R-SORTIE XML Maker

Sarah Beukema, ESSA Technologies Ltd.
March 2021

Contents

This R program is designed to allow an experienced R user to combine a complete Sortie input xml file with new parameters or xml code to create several new input files.

Summary

The tool will:

- create one or more new files that contain updated information based on user-provided parameters or xml code;
- run Sortie with these new files;
- extract the output files; and
- process the output treelist and/or grids.

Each component may also be run separately.

Update and Edit Input XML files

To edit the Sortie XML file, you must provide the tool with a set of information about the base file, the parameters or code that you are changing, and information about how to combine the input to make different files.

Input

There are a minimum of four files that must be present to run the model.

1. **Files.txt**: this contains the list of files that you wish to use.

   There are 6 different slots allowed in the file, with a minimum of two types required.

Type	Description	Required
0	Base xml file – the input xml file that contains the necessary parameters, behaviours, and structures. This file must contain all possible examples of variables and behaviours that will be modified in other files.	Yes
1	Parameter value file or xml section	Yes
2-5	Parameter value file or xml section	No

   There can be more than one of any of the file types. Details about the formats of the files are below.

   The code will loop through all combinations of ~~the~~ all the files.

   The resulting xml files will be named with a combination of each of the names present: type0-type1-type2.xml.

   Example:

type, name
0, Test.xml
1, t1.csv
2, f2.xml
2, g2.csv
3, p3.csv
3, r3.csv

The resulting output files will be named: test-t1-f2-p3.xml, test-t1-f2-r3.xml, test-t1-g2-p3.xml, test-t1-g2-r3.xml.

2. **VariableNames.csv**: This a translation file that matches the parameter names in the input file with those in the xml file, and tells general format of the variable. Formats are described more later. This file will only be changed to add new variables or to change the parameter name of a variable.

   File format:
   Parameter name, variable type, xml variable name, group name (if applicable)

   All variable names are completely case and space sensitive; the code is looking for exact matches. For example, use Timesteps not TimeSteps or timesteps or Time steps.

   The parameter names in the parameter value file can be specific to a user, but must be the same for all the parameter value files listed in the Files.txt.

3. **Base Xml file**: (type 0) This file should be a standard SORTIE input parameter file, with no modifications required. It is very important that this file contains all variables and sections that will be modified. For example, if you might be adding xml code with new harvest rules, a harvest section must be in the original SORTIE file. Similarly, if you are changing initial densities, then each size class that you might want to use should be in the original file.

4. **Parameter value files**:

   There are two types of parameter value files: csv and xml, both of which are further described below.

   In all cases, variables are updated in the xml file in the order that they are in the parameter value file, and in the order of the slot of parameter value files. Parameter value files are read from top to bottom, and file slots are addressed from 1 to 3, a variable that is in slot 3 will supersede one that is in a slot 1 file. This could be useful if you wish, for example, to replace an xml section (e.g. as a slot 2 file), but then change one of its parameters (e.g., in a slot 3 file).

4.1 CSV files

These files contain a set of new values for different parameters.

- The first line must be a blank column and then the species names **as in the xml file.** As with variable names it is critical that the species names are *exactly the same* as they are in the xml file and the code searches for an exact match.
- The second line must either be "na" or the name of BehaviourList to which the parameters following it apply. The BehavoiurList name comes from the original xml file, so requires knowledge of the xml file and its naming. However, it is necessary to give the list name because there could be multiple behavior lists with the same set of parameter names.
- When the parameters for the behavior list is finished, you need to mark the start of a new section with either a new BehaviourList or "na".
- If the parameter is by species, the code looks for a parameter in the corresponding column. If the parameter is not species specific (e.g. timestep), then only the first column is used. *Values can be in the other columns, they will just be ignored.*
- The code will create output file names for each new xml file. **The Parameter value file should include the directory into which the output should be placed.** The format must include a final "\" and will look something like:

  Output,C:\Projects\SORTIE\test\
  ShortOutput,C:\Projects\SORTIE\test\

  The file name will be a combination of the original xml filename and the parameter value file name, and will be the same filename used for the final xml file. Note that the file extension on the Output file will be _det.gz.tar and on the ShortOutput fill be .out.
  If these variables are not present in an xml file or csv file, and you are creating multiple files, you will run the risk of overwriting your output when you run SORTIE.

4.2 XML files

The second type of parameter value file allows for the replacement or deletion of xml code. The tool will replace the existing lines of text with the new lines of text from the file or delete a section of code. The file must contain complete xml format code.

- ***New behavior***: The file can contain a completely new set of behaviours or parameters, such as allometry or GapDispersal21. The tool will look at the first line of the file, determine what to look for, and replace all the lines between the beginning and end of the section with the new code. For example, you could use this to completely replace the allometry section of the input file.
- ***Delete an xml section***: Put the name of the section that you wish to delete in the first line of the xml file (e.g. StochasticMortality9). The second line of the file must contain -999. The tool will then find the section that corresponds to what is in the first line of the file, and then remove it entirely from the final xml file. For example:

  <StochasticMortality9>
  <delete>-999</delete>
  < /StochasticMortality9>

The tool will search through the xml file replacing one section at a time, in order. Any number of xml sections can be present in this file, and the file can include both sections to delete or sections to replace.

