aXel – Aaron's XSLT Editor & Library
User Guide

This document is the comprehensive guide for aXel, an application designed to easily work with XSLT and XPath.

The aXel Editor Window

The basic aXel editor window is a three-pane arrangement with a window for XML input (left), XSLT input (right), and output (bottom):

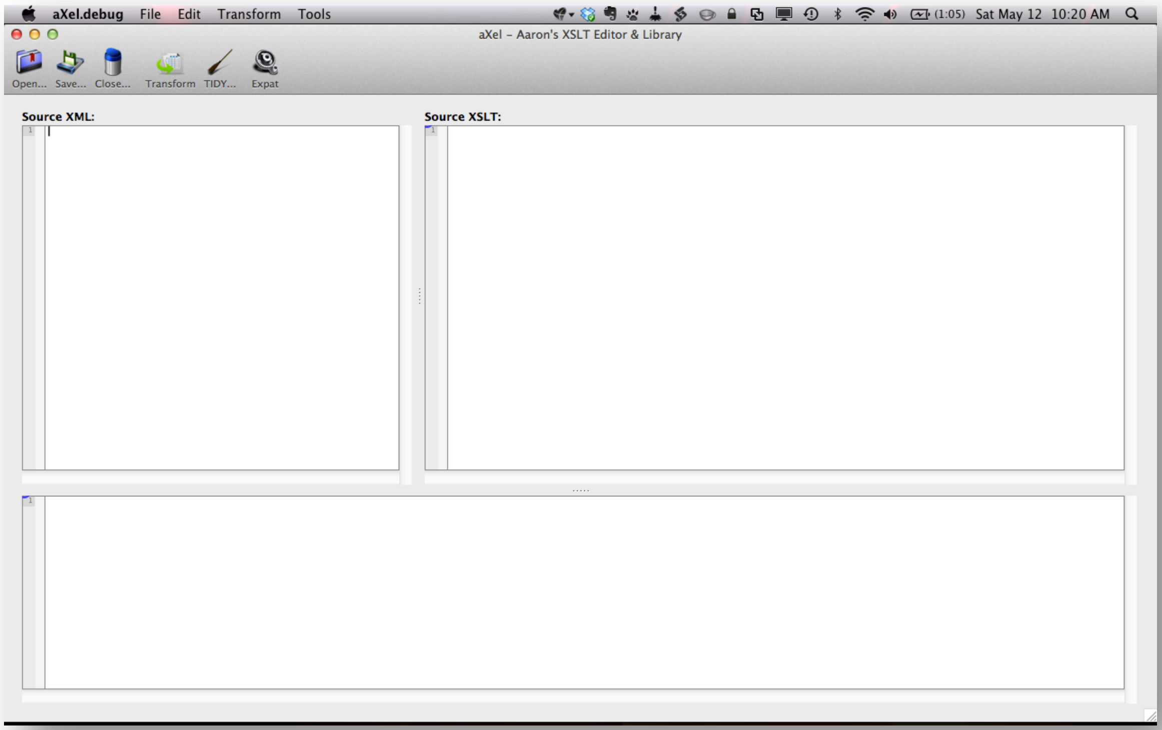


Figure 1. The main aXel editor window

There are multiple methods for adding XML/XSLT into their respective editor windows. Explanations of each method are detailed below.

Manually Entering XML/XSLT

The easiest manner to input XML/XSLT into aXel is to manually enter it into the editor. Simply click on either editor and start typing!

Opening XML/XSLT Files on the Filesystem

Selecting “File >> Open XML…” (⌘O on OS X; Ctrl+O on Windows) allows you to select an XML file on your filesystem to load. Similarly, “File >> Open XSLT…” (⇧⌘O on OS X; Shift+Ctrl+O on Windows) allows you to load XSLT via a file.

Note that in order to be recognized by aXel, files must have either a .xml file extension or a .xsl file extension.

Downloading XML/XSLT from an HTTP Source

aXel contains the ability to download XML/XSLT directly from a HTTP-based URL. At any point, you can select “File >> Download XML…” (⌘D on OS X; Ctrl+D on Windows) or “File >> Download XSLT…” (⇧⌘D on OS X; Shift+Ctrl+D on Windows) to initiate the process.

