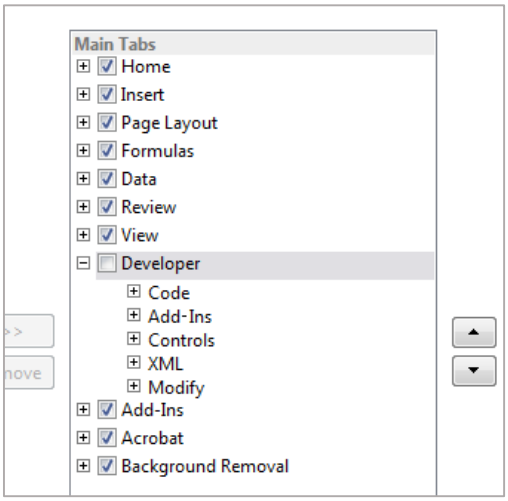
The XSD defines the structure of the XML file to be created and identifies where the data from the columns are to be mapped. The XSD allows for the creation of elements that are not in the spreadsheet but that hold elements. In the XSD example below, those "holding" or container elements are marked with yellow stars. The XSD also includes an element that will contain data mapped from the columns in the spreadsheet. That element is circled in red in the example.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="modsCollection">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="mods" minOccurs="0" maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="mods">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="titleInfo">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="Title" type="xs:string"/>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
                <xs:element name="subject">
```

XSD Example A

Setting Up Excel for XML

In order to use XSD in Excel, you will have to add the developer tab. This is a simple procedure. First go to "File" and select "Options." From there, select "Customize Ribbon." You'll see a listing of the main tabs. By default, the developer tab is unchecked. Simply check it. When you return to the main Excel page, you will see the Developer tab along the top. You will use the source tool within this tab to map elements in your XSD to your spreadsheet.



Mapping Elements

To avoid confusion from namespaces and specific schema requirements, I'm going to work through a simple example that is not based on a particular standard. We're going to go over the mapping of elements based on a small spreadsheet about fruit. Our spreadsheet contains three columns, each of which will become an element.

However, we're not going to worry about those elements just yet. The first thing we need to do when creating an XSD is identify the "invisible" elements that hold the elements shown in the cells.

This is actually not a difficult thing to do. The data in the cells are not independent. For example, the three cells in row two are all related; they all provide details about a type of fruit called an apple, which is red and that grows on a tree. Each row of data represents one *thing*. In the case of this spreadsheet, each *thing* is a type of fruit. In library data, such as Spreadsheet Example A above, each row is often a record recording metadata about an individual item.

	A	B	C	D
1	name	color	growsOn	
2	apple	red	tree	
3	banana	yellow	tree	
4	plum	purple	tree	
5	pear	green	tree	
6	melon	orange	vine	
7	raspberry	red	bush	
8				
9				

Spreadsheet Example B

The other "invisible" element we have to worry about is what is represented by the entire spreadsheet, all the rows. If each individual row represents a type of fruit, all of the rows as a whole represent a group of fruits. Therefore the XML file created from this spreadsheet will look something like the XML snippet to the right.

```
<fruits>
   <fruit>
      <name>apple</name>
      <color>red</color>
      <growsOn>tree</growsOn>
   </fruit>
   <fruit>
      <name>banana</name>
   [… other elements …]
</fruits>
```

To create the XSD to map this spreadsheet, we start with these “holder” or container elements: <fruits> and <fruit>.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3      <xs:element name="fruits">
4          <xs:complexType>
5              <xs:sequence>
6                  <xs:element ref="fruit" minOccurs="0" maxOccurs="unbounded"/>
7              </xs:sequence>
8          </xs:complexType>
9      </xs:element>
10     <xs:element name="fruit">
11         <xs:complexType>
12             <xs:sequence>
13                 <xs:element name="" type="xs:string"/>
14             </xs:sequence>
15         </xs:complexType>
16     </xs:element>
17 </xs:schema>
```

The <fruits> element is declared in line three. It is followed in line four by <xs:complexType>. This means that this is an element that contains other elements or attributes. <xs:sequence>, which follows in line five, contains those elements. <fruits> only contains one element, <fruit>. Note the attributes *minOccurs=”0”* and *maxOccurs=”unbounded”.* This means that our XML file must have at least one <fruit> element but can have as many as we want. In the terms of our spreadsheet, we must have at least one row of data.

