SIMO Documentation

Release 1.0.0

Simosol Oy

March 10, 2016

CONTENTS

Contents:

CHAPTER

ONE

INSTALLATION

Different versions can be downloaded from www.simo-project.org/download/releases/latest/.

1.1 Executable version

If you downloaded the executable distribution and decompressed the downloaded file, you've all set up. You can jump directly to *Using SIMO* section.

IMPORTANT: If you are using the executable distribution on Windows, you MUST install the Visual C++ 2008 Redistributable package or you will get cryptic errors on missing dlls or incorrect Side-by-side configuration.

1.2 Source version

The source distribution needs two more step besides decompression before it's usable.

You need to have Python programming language installed. You can get it from www.python.org. SIMO is not ready for Python 3, so download the latest Python 2.6.x version. Installing Python is usually necessary only on Windows, on Linux and OS X you should already have a working Python installation.

Besides Python, you'll need a Python package called setuptools. You can test whether you have setuptools installed by writing *easy_install* on the command line. If the output is something like "error: No urls, filenames, or requirements specified (see –help)" you're good to go, but if easy_install is not recognized as a command, download ez_setup.py and run:

```
python ez_setup.py
```

After you've got working versions of Python and setuptools, the next steps depend on whether you're using a Unix variant or Windows.

1.2.1 Linux and OS X users

On the command line in the directory into which you decompressed the downloaded SIMO source distribution, issue this command:

```
python build_env.py
```

Then go and grab a cup of coffee, the building process will take a good while. When it finishes you should have a working SIMO installation and you are ready for the *Using SIMO* section.

If you run into trouble in the *python build_env.py* stage, have a look at the *Known issues with SIMO installation* section.

Building a 32 & 64 bit Python on OS X 10.6 Snow Leopard

Buildout will build a Python for you within the buildout environment, but in addition to that you might want to build a general Python version that is more up to date than the system Python that Apple provides. These instructions are based on *this blog post<http://tinyurl.com/ybdsf43>*.

Making sure you have 32- and 64-bit enabled readline. First download the latest readline release and then:

```
export MACOSX_DEPLOYMENT_TARGET=10.6
export CFLAGS="-arch i386 -arch x86_64"
export LDFLAGS="-Wall -arch i386 -arch x86_64"
./configure
make
sudo make install
```

Download latest Python source release and:

```
./configure --enable-framework --with-universal-archs=intel \
            --enable-universalsdk=/Developer/SDKs/MacOSX10.6.sdk
```

Now you should have a working Python installation in /Library/Frameworks/Python.framework

1.2.2 Windows users

Python 2.6 (recommended):

1. Download and install Python 2.6 (if you didn't already)
2. Download and install the compatible C compiler: Visual C++ 2008 Express
3. Install setuptools, which also includes easy_install, using the exe-installer found here
4. Download sqlite3 amalgamation source
5. Download pysqlite source (.tar.gz):
6. Download and install something that can unpack tar and gzip files, (e.g. 7-zip
7. Unpack sqlite3 to a location of your choosing, start up the Visual Studio 2008 command prompt (under tools in the visual studio start menu), navigate to where you unpacked sqlite3 and run the following (two separate commands):

   ```
   cl /c /O2 /D SQLITE_ENABLE_LOAD_EXTENSION /D SQLITE_ENABLE_RTREE /D SQLITE_THREADSAFE /D HAVE

   link /LIB /OUT:sqlite3.lib sqlite3.obj
   ```

8. Unpack pysqlite to a location of your choosing, find the setup.cfg in the folder and alter it to match the following (where <sqlite_path> is where you just compiled sqlite3):

   ```
   [build_ext]
   include_dirs=<sqlite_path>
   library_dirs=<sqlite_path>
   libraries=sqlite3
   define=HAVE_LOAD_EXTENSION,SQLITE_ENABLE_RTREE
   ```

9. Now run (in the folder where you extracted the source):

   ```
   python setup.py build --force
   python setup.py install
   ```

Python 2.6+ (IF compiling fortran libraries):

1. Download and install the compatible C compiler: MinGW (make sure it's on PATH for simplicity)

2. In the mingw installation folder, copy liblibmingw32.a -> <python installation>libsmingw32.lib and liblibmingwex.a -> <python installation>libsmingwex.lib
3. In <python installation>libs run the following:

```
lib -remove:mbrtowc.o mingwex.lib
lib -remove:wcrtomb.o mingwex.lib
```

Python 2.5:

1. Download and install Python 2.5 (if you didn't already)
2. Download and install the compatible C compiler: MinGW (make sure it's on PATH for simplicity)
3. Install setuptools, which also includes easy_install, using the exe-installer found here
4. Download sqlite3 amalgamation source
5. Download and install something that can unpack tar and gzip files, (e.g. 7-zip
6. Unpack sqlite3 to a location of your choosing, navigate to that location in command prompt and run *with the mingw gcc*:

```
gcc -O2 -USQLITE_OMIT_LOAD_EXTENSION -DSQLITE_ENABLE_LOAD_EXTENSION=1 -DSQLITE_ENABLE_RTREE=1

gcc -shared -s -o sqlite3.dll *.o
```

7. Copy sqlite3.dll to C:WINDOWSSYSTEM32 (or to a folder in the path if you don't have access to system32). If you plan on using pyinstaller, you also need to copy sqlite3.dll to <pythonpath>DLLs
8. Download pysqlite source
9. Unpack pysqlite to a location of your choosing, find the setup.cfg in the folder and alter it to match the following (where <sqlite_path> is where you just compiled sqlite3):

```
[build_ext]
include_dirs=<sqlite_path>
library_dirs=<sqlite_path>
libraries=sqlite3
define=HAVE_LOAD_EXTENSION,SQLITE_ENABLE_RTREE
```

Now run (in the folder where you extracted the source):

```
python setup.py build --force -c mingw32 install --force
```

Common:

10. Download and run the PostgreSQL one click installer when asked if you would like to install additional packages, choose yes and install PostGIS (this can also be done afterwards with the included StackBuilder, but it's easier to do here) and let it make the template/example when it asks.
11. Download and install the latest version of psyocopg2
12. Download and install lxml latest exe-installer for your system/python
13. Download and install numpy exe-installer for your system/python.
14. Make sure you have the easy_install script in your system path (or go to the folder where it is) and execute the following in the command prompt

```
easy_install -Z cython (on 2.6 run in the  Visual Studio 2008 command prompt)
easy_install -Z ZODB3
easy_install -Z pygooglechart
easy_install -Z python-dateutil
easy_install -Z pyparsing
```

If easy_installing cython fails, you can get ready made Windows binaries

If easy_installing pyparsing, dry specifying the pyparsing package explicitly

```
easy_install http://cheeseshop.python.org/packages/source/p/pyparsing/pyparsing-1.5.5.tar.gz
```

16. If you want to run tests, you'll also need

```
easy_install nose
easy_install minimock
easy_install interlude
```

17. If you want to run code profiling (only for developers)

```
easy_install guppy
```

18. If any of the above easy_installs failed, see if there's an exe-installer available for your system and use that instead (or you can do it anyway, it shouldn't make any difference).
19. Download libiconv, geos, proj and libspatialite from the spatialite install site
20. Unzip the zips from the previous step and copy all dll-files from the bin folders into C:WINDOWSSYSTEM32 and/or to <SIMO_FOLDER>srcsimodbdatadb
21. Now you are ready to build a working SIMO installation. On the command prompt, while in the directory where you unzipped the source distribution, run

```
python build_env.py
```

Enabling parallel execution

If you wish to run simo in parallel, you will also need to do the following:

1. Download redis, unzip and copy the files to a location of your choosing (for example C:redis, adding it to the path is convenient for usage)
2. run easy_install -Z Celery
3. run easy_install -Z redis

1.3 SVN Checkout

There's a third option for installing SIMO if you want to track the latest and greatest developments. You can do a subversion checkout for the SIMO codebase and you can track the development day in day out. The checkout address is:

```
svn.simo-project.org/simo/trunk
```

After the checkout you essentially have the source distribution (plus a few bits and pieces), so follow the instructions above.

For Unix, change /etc/sysctl.conf (PLEASE backup this file first!) Linux: kernel.shmmax=value Mac OS: kern.sysv.shmmax=value FreeBSD: kern.ipc.shmmax=value

Note, if getconf is on your system, you can run the following to get good inital values for shmmax and shmall:

> #!/bin/bash # simple shmsetup script page_size=‘getconf PAGE_SIZE‘ phys_pages=‘getconf _PHYS_PAGES‘ shmall=‘expr $phys_pages / 2‘ shmmax=‘expr $shmall * $page_size‘ echo kernel.shmmax = $shmmax echo kernel.shmall = $shmall

sysctl.conf (on a 4 core 16GB RAM system)

kernel.shmall = 2057198 kernel.shmmax = 8426285056 # note, this value must be equal or greater than postgresql shared_buffers setting, though roughly half of your RAM is a good example. kernel.sem = 250 32000 100 128 # may need to increase these further if a large number of processes are on the same system # Potentially helpful, but untested settings # vm.swappiness=0 # try to avoid swapping; may hurt performance if you run out of memory # vm.overcommit_memory=2 # don't let processes allocate more memory than they need.