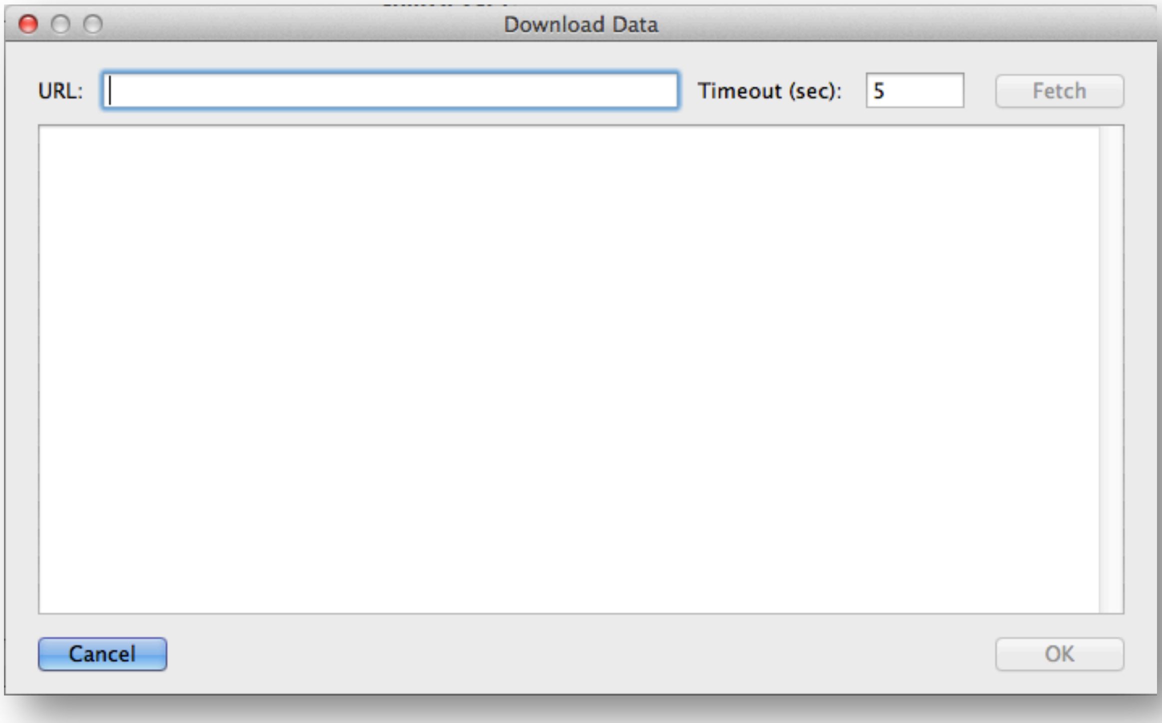
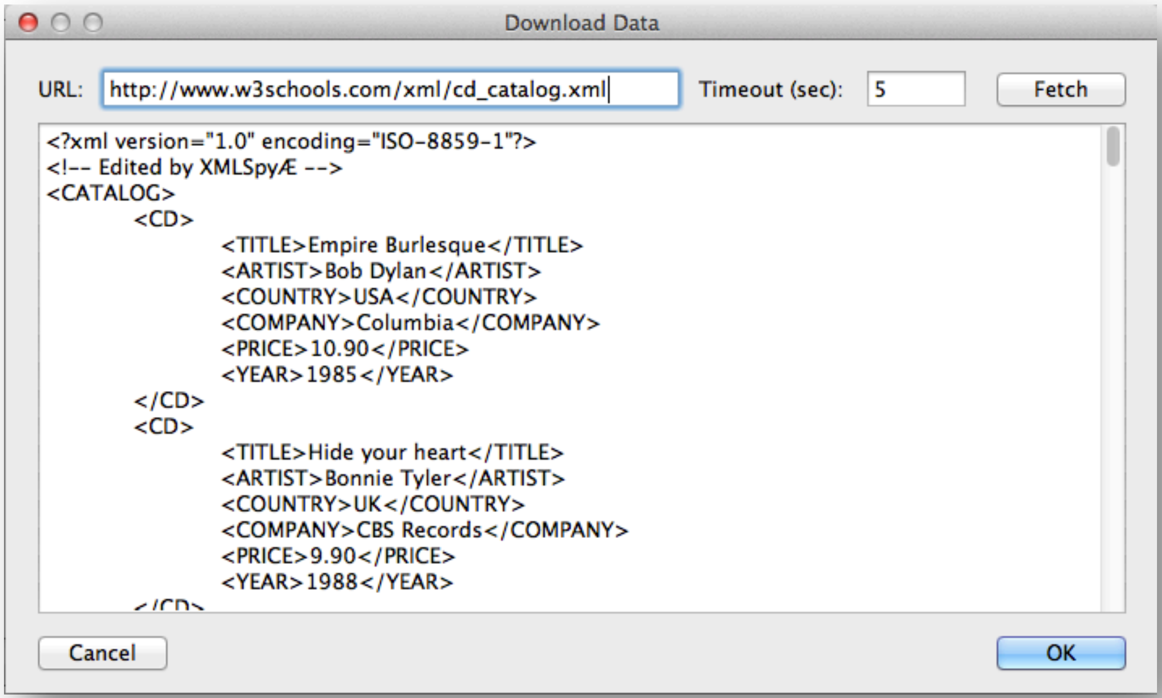


Figure 2. The "Download Data" window

Simply input a valid URL in the “URL” text field, adjust the timeout (the amount of time aXel should try to fetch data from the URL before it quits) if desired, and click “Fetch”. Assuming data retrieval is successful, it will be placed into the main text area:

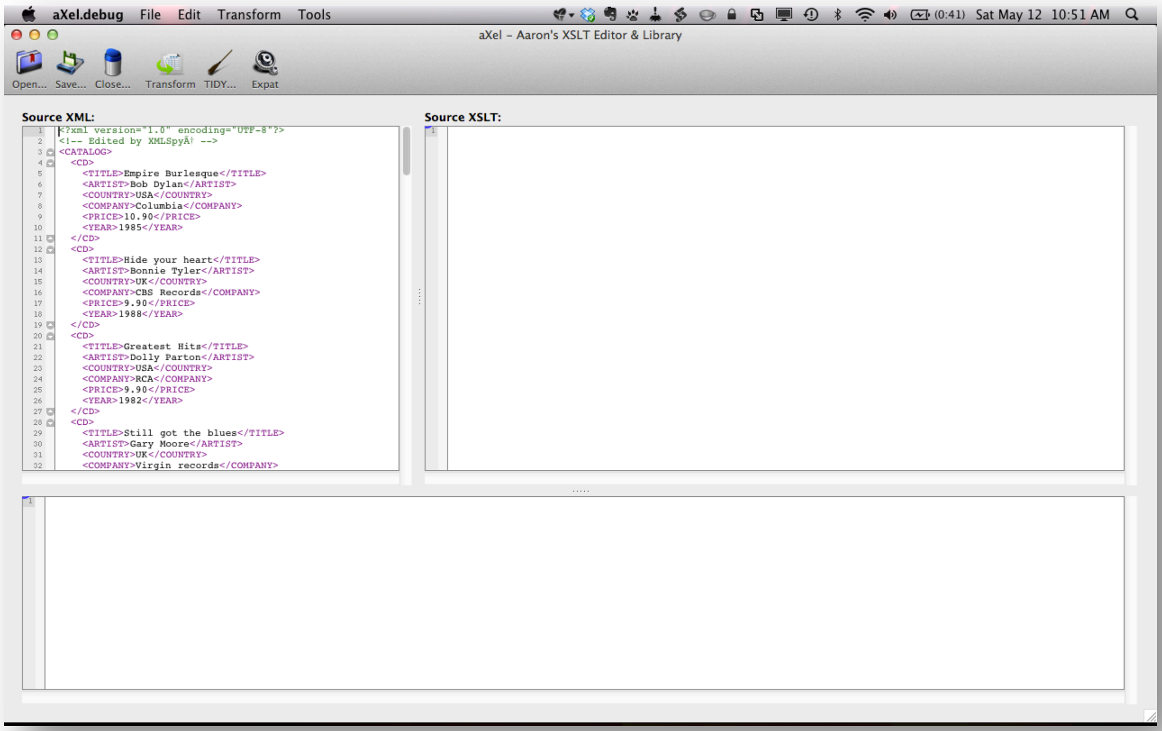


```
http://www.w3schools.com/xml/cd_catalog.xml
```