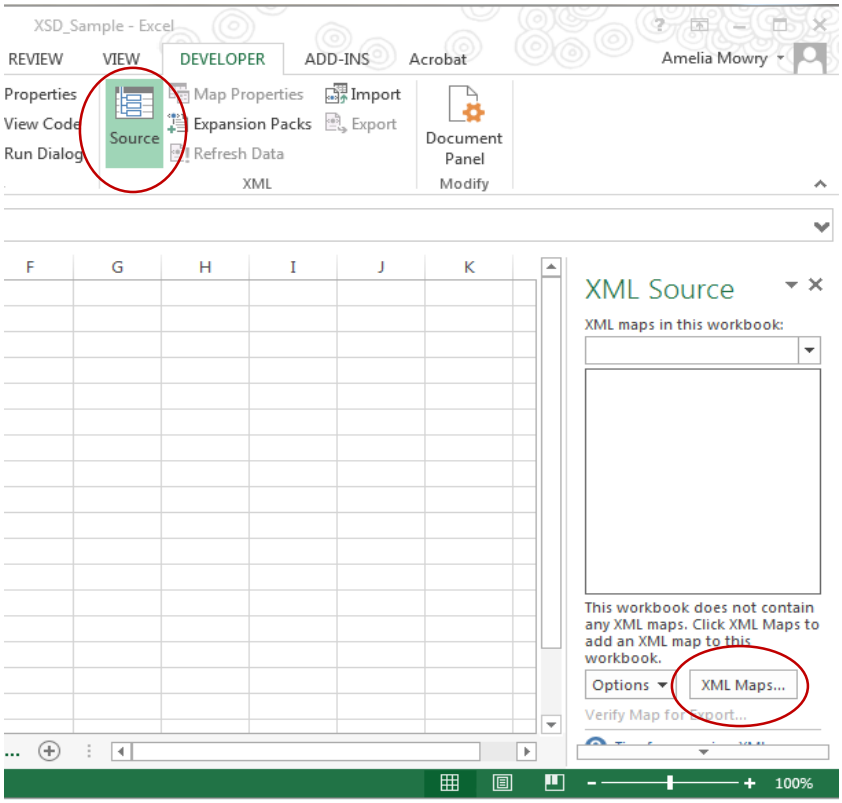
The other attribute on line six is *ref=”fruit”.* This means that the element contained within <fruits> is declared and described in the <xs:element> with the name *fruit*. The <xs:element> in line six is *referring* to the <xs:element> beginning on line ten. Note that <fruit> also includes an empty <xs:element> where we will add other elements. On line thirteen, we do not have a *ref* attribute. In this case, we will be putting the declarations for elements within the <fruit> element. XSDs can be constructed either way, with all the elements separate and referring to one another or with elements declared within their container elements. For this example, I’m including both methods. What you choose to do will depend on personal taste and the complexity of the XSD you are creating.

Now that the “invisible” elements from our spreadsheet are mapped, we can map the elements that represent the columns in our spreadsheet. These elements will be inside of our <fruit> element. They will be placed between the sequence tags, and the elements will look like <xs:element name=”name” type=”xs:string”/>

