Using the Model Interface to create XML files from CSV files

OBJECTS uses XML files as inputs. Often, however, it is most convenient to create input files an Excel spreadsheet or from other analysis tools. The model interface contains tools to convert CSV (Comma Separated Value) files to XML files. To do this the spreadsheet must be saved as a CSV file, and an appropriate header for the CSV file must be written. The Model Interface is able to open the CSV and header files in order to create XML format files.

1) CSV Files

CSVs are data files made in MS Excel. A single worksheet consisting of any number of data tables will be converted into a single xml file. Each table in the csv should be preceded by the following label and a number:

INPUT_TABLE
Variable ID
100

The number (100 in the example above) corresponds to the appropriate number in the header file, which is a .txt document written to instruct the model interface how to read the tables of the csv file. Headers are covered later in this document. The first two lines below the table number are not read in; these are used only for notation purposes, to identify the columns of the table and make other notes. Data will be read in starting three lines below the number. Tables with the same headers can share a number; in other words, there can be many tables numbered "100" as long as they have the same sequence of tags, and the information in the tables is sufficient to place the data in the correct place in the xml file to be created.

To save an Excel file as a CSV, click on File -> Save As, and then under "Save as type," select comma-separated-value. When this is done, only the present worksheet is saved as a CSV. CSVs have no linked data; all references in cells are lost, replaced by only the numbers. However, after saving an Excel file as a CSV, it will continue to appear as it was: all links and formulas will remain valid, and you'll be able to go to different worksheets in the workbook. This is important because if you make changes to the file and save as you go along, but exit without then re-saving the file as a .XLS workbook, you'll lose all changes you made.

2) Headers

Header files are .txt files made to provide instructions to the model Interface on how CSV input files are to be converted to XML files. The first line of a header file is not read in; it's used for titles or other notes. A header file consists of any number of individual headers, separated by line breaks, and each header in turn consists of parent/child pairs separated by commas. Each header should start out with a number (corresponding to a table), followed by parent/child pairs starting with leftmost column, and covering all

columns of the relevant table. Data will be read from the table for any parent/child pair with a “+” sign, and assigned to the element immediately after the +. Not all parent/child pairs correspond to data being read from the table, but those which do not correspond to data should be in the header after all pairs with data. Also, it is common to name a tag at the level of the tag, in which case the following notation is used: parent/+{name}tag. Below is an example, followed by an explanation of what is happening.

CSV example:

INPUT_TABLE				
Variable ID				
	1003			
		object-meta-info		
renewresource		name	value	
Class 3 wind		resourceCapacityFactor		0.35
Class 4 wind		resourceCapacityFactor		0.35
Class 5 wind		resourceCapacityFactor		0.35
Class 6 wind		resourceCapacityFactor		0.35
Class 7 wind		resourceCapacityFactor		0.35

Corresponding header:

1003,region/+{name}renewresource,renewresource/+{name}object-meta-info,object-meta-info/+value,world/{name=USA}region,scenario/world,scenario

1003 tells the model interface to use this line to interpret any tables labeled 1003 in the csv file. The 1003 is recognized as a table because it is immediately below the first two identifier lines.

region/+{name}renewresource tells it to take the data in the leftmost column, and place that as the name of the renewresource, which is a child of region. The same process is happening in the next parent/child pair: data from the table (resourceCapacityFactor) is the name of the object-meta-info, which is placed within the renewresource.

object-meta-info/+value assigns the number in the third column to the tag “value” within object-meta-info.

The remaining parent/child pairs are necessary for fitting this data into the global model. The order of the remaining pairs does not matter, but all headers should have a “scenario” (note that this has no parent; it is the root of the XML tree), and a “scenario/world”.