Timeout (sec): 5

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- Edited by XMLSpyÆ -->
<CATALOG>
	<CD>
		<TITLE>Empire Burlesque</TITLE>
		<ARTIST>Bob Dylan</ARTIST>
		<COUNTRY>USA</COUNTRY>
		<COMPANY>Columbia</COMPANY>
		<PRICE>10.90</PRICE>
		<YEAR>1985</YEAR>
	</CD>
	<CD>
		<TITLE>Hide your heart</TITLE>
		<ARTIST>Bonnie Tyler</ARTIST>
		<COUNTRY>UK</COUNTRY>
		<COMPANY>CBS Records</COMPANY>
		<PRICE>9.90</PRICE>
		<YEAR>1988</YEAR>
	</CD>
```

Figure 3. Fetching data from the entered URL



```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Edited by XMLSpyÂ† -->
<CATALOG>
  <CD>
    <TITLE>Empire Burlesque</TITLE>
    <ARTIST>Bob Dylan</ARTIST>
    <COUNTRY>USA</COUNTRY>
    <COMPANY>Columbia</COMPANY>
    <PRICE>10.90</PRICE>
    <YEAR>1985</YEAR>
  </CD>
  <CD>
    <TITLE>Hide your heart</TITLE>
    <ARTIST>Bonnie Tyler</ARTIST>
    <COUNTRY>UK</COUNTRY>
    <COMPANY>CBS Records</COMPANY>
    <PRICE>9.90</PRICE>
    <YEAR>1988</YEAR>
  </CD>
  <CD>
    <TITLE>Greatest Hits</TITLE>
    <ARTIST>Dolly Parton</ARTIST>
    <COUNTRY>USA</COUNTRY>
    <COMPANY>RCA</COMPANY>
    <PRICE>9.90</PRICE>
    <YEAR>1982</YEAR>
  </CD>
  <CD>
    <TITLE>Still got the blues</TITLE>
    <ARTIST>Gary Moore</ARTIST>
    <COUNTRY>UK</COUNTRY>
    <COMPANY>Virgin records</COMPANY>
```

Figure 4. Accepting the data download, which places it into the editor window

Saving XML/XSLT to the Filesystem

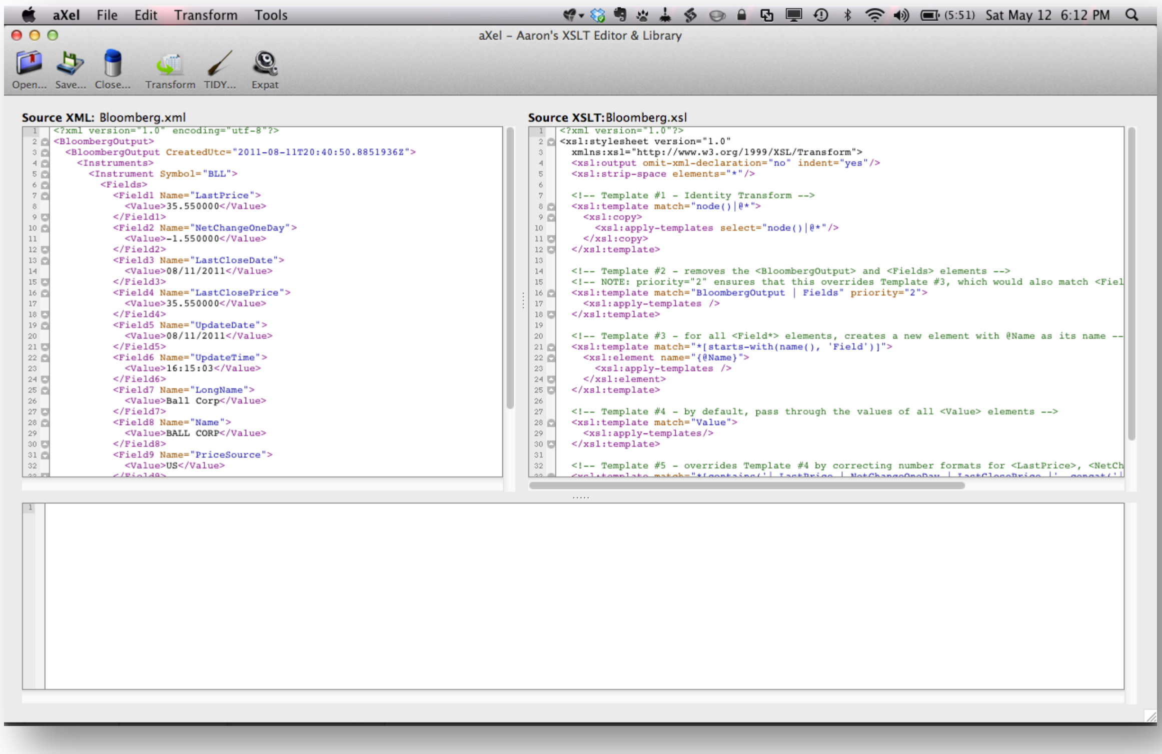
Saving XML/XSLT is a simple and straightforward process. At any point, you can select “File >> Save XML…” (⌘D on OS X; Ctrl+D on Windows) or “File >> Save XSLT…” (⇧⌘D on OS X; Shift+Ctrl+D on Windows) to initiate the process.

Working with XML with XSLT

At its core, aXel’s primary purpose is to transform XML documents via XSLT. This section of the document explains the basic process for accomplishing this task within aXel.

Transforming XML via XSLT

Transforming an XML document is a very simple process. First, ensure that both XML and XSLT editors have valid data:



Source XML: Bloomberg.xml