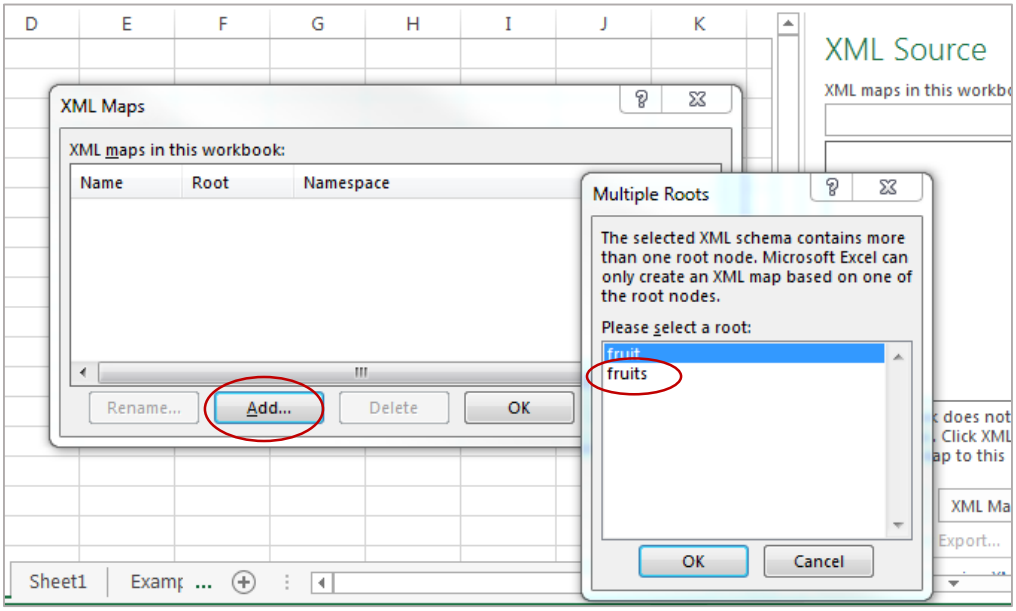
There are a few important things to note about these elements. First is that the *type* attribute is required for elements that will contain data from the spreadsheet. In most cases, you will want to use the type “xs:string”. This simply means that the data from the cells will transfer as is to the XML file. This will be covered in more detail later on. Second is that these elements end it “/>.” Elements must be closed or your file will not be valid. If your element does not contain other elements, you can use a self-closing tag, which is a tag that is closed at the end of the tag that opened it. It ends with “/>.” Our example XSD with the columns mapped is below.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="fruits">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="fruit" minOccurs="0" maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="fruit">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="name" type="xs:string"/>
                <xs:element name="color" type="xs:string"/>
                <xs:element name="growsOn" type="xs:string"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

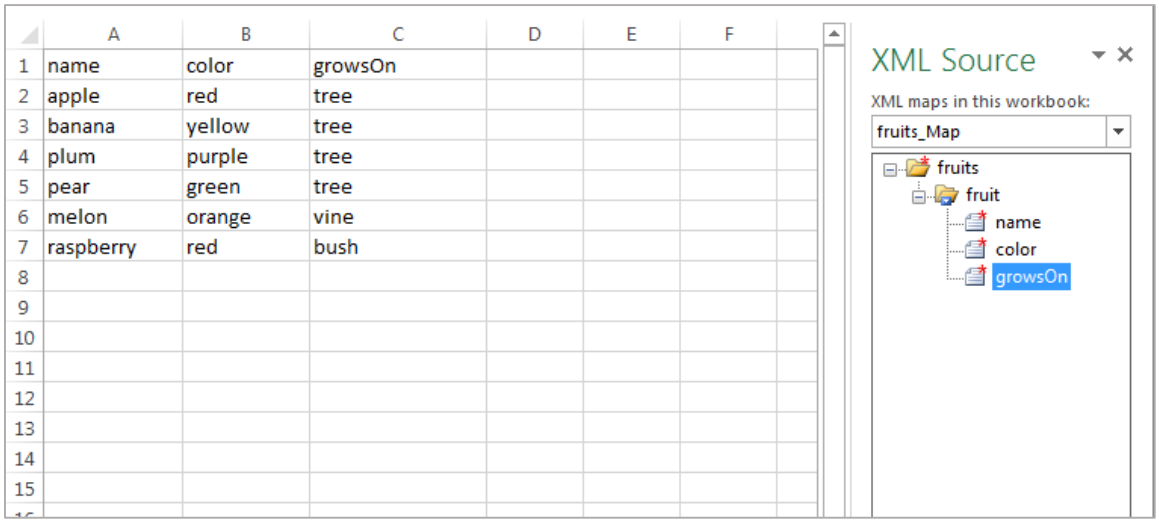
Once the XSD is created, save it. Make sure that you save it as an xsd. In jEdit, you can do this by simply writing the file as **[filename].xsd**. The next step is to map this XSD file to your Excel spreadsheet. Go to the Excel file and click on the developer tab, then go to source. This will add a panel to the right, which will allow you to add your XSD by clicking on "XML Maps" at the bottom right.