Also, consider setting 'noatime' to the database volume mount options to disable constantly updating file access times.

reboot (or sudo sysctl -p should also work)

Use pgtune (http://pgfoundry.org/projects/pgtune/), which on a 4 core 16GB RAM system resulted in the following (IMPORTANT! note the non pgtune wizard entries):

#————————————————————————— # CUSTOMIZED OPTIONS #———————————————————————————

SSD options (only if you have a seperate ssd) data_directory = '/ssd/postgresql_data' # use data in another directory seq_page_cost = 0.05 random_page_cost = 0.1

#custom_variable_classes = '' # list of custom variable class names default_statistics_target = 50 # pgtune wizard 2011-10-12 maintenance_work_mem = 960MB # pgtune wizard 2011-10-12 constraint_exclusion = on # pgtune wizard 2011-10-12 checkpoint_completion_target = 0.9 # pgtune wizard 2011-10-12 effective_cache_size = 11GB # pgtune wizard 2011-10-12 work_mem = 96MB # pgtune wizard 2011-10-12 work_mem = 128MB # book recommendation for speed wal_buffers = 8MB # pgtune wizard 2011-10-12 wal_buffers = 16MB # book recommendation for any server (but especially on heavy insert work) checkpoint_segments = 16 # pgtune wizard 2011-10-12 shared_buffers = 3840MB # pgtune wizard 2011-10-12 max_connections = 80 # pgtune wizard 2011-10-12 # BULK LOAD OPTIONS, DISABLE AFTER DONE!!! checkpoint_segments = 128 # or even 256? checkpoint_timeout = 1800 # in seconds; 30 min work_mem = 256MB # improves order by, distinct etc # synchronous_commit = off # this can be done clientside, so no need to do it here. Iit will mean however that the data may be malformed on catastrophe # fsync = off # may be unnecessary and could destroy the existing database if something goes wrong!!!

CHAPTER

TWO

INTRODUCTION

This documentation includes the description of SIMO framework. It's meant to give a general overview of the structure and use of the framework for the users and developers.

During the fall of 2004 the department of Forest Resource Management at the University of Helsinki embarked on a joint project together with Metsähallitus, Forestry Centres and Forestry Development Centre Tapio, Metsämannut Oy, Tornator Oy and UPM-Kymmene Oyj to raise the level of forest management planning in Finland as well as to improve the compatibility of planning systems. The project was given the name SIMO, as **SIM**ulation is used to produce alternative scenarios for forest management and **O**ptimization is used to choose the one matching the goals for planning the best.

The goals for the project weren't too modest:

```
The system should be flexible with regards to the data and models used.
It should be adaptable to different planning problems and extendable
to cover the future planning needs.
```

Hence, during the development attention was paid to enable use of data sources of varying content in the planning. Analysis of the suitable application domain for each model should reduce the errors in the planning results. The awareness of the users of these errors is raised by including a mechanism that warns the user when models are used outside their domain. The usability of models is extended by including a flexible mechanism for level correction.

Special attention has been paid to the matching of the planning data properties and the characteristics of the models used to process the data. In the documentation you'll frequently come across the term 'model chain'. It refers to a set of actions in which individual models are applied to the data to get the prediction of the value of the attribute under interest. One example of a model chain in a individual tree level simulator would be a sequence of models to predict the survival probability of trees; mortality due to competition, mortality due to aging, and the total mortality.

The simulations in SIMO are collections of these model chains. This is the key to the flexible assembly of different simulators that all utilize the SIMO framework.

CHAPTER

THREE

COMPONENTS OF THE SIMO FRAMEWORK

SIMO framework contains program modules designed for forest management planning. It's possible to combine different modules for different computation tasks. The properties of SIMO include easily modified control of planning computation and a set of model libraries that can be freely combined in the computation.

The different modules of the framework are data input, simulation, optimization and reporting. The modules are controlled with XML documents which contain the actual forest related information in the system. This enables the changes in the simulation logic without the need to resort to actual program code changes. The modules have been implemented mainly in Python. Some computationally heavy modules have been implemented in *Cython* and *C*.