Below is an example an entire xml file generate from a single header/csv combination:

```
<scenario>
  <world>
    <region name="USA">
      <renewresource name="Class 3 wind">
```

```
        <object-meta-info name="resourceCapacityFactor">
          <value>0.35</value>
        </object-meta-info>
      </renewresource>
      <renewresource name="Class 4 wind">
        <object-meta-info name="resourceCapacityFactor">
          <value>0.35</value>
        </object-meta-info>
      </renewresource>
      <renewresource name="Class 5 wind">
        <object-meta-info name="resourceCapacityFactor">
          <value>0.35</value>
        </object-meta-info>
      </renewresource>
      <renewresource name="Class 6 wind">
        <object-meta-info name="resourceCapacityFactor">
          <value>0.35</value>
        </object-meta-info>
      </renewresource>
      <renewresource name="Class 7 wind">
        <object-meta-info name="resourceCapacityFactor">
          <value>0.35</value>
        </object-meta-info>
      </renewresource>
    </region>
  </world>
</scenario>
```

Headers can be used to directly assign attributes (these are generally used for names and years) to tags, or the attributes can be pulled from the table. The former is useful for reducing the number of columns and redundancy of CSV tables. The latter is useful for reducing the number of tables in the CSV file and the number of separate headers necessary for the conversion to xml. For instance, this example was excerpted from an input for US wind power, and it would have been needlessly redundant to identify the region in every single CSV table when it would not vary between any of them. At the same time, the above example could have been written with five separate headers and tables, one for each of the different renewresource. In that case, each CSV would not have the renewresource column.

This program is quite flexible, and there are generally a number of options for headers and CSVs that will work to convert data into an XML file.

a. Header File Syntax

CSV to XML header files are written according to the following syntax rules:

1. **Parent/Child**
   Define hierarchy with no Attributes or data

2. **Parent/+Child**

Read data from table and associate it with the Child tag

3. **Parent/+{AttrName}Child**

Read attribute value from the table and set Child's AttrName to that value

4. **Parent/{AttrName=AttrValue}Child**

Specify the attribute's name and value; could be used to ignore a column and specify your own attribute value

5. **Parent/+{AttrName=AttrValue}Child**

Specify the attribute's name and value and read the tag's data from the table.

6. **{AttrName=AttrValue}Parent/Child**

Specify exactly which parent the child should nest under. Any of the above child notations can be used.

7. **{AttrName=AttrValue}GrandParent/.../Parent/Child**

Method I

Specify which Grandparent this node should nest under. Grandparents can be as far up the path as desired as long as there is a path defined to get to the Parent node. To define that path you need to start at @Grandparent. Any of the above notations can be used for the Parent/Child.

Note: Failing to place an '@' symbol in front of a header, such as @populationSGMFixed/ageCohort, will leave in empty tags like <ageCohort /> in the XML file.

Method II

Specify the parent/child pair using the immediate parent. Finish all parent/child pairs that correspond to the data in the csv. Then enter the parent/child pair (no data attached, though attributes can be assigned) with the child as the parent of the value of interest. Continue to do this until that value is placed in the hierarchy of existing nodes. For instance, to put another level between object-meta-info and value in the example above, the header would look like:

1003,region/+{name}renewresource,renewresource/+{name}object-meta-info,CalValue/+value, object-meta-info/CalValue,world/{name=USA}region,scenario/world,scenario

The xml (excepted) would look like:

```
<scenario>
  <world>
```

```
    <region name="USA">
      <renewresource name="Class 3 wind">
        <object-meta-info name="resourceCapacityFactor">
          <CalValue>
            <value>0.35</value>
          </CalValue>
        </object-meta-info>
      </renewresource>
    </region>
  </world>
</scenario>
```

8. Placing multiple attributes in a single tag

This is done with a semicolon (and no space) between the attributes. Only one can be pulled from the table using the plus sign. Therefore, at least one of the attributes should be set to an assigned value. The "fillout" term is often used to fill out all periods. For instance if the header reads:

CalValue/+{year=1975;fillout=1}value

The xml would read:

```
<value fillout="1" year="1975">0.35</value>
```

This would assign 0.35 to the value of all years starting from 1975.

Note: The converter will try to merge children on the same level as a single child. This should only happen if there are no conflicting attributes (e.g. parent/{year=1975}child and parent/{year=1990}child do not get merged). The following is an example:

An example of header files with mutiple attributes…

```
{attr1=value1;attr2=valu2}parent/+{name;otherAttr=value}child,{attr1=value1;attr2=val
ue2}parent/+child
```

…would yield something like:

```
<parent attr1="value1" attr2="value2">
    <child name="read value" otherAttr="value">
            Read Value
    </child>
</parent>
```

b. Common Errors

It's helpful to understand the most common errors.