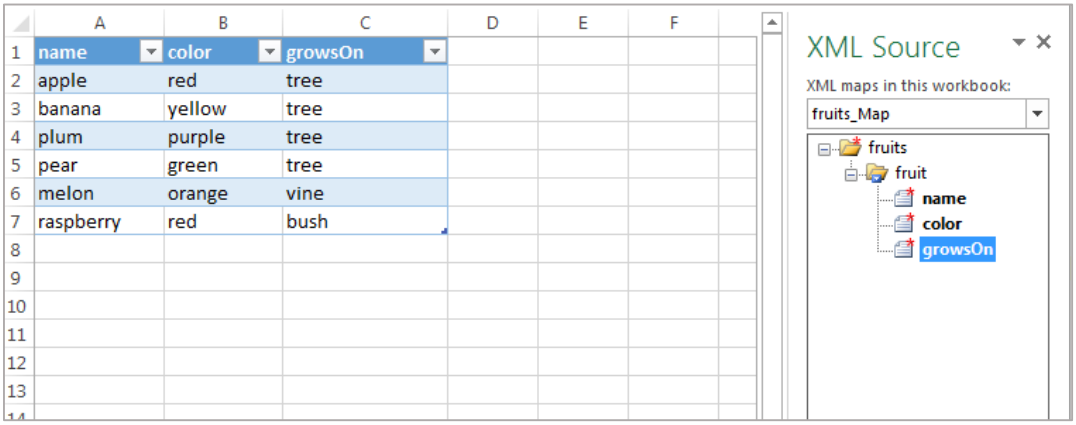
The XML Maps window will open. Click on “Add” and add your XSD. If there is something wrong with the XSD, an error message will pop up. If not, you’ll be asked to identify the root element of you XML file. This is element that contains all of the others; it will be the one that represents the spreadsheet as a whole. In our example, that will be fruits. When this is finished, press “OK.”



When the XSD has been successfully loaded, the map will be visible in the XML Source Pane. It is possible to automatically map your elements based on the column headings, but I have had that method fail. I prefer to simply drag the element names from the XML Source Pane to the appropriate column heading.



There are a few things known to cause issues with mapping the columns. First, keep your column headings simple, unique, and without special characters or spaces. Also try to map the columns in spreadsheet order. Sometimes columns won't map properly if they are surrounded by unmapped columns. Make sure the columns map properly as you go. They should be completely highlighted as shown below if the mapping worked.



When the XSD has been mapped, go to File > Save As. Change the "Save as Type" from an excel format to XML Data. Then simply save. The result should be an XML file that looks like the XML below.

More on Data Types

Earlier I wrote that any element that will be containing data as opposed to other elements must have an attribute giving the data type. Generally, this type will be "xs:string," which will simply add the content of the cell as text. There are many different data types that can be used, which can be found in Excel's documentation. ("XML Schema Definition") One case in which you may use a different data type would be if you have date information. Using this data type, you can select how you would like the date elements formatted in your XML, even if the dates are not consistently formatted in your spreadsheet.

D	E
growsOn	eaten
tree	Sep-14
tree	Aug-12
tree	Sep-14
tree	8/1/2014
vine	1-Jul
bush	Aug-14

As an example, I added another column to the fruit spreadsheet containing a (random) date I ate said fruit. Take note that the dates are formatted inconsistently. (Anyone who has worked with Excel knows that consistently formatting dates can be problematic.) Here we have some dates that have an abbreviated text month, either followed or preceded by the day. We also have a date made up all numerals. For this XML file, I elected to put all of the dates in the format YYYY-MM. To do that, we add the XSD element for the <eaten> element with the data type "xs:gYearMonth." ("XML Schema Definition")

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="fruits">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="fruit" minOccurs="0" maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="fruit">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="name" type="xs:string"/>
                <xs:element name="color" type="xs:string"/>
                <xs:element name="growsOn" type="xs:string"/>
                <xs:element name="eaten" type="xs:gYearMonth"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

This will result in an XML file that has all of the dates in this YYYY-MM format. See the XML snippet at the right.

```
<fruit>
        <name>apple</name>
        <color>red</color>
        <growsOn>tree</growsOn>
        <eaten>2014-09</eaten>
</fruit>
<fruit>
        <name>banana</name>
        <color>yellow</color>
        <growsOn>tree</growsOn>
        <eaten>2012-08</eaten>
</fruit>
```

Overall, you don't have to be too concerned about data types. If you attempt to map an element with a data type that does not match the format of the Excel column, Excel will alert you to the discrepancy and allow you to select your desired data type.