```
<?xml version="1.0" encoding="utf-8"?>
<BloombergOutput>
  <BloombergOutput CreatedUtc="2011-08-11T20:40:50.8851936Z">
    <Instruments>
      <Instrument Symbol="BLL">
        <Fields>
          <Field1 Name="LastPrice">
            <Value>35.550000</Value>
          </Field1>
          <Field2 Name="NetChangeOneDay">
            <Value>-1.550000</Value>
          </Field2>
          <Field3 Name="LastCloseDate">
            <Value>08/11/2011</Value>
          </Field3>
          <Field4 Name="LastClosePrice">
            <Value>35.550000</Value>
          </Field4>
          <Field5 Name="UpdateDate">
            <Value>08/11/2011</Value>
          </Field5>
          <Field6 Name="UpdateTime">
            <Value>16:15:03</Value>
          </Field6>
          <Field7 Name="LongName">
            <Value>Ball Corp</Value>
          </Field7>
          <Field8 Name="Name">
            <Value>BALL CORP</Value>
          </Field8>
          <Field9 Name="PriceSource">
            <Value>US</Value>
```

Source XSLT:Bloomberg.xsl

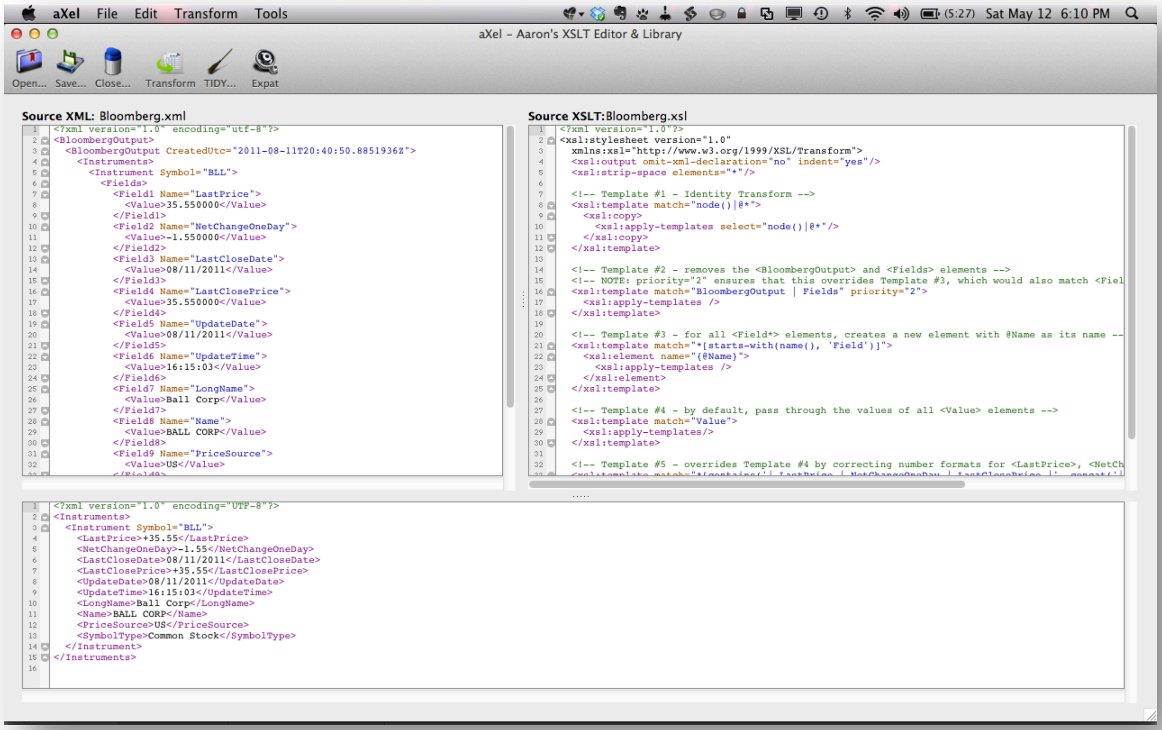
```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output omit-xml-declaration="no" indent="yes"/>
  <xsl:strip-space elements="*"/>

  <!-- Template #1 - Identity Transform -->
  <xsl:template match="node()|@*">
    <xsl:copy>
      <xsl:apply-templates select="node()|@*"/>
    </xsl:copy>
  </xsl:template>

  <!-- Template #2 - removes the <BloombergOutput> and <Fields> elements -->
  <!-- NOTE: priority="2" ensures that this overrides Template #3, which would also match <Fiel
  <xsl:template match="BloombergOutput | Fields" priority="2">
    <xsl:apply-templates />
  </xsl:template>

  <!-- Template #3 - for all <Field*> elements, creates a new element with @Name as its name --
  <xsl:template match="*[starts-with(name(), 'Field')]">
    <xsl:element name="{@Name}">
      <xsl:apply-templates />
    </xsl:element>
  </xsl:template>

  <!-- Template #4 - by default, pass through the values of all <Value> elements -->
  <xsl:template match="Value">
    <xsl:apply-templates/>
  </xsl:template>

  <!-- Template #5 - overrides Template #4 by correcting number formats for <LastPrice>, <NetCh
```

Figure 5. Editor windows with XML and XSLT entered

Then, run the transformation via any of the three methods:

- Click on Transform >> Transform
- Click on the “Transform” icon in the menu bar
- Use the hotkey combination: ⌘D on OS X; Ctrl+D on Windows

The results from the transformation will be placed into the Output editor:



Source XML: Bloomberg.xml