1. **Exception thrown while trying to read CSV and header files**
   Java.land.NullPointerException

   Nothing was read in. Maybe the header did not match the CSV, or the first row was not left blank and it was the only header.

2. **Missing root header**
   The headers did not have the parent-less "scenario" tag

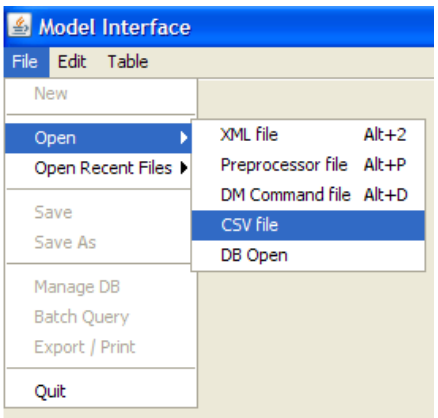
3. **Exception thrown…index 4, size 3**
   This is the most common error. It means there were too few columns in the CSV table compared to the number expected by the model interface as defined in the header file. In this case, the header had four places where it instructed the model interface to read from a table, and the actual table only had three columns. While it doesn't identify the table numerically, it does help in the debugging process to know that the relevant table only had three columns, and its header either had four plus signs, or had a parent/child pair with no data in the first three entries in the header. Even an empty parent/child pair is treated as corresponding to a column in the CSV table if its placement in the sequence of parent/child pairs is less than the number of columns in the table.

4. **If a plus sign is missing, this will not cause any errors, but data will not be added to the XML file.** So after running the conversion, always check to make sure all of the required data is present.

3) CSV to XML Conversion

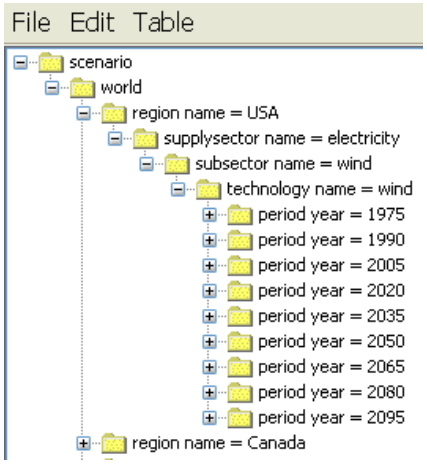
The purpose of the CSV to XML conversion is to get MS Excel data tables into xml format, so that they can be read into ObjECTS. To open CSV's with pre-existing headers:

a. Go to File and select Open CSV file.

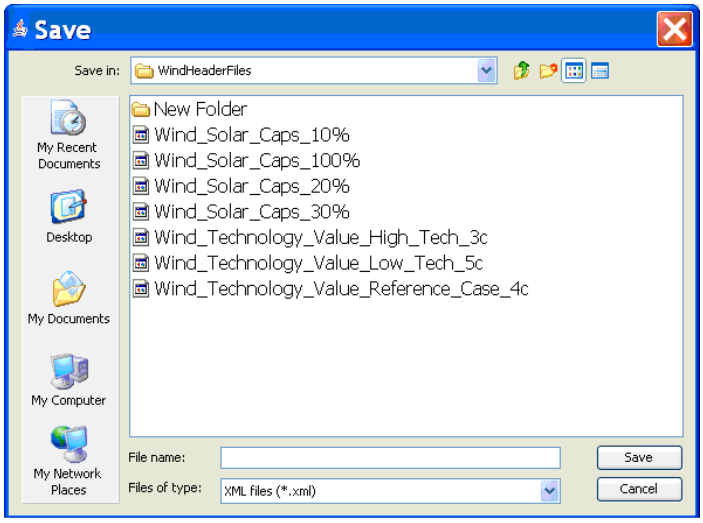


b. Select the CSV file which you wish open.

c. You will be prompted to open the appropriate header file. Select the header file which corresponds to the CSV file. A screen similar to the one below will appear. In the example below, the user has used the XML Database viewer to visualize data inputs for wind technology.



d. Go to File and select Save. A screen similar to the one below will appear.



e. Specify a file name and choose "XML files (*.xml)" in the Files of type scroll-down box. Click the Save button. An XML file has now been created from the CSV and header files. This XML file can be specified as an input file when running Objects.