Mapping Attributes

Mapping Excel data to XML attributes is slightly more complicated than mapping elements. There are also differences depending on whether you are attempting to map an element that contains both an attribute and text, see Attribute Example A, or whether you are trying to map and element that contains only an attribute or an attribute and elements, but no text, such as Attribute Example B.

```
<mods:originInfo>
    <mods:place>
        <mods:placeTerm type="code" authority="marccountry">miu</mo
    </mods:place>
    <mods:dateIssued keyDate="yes" encoding="w3cdtf">1912</mods:dat
</mods:originInfo>
```

Attribute Example A

```
<mods:subject authority="lcsh">
    <mods:topic>Cadillac automobile</mods:topic>
    <mods:topic>History</mods:topic>
</mods:subject>
```

Attribute Example B

First, we will go over how to map attributes as shown in Example A, where the elements contain both content and attributes. In order to do this, we will make use of two new XSD elements, simpleContent and extension. SimpleContent is used when an elements contains only text or attributes. Extension is used in conjunction with simpleContent to define the content allowed in the element. ("XSD Text-Only Elements")

In our example, we will add a column for the scientific name of our fruits, as shown on the right. Rather than include this data as a new element, we are going to include it as an attribute of our name element.

	A	B	C	D	E	F
1	name	scientificName	color	growsOn	eaten	
2	apple	malus domestica	red	tree	Sep-14	
3	banana	musa acuminata	yellow	tree	Aug-12	
4	plum	prunus domestica	purple	tree	Sep-14	
5	pear	pyrus communis	green	tree	8/1/2014	
6	melon	cucumis melo	orange	vine	1-Jul	
7	raspberry	rubus idaeus	red	bush	Aug-14	
8						

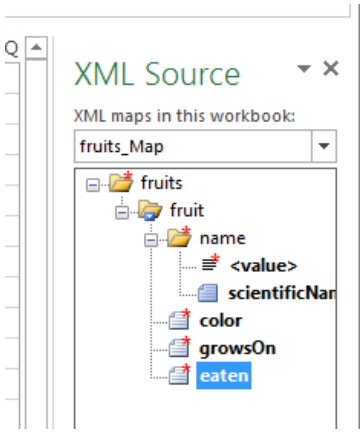
Then we will extend the content of our name element in our XSD as shown in the screenshot below. "Name" now becomes a complex element, like "fruits" and "fruit." This means that the tag is no longer self-closing. The element also cannot declare a type of textual content and contain an attribute, so the "name" element will now look like this: <xs:element name="name"> Do not forget to close this tag with </xs:element> as was done with "fruit" and "fruits."

Within the name element, add the tag <xs:complexType> to declare that this element contains other items. However, because this element only contains text and an attribute, we use the tag <xs:simpleContent> rather than the <xs:sequence> tags used in "fruit" and "fruits." Next, add the tag <xs:extension base="xs:string">. This tag extends our "name" element to allow for textual content. Note that this tag is not self-closing. The tag declaring the attribute is located within it. That tag is named <xs:attribute>. It is formatted the same as tags declaring elements, with an attribute for the name of the attribute and another attribute for the type of the content.