```
<?xml version="1.0" encoding="utf-8"?>
<BloombergOutput>
  <BloombergOutput CreatedUtc="2011-08-11T20:40:50.8851936Z">
    <Instruments>
      <Instrument Symbol="BLL">
        <Fields>
          <Field1 Name="LastPrice">
            <Value>35.550000</Value>
          </Field1>
          <Field2 Name="NetChangeOneDay">
            <Value>-1.550000</Value>
          </Field2>
          <Field3 Name="LastCloseDate">
            <Value>08/11/2011</Value>
          </Field3>
          <Field4 Name="LastClosePrice">
            <Value>35.550000</Value>
          </Field4>
          <Field5 Name="UpdateDate">
            <Value>08/11/2011</Value>
          </Field5>
          <Field6 Name="UpdateTime">
            <Value>16:15:03</Value>
          </Field6>
          <Field7 Name="LongName">
            <Value>Ball Corp</Value>
          </Field7>
          <Field8 Name="Name">
            <Value>BALL CORP</Value>
          </Field8>
          <Field9 Name="PriceSource">
            <Value>US</Value>
```

Source XSLT:Bloomberg.xsl

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output omit-xml-declaration="no" indent="yes"/>
  <xsl:strip-space elements="*"/>

  <!-- Template #1 - Identity Transform -->
  <xsl:template match="node()|@*">
    <xsl:copy>
      <xsl:apply-templates select="node()|@*"/>
    </xsl:copy>
  </xsl:template>

  <!-- Template #2 - removes the <BloombergOutput> and <Fields> elements -->
  <!-- NOTE: priority="2" ensures that this overrides Template #3, which would also match <Fiel
  <xsl:template match="BloombergOutput | Fields" priority="2">
    <xsl:apply-templates />
  </xsl:template>

  <!-- Template #3 - for all <Field*> elements, creates a new element with @Name as its name --
  <xsl:template match="*[starts-with(name(), 'Field')]">
    <xsl:element name="{@Name}">
      <xsl:apply-templates />
    </xsl:element>
  </xsl:template>

  <!-- Template #4 - by default, pass through the values of all <Value> elements -->
  <xsl:template match="Value">
    <xsl:apply-templates/>
  </xsl:template>

  <!-- Template #5 - overrides Template #4 by correcting number formats for <LastPrice>, <NetCh
```

```
<?xml version="1.0" encoding="UTF-8"?>
<Instruments>
  <Instrument Symbol="BLL">
    <LastPrice>+35.55</LastPrice>
    <NetChangeOneDay>-1.55</NetChangeOneDay>
    <LastCloseDate>08/11/2011</LastCloseDate>
    <LastClosePrice>+35.55</LastClosePrice>
    <UpdateDate>08/11/2011</UpdateDate>
    <UpdateTime>16:15:03</UpdateTime>
    <LongName>Ball Corp</LongName>
    <Name>BALL CORP</Name>
    <PriceSource>US</PriceSource>
    <SymbolType>Common Stock</SymbolType>
  </Instrument>
</Instruments>
```

Figure 6. The transformation results

TIDYing the Editors

When any editor has valid XML-based data, aXel can re-indent/”prettify” that data. There are two main ways to accomplish this:

- The File menu contains four options: TIDY XML, TIDY XSLT, TIDY Output, and TIDY All.
- The Menu bar contains the same options.

For reference, aXel accomplishes this “prettification” by running the data through an XSLT that copies everything as-is, but ensures that proper indention occurs.

Changing XSLT Parsers

aXel provides the ability to change the XSLT parsing engine used for transformations, XPath analysis, etc. Changing the parser can be accomplished via two methods:

- The Transform >> XSLT Parser menu
- The gear icon on the menu bar

A description of each parser is given below. Note that the parsers available depend on which operating system aXel is running on; this information is included below.

Expat

Available on Windows and OS X.

Expat (http://expat.sourceforge.net/) is a C-based XSLT parser that is built for light, quick transformations. It is the default parser within aXel.

Pros:

- Fast

Cons:

- Does not implement common extension functions, like those found in EXSLT.
- Older and not in active development
- Only implements XSLT/XPath 1.0

libxslt

Available on Windows and OS X.

libxslt (http://xmlsoft.org/xslt/) is a C-based XSLT parser that was originally built for the GNOME project. It is one of the most widely used XSLT 1.0 parsers in existence today. Additionally, out of all the parsers that aXel implements, its functionality is closest to that found within FWi Content Manager.

Pros:

- Fast
- Implements EXSLT extension functions
- Incredible stable
- Long track record as a reliable parser

Cons:

- Only implements XSLT/XPath 1.0

Saxon

Available on Windows and OS X.

Saxon (http://saxon.sourceforge.net/) is a Java-based XSLT parser that was originally built for the GNOME project. In addition to being developed by Michael Kay, the proverbial godfather of XML and XSLT, it is one of only a few processers to implement XSLT/XPath 2.0.

Pros:

- Implements XSLT/XPath 2.0
- Incredible stable
- Long track record as a reliable parser

Cons:

- Runs through a Java interpreter; slower than other parsers

AltovaXML

Available on Windows.

AltovaXML (http://www.altova.com/altovaxml.html) is a C-based XSLT parser that was originally built for the GNOME project. It implements both XSLT 1.0 and XSLT 2.0 (with aXel intelligently choosing between the two based on the source data).

Pros:

- Implements XSLT/XPath 2.0
- Fairly fast

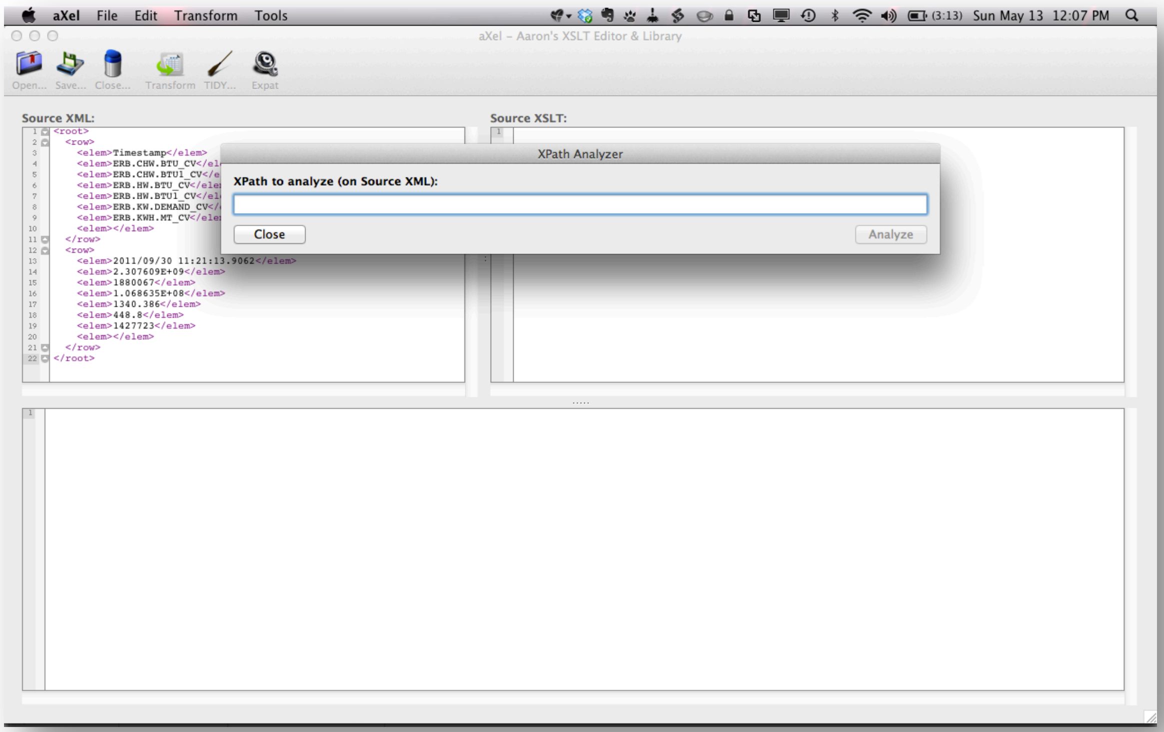
Cons:

- Only available on Windows

XPath Analysis

aXel has the ability to test XPaths against source XML; this allows users to quickly see what, if any, nodes/attributes/etc. in an XML document match a provided XPath.

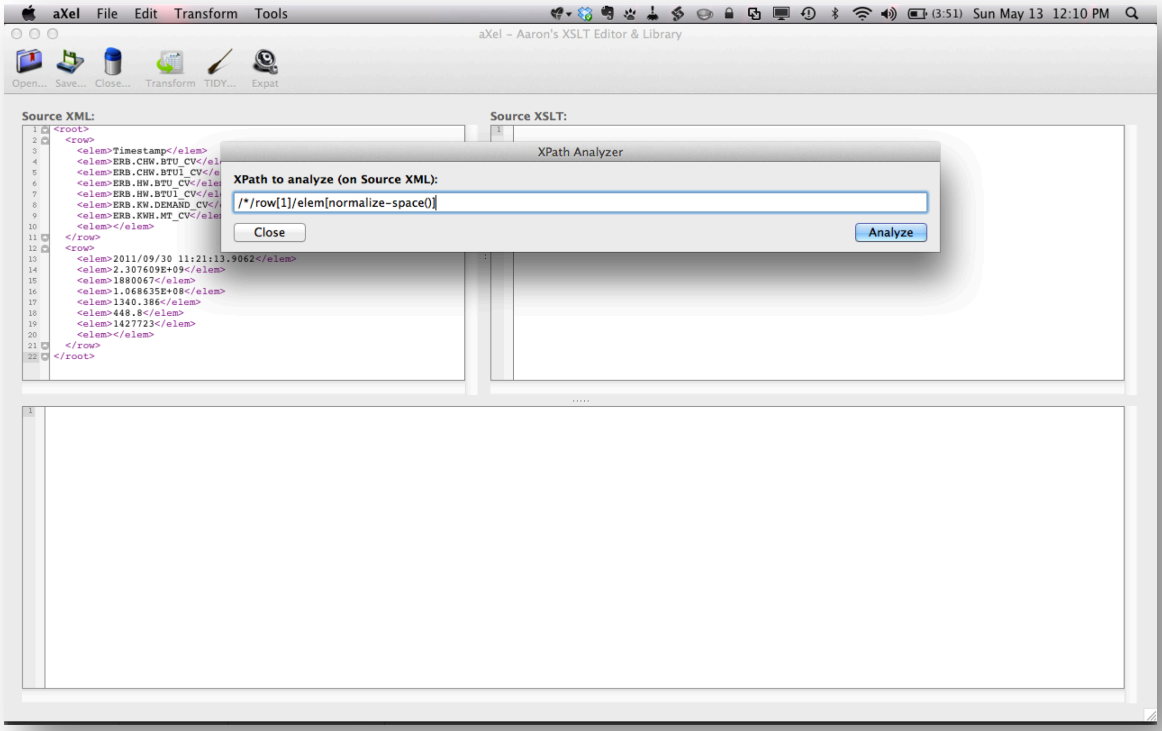
Once there is valid XML in the XML editor, launch the XPath Analyzer, either via the Tools >> XPath Analyzer menu or via the system hotkey (⇧⌘X on OS X; Shift+Ctrl+X on Windows).



```
<root>
  <row>
    <elem>Timestamp</elem>
    <elem>ERB.CHW.BTU_CV</el
    <elem>ERB.CHW.BTU1_CV</e
    <elem>ERB.HW.BTU_CV</elem
    <elem>ERB.HW.BTU1_CV</el
    <elem>ERB.KW.DEMAND_CV</
    <elem>ERB.KWH.MT_CV</elem
    <elem></elem>
  </row>
  <row>
    <elem>2011/09/30 11:21:13.9062</elem>
    <elem>2.307609E+09</elem>
    <elem>1880067</elem>
    <elem>1.068635E+08</elem>
    <elem>1340.386</elem>
    <elem>448.8</elem>
    <elem>1427723</elem>
    <elem></elem>
  </row>
</root>
```