Output

The output from this program will be one or more xml files. A new xml file will be created for each combination of xml file and parameter value file that is listed the files.txt input file. The new xml files will be named as a combination of the names of the original xml and the parameter value file, and will be located in the run directory. (See the example above).

The filenames are also added to an internal R list that could be accessed by another program if necessary.

Running the code

This R project assumes that you are familiar with running a project in R.

The program assumes that all the code (R files), input files (i.e., VariablesNames.txt, Files.txt and all the required xml and parameter files) are in the same directory. All newly created xml files will also be placed in this directory.

You must have the R following libraries *xml2*, *stringr*, *tidyr, dplyr*

While all the R files that are included with this set of code are necessary, the file that you run is:

MakeFiles.r

There are several flags at the top of the code which will allow you to tell R to automatically run other routines:

- Run Sortie with the newly created files, with an option to give the location of the SORTIE executable.
- Extract the Sortie output files after the run.
- Parse the Sortie output files, and if so, which years to extract.

Each of these options can also be run independently or called from other R programs. Examine the code to see what must be defined for the run (usually just a filename).

The code should report errors for the two following cases:

1. The parameter name in the parameter value file is not in the VariableName.txt file. In this case the error message may be a bit ambiguous and may say that it could not find the type of the parameter. To fix, check the VariableName file to ensure that the parameter is present, is exactly the same as in the parameter value file, and has a valid type. If it is not present, use the rules described later to add it to the VariableName file.
2. The variable or group name is not in the xml file. This could be for three reasons:
    a. There is an error in name in the VariableName file. Remember that variable names must match exactly, including spacing and capitalization.
    b. The variable is not in the xml file. The xml file must have all variables that you plan to change. Not all these missing variables are an error that needs fixing, however. The most common one occurs if, for example, the parameter file contains potential values for all possible size classes, but the xml file only contains a subset of the size classes, such as for initial conditions.

c. The group is ~~not~~ in the xml file, but the variable name in question contains a number (i.e. Harvest9). The variable "Harvest" is likely in the original xml file, but it might have a different number. Because SORTIE allows the user to have multiple instances of some behaviours, each is given a unique number. The numbers in the original xml file and in the new parameter or xml file must match.

Editing the VariableNames File

The VariableNames.csv file is critical to the success of the code when using new values defined in the csv-format parameter value files. The VariableNames file contains two important pieces of information: the mapping of the parameter name in the input parameter value file to the corresponding xml variable name, and the type of the variable in the xml file. **You must be familiar with the SORTIE xml file format** because this file contains the names that the code is using to identify what to change.

The xml file contains many different formats for its variables. Some are simple formats with the variable and its value on the same line. Some include species, and some are further grouped. The Type column in the VariableNames file is critical to identify the format of the variable and thus the rules in the code for finding and replacing the right variable. So far, 7 different formats have been identified and coded.

The current version of the VariableNames file is not exhaustive. It only contains those variables that were being used in the testing files. However, new variables that fit one of the 7 currently defined types should be able to be added and the code should run.

***Variable Types**: Blue text is an example from the xml file (the SORTIE-generated file). Green text is an example from the VariableNames file and purple text is the example from the parameter value file (the csv file).*

Type 1: the parameter is on the same line and directly after the variable name:

```
<timesteps>20</timesteps>
Timesteps,1,timesteps,
Timesteps,66,66,66,66
```

Type 2: the parameter includes a species name, but the value is still on the same line:

```
<tr_mshVal species="Interior_Spruce">1.35</tr_mshVal>

Max.Seedling.Hgt.m, 2, tr_mshVal species,
Max.Seedling.Hgt.m,1.35,1.35,1.35,1.35
```

Type 3: the parameter is like a type 1, but is inside a BehaviourList. The BehaviourList becomes part of the input parameter file, since it is specific to the xml file that is being used.

```
<NCIMasterGrowth5>
        <nciDbhDivisor>100.0</nciDbhDivisor>

NCI.DBH.div,3,nciDbhDivisor,
```

```
NCIMasterGrowth5,
NCI.DBH.div,100,100,100,100
```

This type could also be inside a grid

```
 <grid gridName="harvestmastercuts">
  <ma_plotLenX>100</ma_plotLenX>
  <ma_plotLenY>200</ma_plotLenY>
 </grid>

plotLenX,3,ma_plotLenX,
plotLenY,3,ma_plotLenY,

harvestmastercuts
plotLenX,100,100,100,100
plotLenY,200,200,200,200
```