The "name" element in your finished XSD will look like the one in the screenshot to the right. The rest of your XSD will look the same. A common mistake here is for elements not to be closed or not to be closed in the correct order.

```
<xs:element name="fruit">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="name" >
                <xs:complexType>
                    <xs:simpleContent>
                        <xs:extension base="xs:string">
                            <xs:attribute name="scientificName" type="xs:string"/>
                        </xs:extension>
                    </xs:simpleContent>
                </xs:complexType>
            </xs:element>
            <xs:element name="color" type="xs:string" />
```

Add the XSD to your new spreadsheet. The mapping will look slightly different than it did previously. Apply <value> underneath name to the column representing the common fruit name and add "scientificName" to the column for the attribute. The resulting XML should look like the screenshot below.




```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<fruits>
    <fruit>
        <name scientificName="malus domestica">apple</name>
        <color>red</color>
        <growsOn>tree</growsOn>
        <eaten>2014-09</eaten>
    </fruit>
    <fruit>
        <name scientificName="musa acuminata">banana</name>
        <color>yellow</color>
```

There is a different process for adding attributes to elements that do not hold textual data. Let us say we want to add an attribute "id" to <fruit> that will specifying a unique identification number for each different fruit. Add the column as shown below.

	A	B	C	D	E	F
1	id	name	scientificName	color	growsOn	eaten
2	1	apple	malus domestica	red	tree	Sep-14
3	2	banana	musa acuminata	yellow	tree	Aug-12
4	3	plum	prunus domestica	purple	tree	Sep-14
5	4	pear	pyrus communis	green	tree	8/1/2014
6	5	melon	cucumis melo	orange	vine	1-Jul
7	6	raspberry	rubus idaeus	red	bush	Aug-14
8						

In our XSD, we will add our <xs:attribute> tag just above the </complexType> and below the </sequence> tag. Placement is important; the tag must be below any elements that are also contained in the element. If this element only contained an attribute, no elements, the <xs:attribute> tag would simply be declared between the <complexType> tags. (“Creating XML Mappings”)

```
    <xs:element name="fruit">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="name" >
                    <xs:complexType>
                        <xs:simpleContent>
                            <xs:extension base="xs:string">
                                <xs:attribute name="scientificName" type="xs:string"/>
                            </xs:extension>
                        </xs:simpleContent>
                    </xs:complexType>
                </xs:element>
                <xs:element name="color" type="xs:string" />
                <xs:element name="growsOn" type="xs:string" />
                <xs:element name="eaten" type="xs:gYearMonth" />
            </xs:sequence>
            <xs:attribute name="id" type="xs:int"/>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

The resulting XML is shown below:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<fruits>
    <fruit id="1">
        <name scientificName="malus domestica">apple</name>
        <color>red</color>
        <growsOn>tree</growsOn>
        <eaten>2014-09</eaten>
    </fruit>
    <fruit id="2">
        <name scientificName="musa acuminata">banana</name>
        <color>yellow</color>
        <growsOn>tree</growsOn>
        <eaten>2012-08</eaten>
    </fruit>
    <fruit id="3">
        <name scientificName="prunus domestica">plum</name>
        <color>purple</color>
        <growsOn>tree</growsOn>
        <eaten>2014-09</eaten>
    </fruit>
```

Sources

"Creating XML Mappings in Excel 2003." *Office|Dev Center.* Microsoft, n.d. Web. 28 Apr. 2015
https://msdn.microsoft.com/en-us/library/office/aa203737%28v=office.11%29.aspx

"XML Schema Definition (XSD) data type support." *Office.* Microsoft, n.d. Web. 27 Apr. 2015
https://support.office.com/en-in/article/XML-Schema-Definition-XSD-data-type-support-7cd3c906-9b9e-4a64-ba77-1b23dc5c771c

"XSD Text-Only Elements." *W3schools.com.* Refsnes Data, n.d. Web. 27 Apr. 2015
http://www.w3schools.com/schema/schema_complex_text.asp

Appendix

Full Final XSD

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="fruits">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="fruit" minOccurs="0" maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="fruit">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="name" >
                    <xs:complexType>
                        <xs:simpleContent>
                            <xs:extension base="xs:string">
                                <xs:attribute name="scientificName" type="xs:string"/>
                            </xs:extension>
                        </xs:simpleContent>
                    </xs:complexType>
                </xs:element>
                <xs:element name="color" type="xs:string" />
                <xs:element name="growsOn" type="xs:string" />
                <xs:element name="eaten" type="xs:gYearMonth" />
            </xs:sequence>
            <xs:attribute name="id" type="xs:int"/>
        </xs:complexType>
    </xs:element>
</xs:schema>
```