Figure 7. Opening the XPath Analyzer

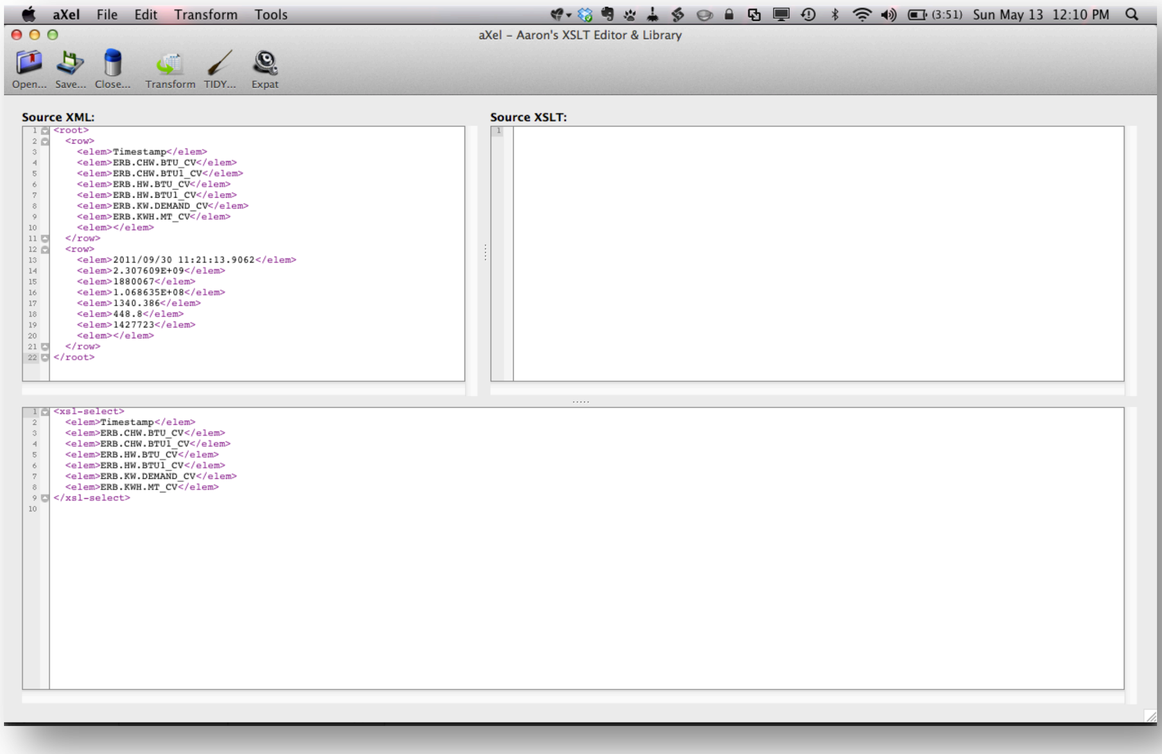
Simply type in a valid XPath and press “Analyze” to see the results. Currently, those results are output to the Output editor window; in future releases, aXel will highlight the results within the source XML document.



```
<root>
  <row>
    <elem>Timestamp</elem>
    <elem>ERB.CHW.BTU_CV</el
    <elem>ERB.CHW.BTU1_CV</e
    <elem>ERB.HW.BTU_CV</ele
    <elem>ERB.HW.BTU1_CV</el
    <elem>ERB.KW.DEMAND_CV</
    <elem>ERB.KWH.MT_CV</ele
    <elem></elem>
  </row>
  <row>
    <elem>2011/09/30 11:21:13.9062</elem>
    <elem>2.307609E+09</elem>
    <elem>1880067</elem>
    <elem>1.068635E+08</elem>
    <elem>1340.386</elem>
    <elem>448.8</elem>
    <elem>1427723</elem>
    <elem></elem>
  </row>
</root>
```

XPath to analyze (on Source XML): `/*/row[1]/elem[normalize-space()]`

Figure 8. Entering an XPath to analyze


aXel File Edit Transform Tools
(3:51) Sun May 13 12:10 PM
aXel – Aaron's XSLT Editor & Library
Open... Save... Close... Transform TIDY... Expat
Source XML:
Source XSLT:

```
<root>
  <row>
    <elem>Timestamp</elem>
    <elem>ERB.CHW.BTU_CV</elem>
    <elem>ERB.CHW.BTU1_CV</elem>
    <elem>ERB.HW.BTU_CV</elem>
    <elem>ERB.HW.BTU1_CV</elem>
    <elem>ERB.KW.DEMAND_CV</elem>
    <elem>ERB.KWH.MT_CV</elem>
    <elem></elem>
  </row>
  <row>
    <elem>2011/09/30 11:21:13.9062</elem>
    <elem>2.307609E+09</elem>
    <elem>1880067</elem>
    <elem>1.068635E+08</elem>
    <elem>1340.386</elem>
    <elem>448.8</elem>
    <elem>1427723</elem>
    <elem></elem>
  </row>
</root>
```