Type 4: the parameter is like a type 2, but is inside a BehaviourList. The BehaviourList becomes part of the input parameter value file, since it is specific to the xml file that is being used. Note that type 4 must also contain a group name in the VariableName file. (This is because in a very few cases, the same variable is use multiple times in the xml file and we need the group name to figure out which case it is.)

```
 <NCIMasterGrowth5>
        <gr_nciMaxPotentialGrowth>
              <gr_nmpgVal species="Interior_Spruce">4.0152016</gr_nmpgVal>

NCI.MaxGrowth,4,gr_nmpgVal species,  gr_nciMaxPotentialGrowth

NCIMasterGrowth5,
NCI.MaxGrowth,1.65655768851703,2.22957895018928,1.75110747720752,2.37639294490124
```

Type 5: Output files. The code assumes that the parameter value file contains a directory. When the code writes this information to the xml file, it writes the directory specified in the parameter value file and then a unique file name. Currently the code is set-up to do Output and ShortOutput with specific file extensions. In this version of the code, other file types may not work, even if added to the VariablesName file.

```
ShortOutput,5,so_filename,
ShortOutput,C:\Projects\SORTIE\test\
```

Type 6: groups with species on the previous line.

```
 <tr_idVals whatSpecies="Lodgepole_Pine">
        <tr_initialDensity sizeClass="s4.0">25.0</tr_initialDensity>

Init.Dens.4,6,tr_initialDensity sizeClass="s4.0",tr_idVals
```

Init.Dens.4,0,0,75,0

Type 7: Disturbance types (e.g. Episodic Mortality). This is like the Behaviour types. In the parameter value file, the harvest type goes on one line and then the name and parameters for each species on the following line. The VariableNames file just requires the names and type. You would use this type to update species-specific parameter values (i.e. amount to harvest), as opposed to replacing the harvest layout (use .xml replace – section 4.2)

amountToCut,7,ds_amountToCut,

EpisodicMortality1
amountToCut,55,56,57,58

To add a new parameter to the VariableNames file:

1. Find the line in the base xml file that contains the right information.
2. See which of the current 7 types fit.
3. Make a note of the relevant parameter name(s) as written in the base xml file. The species name is not important, but the Behaviour List name or the group name may be important.
4. Look at the parameter value file and determine or assign the corresponding parameter name.
5. Edit the VariableNames file to add the parameter name (step 4), type (step 2), xml variable name(s) (step 3). Note that the order of the lines in the VariableNames file does not matter, so this new line may be added at any point in the file.
6. In the parameter value file, if the new variable is part of a Behaviour List, make sure the list name is in the preceding line. If not, make sure that the preceding line is 'na'. (Note, that the na or the BehaviourList name could be earlier in the file if there are other variable that are part of that grouping). Quotation marks are not necessary around either name.
7. Comments may be added to the VariableNames file by prefacing the line with "!".

Extracting Output Files

It is possible to automatically extract all the output files from one or more SORTIE runs.

The program will look in the directory:
C:/Projects/SORTIE/output/

It will then loop over all the tar files in that directory and extract them to the directory:
C:/Projects/SORTIE/output/extracted

This is a very rudimentary function at this point. It will always open all tarfiles in the output directory, and overwrite all files in the extracted directory.

Further functionality is forthcoming

This function returns a list of all xml files that are in the extracted directory.

Read Tree Output Files

The function ParseXML allows users to read one or more years from one or more xml file.

Inputs to the function include a list of one or more xml files, as well as a list of one or more years to extract.

Put the list of xml files into the variable "ListOfExtractedFile". *Note that this variable can also be filled automatically when using the ExtractFiles function described in the previous section.*

Put the list of years to extract into the variable "YearsToExtract"

The code will then run over each set of files and years to read the tree list file and to put the resulting information into an R variable:

trees_#

where # is the loop number. Right now there is no correspondence between the loop number and the file-year combination.

The new variable will have all the columns that Sortie printed to the output file, including one called ID. Note that ID simply represents the place in the output file that tree record contained. While this ID may refer to the same tree for some timesteps, mortality or establishment of that tree record or other trees will change the id. Use the x-y location to track trees instead as they will remain constant for the life of the tree.

Read Map Output Files

The function ParseMap allows users to read all the maps in an output xml file.

In the function, manually assign the name of the output file to read.

The code will scan the file to find all the maps that are present, then will loop over each map in the file, extract the information, and assign it to two different R variables.

The grid and information will be stored in an R variable called:

map_#

where # is the location in the output file.

The type of map is stored in a variable called:

output_type_#

where # is again the location in the output file. The type of the map is read from the output file and could be something like "Dispersed Seeds" or GLI Map 1".

Read Raster Files

There is a stand-alone file “RasterFunctions.r” that will read a raster file and write the information to a text file. This is a simple file whose goal is to allow users to quickly change a raster file into a format that can be pulled into an xml chunk and added to the Sortie file.

Input: raster file name
Output: an text file in xml format. Each cell that has a value greater than 0 will be printed as:

<ha_applyToCell x=”xloc” y="yloc’/>

Where xloc and yloc are the x-y locations in the raster file.