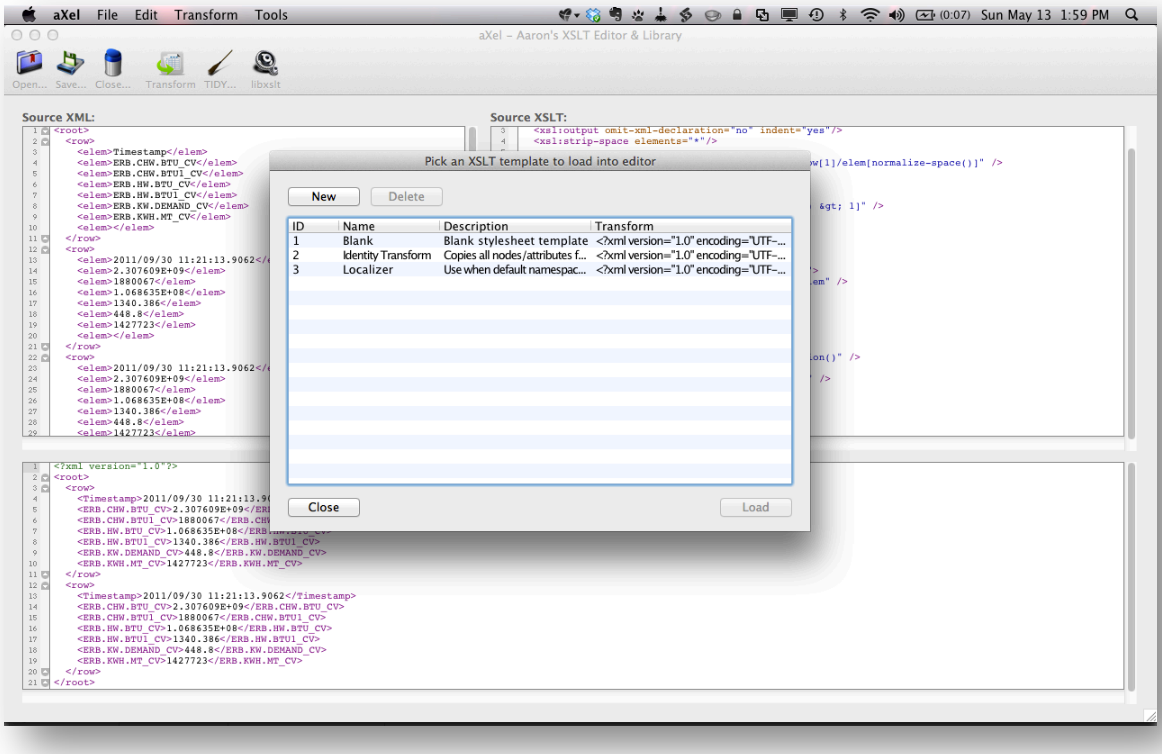
```
<xsl-select>
  <elem>Timestamp</elem>
  <elem>ERB.CHW.BTU_CV</elem>
  <elem>ERB.CHW.BTU1_CV</elem>
  <elem>ERB.HW.BTU_CV</elem>
  <elem>ERB.HW.BTU1_CV</elem>
  <elem>ERB.KW.DEMAND_CV</elem>
  <elem>ERB.KWH.MT_CV</elem>
</xsl-select>
```

Figure 9. XPath results displayed in the Output editor window

Using the XSLT Snippet Library

In addition to being an editor of XML and XSLT, aXel can store commonly used XSLTs in its snippet library.

At any point, the library can be opened via the Tools >> XSLT Snippets menu item or the hotkey combination (⇧⌘L on OS X; Shift+Ctrl+L on Windows).



```
<root>
  <row>
    <elem>Timestamp</elem>
    <elem>ERB.CHW.BTU_CV</elem>
    <elem>ERB.CHW.BTU1_CV</elem>
    <elem>ERB.HW.BTU_CV</elem>
    <elem>ERB.HW.BTU1_CV</elem>
    <elem>ERB.KW.DEMAND_CV</elem>
    <elem>ERB.KWH.MT_CV</elem>
    <elem></elem>
  </row>
  <row>
    <elem>2011/09/30 11:21:13.9062</e
    <elem>2.307609E+09</elem>
    <elem>1880067</elem>
    <elem>1.068635E+08</elem>
    <elem>1340.386</elem>
    <elem>448.8</elem>
    <elem>1427723</elem>
    <elem></elem>
  </row>
  <row>
    <elem>2011/09/30 11:21:13.9062</e
    <elem>2.307609E+09</elem>
    <elem>1880067</elem>
    <elem>1.068635E+08</elem>
    <elem>1340.386</elem>
    <elem>448.8</elem>
    <elem>1427723</elem>
```

```
    <xsl:output omit-xml-declaration="no" indent="yes"/>
    <xsl:strip-space elements="*"/>
w[1]/elem[normalize-space()]" />
 &gt; 1]" />
>
em" />
on()" />
' />
```

ID	Name	Description	Transform
1	Blank	Blank stylesheet template	<?xml version="1.0" encoding="UTF-...
2	Identity Transform	Copies all nodes/attributes f...	<?xml version="1.0" encoding="UTF-...
3	Localizer	Use when default namespac...	<?xml version="1.0" encoding="UTF-...

```
<?xml version="1.0"?>
<root>
  <row>
    <Timestamp>2011/09/30 11:21:13.9
    <ERB.CHW.BTU_CV>2.307609E+09</ERB
    <ERB.CHW.BTU1_CV>1880067</ERB.CHW
    <ERB.HW.BTU_CV>1.068635E+08</ERB.
    <ERB.HW.BTU1_CV>1340.386</ERB.HW.BTU1_CV>
    <ERB.KW.DEMAND_CV>448.8</ERB.KW.DEMAND_CV>
    <ERB.KWH.MT_CV>1427723</ERB.KWH.MT_CV>
  </row>
  <row>
    <Timestamp>2011/09/30 11:21:13.9062</Timestamp>
    <ERB.CHW.BTU_CV>2.307609E+09</ERB.CHW.BTU_CV>
    <ERB.CHW.BTU1_CV>1880067</ERB.CHW.BTU1_CV>
    <ERB.HW.BTU_CV>1.068635E+08</ERB.HW.BTU_CV>
    <ERB.HW.BTU1_CV>1340.386</ERB.HW.BTU1_CV>
    <ERB.KW.DEMAND_CV>448.8</ERB.KW.DEMAND_CV>
    <ERB.KWH.MT_CV>1427723</ERB.KWH.MT_CV>
  </row>
</root>
```

Figure 10. Opening the XSLT Snippet window

Note: by default, aXel ships with three commonly used XSLTs:

- *Blank*: a blank XSLT with the declaration filled out
- *Identify Transform*: a commonly used XSLT that copies the source XML document as-is
- *Localizer*: an XSLT that removes all namespaces from every element in the source XML

Loading an XSLT Snippet

With the XSLT snippet window open, XSLTs can be loaded in two different ways:

1. Double-click on the desired snippet
2. Click once on the snippet to highlight it, then press “Load”

Creating New Snippets

Clicking “New” launches a window to define new XSLT snippets. Simply give your new snippet a name, a description, and the XSLT needed, then press “Save”.

Note that you can quickly turn XSLT in the editor into a snippet by going to Tools >> Quick Save Snippet.