During the data import the input data is converted to the form used internally in SIMO; this applies to the attribute names and units as well as the structure of the data. Simulation will produce different management alternatives for the forest area under planning. In optimization the alternative that best matches the goals and constraints of the planning is selected as the result of the planning. Reporting module is used to transfer data - input, simulation, or optimized - outside the framework. Different text and image based output formats are supported.

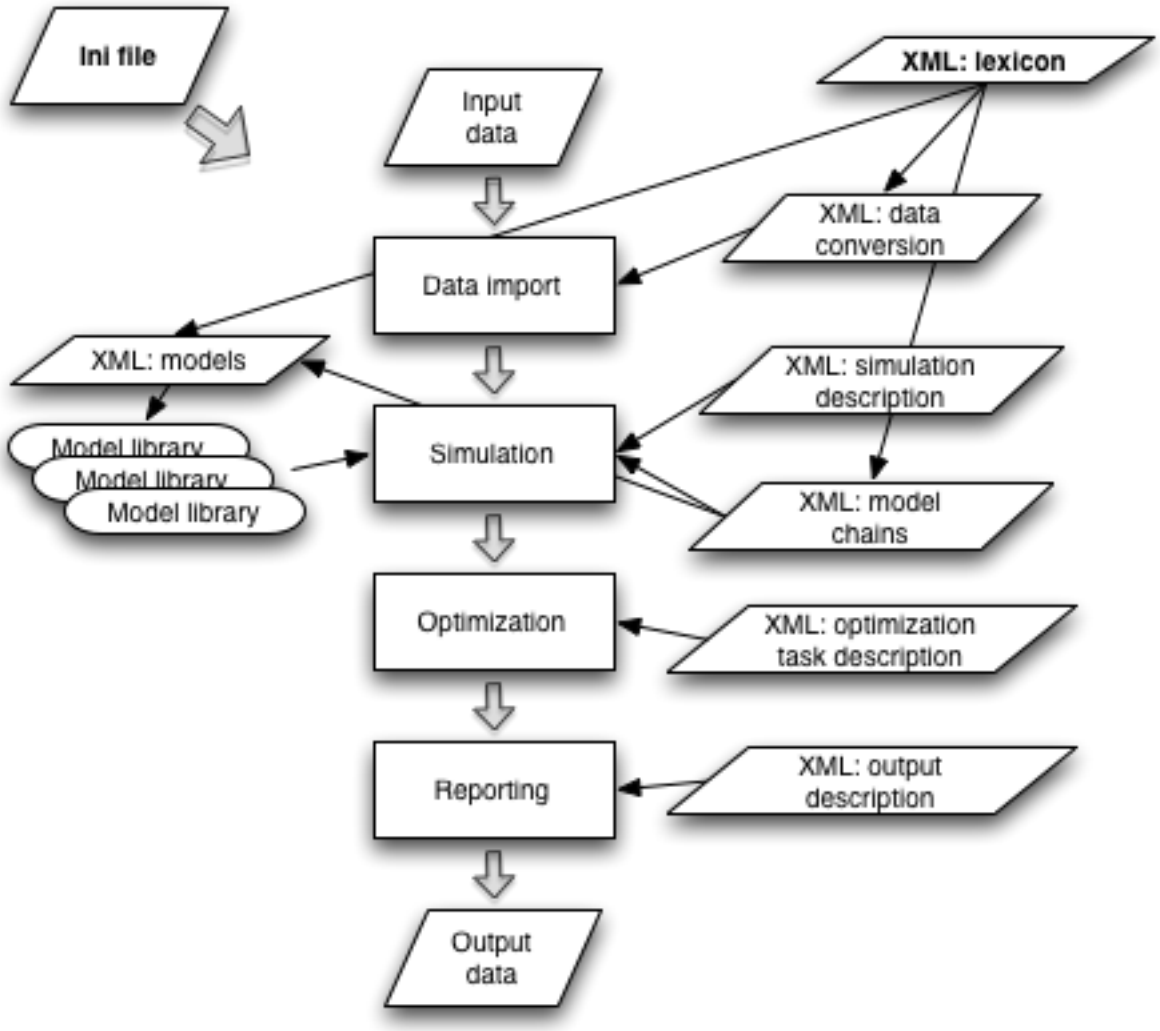

Figure 1. The components and the program flow of the SIMO framework

3.1 Data import module

The data import module is able to transform the input data into the internal format of SIMO. Currently SIMO has support for two types of text files as input data. The data is either in one text file for all data levels or each data level; e.g., stand, tree species stratum, tree; is in its own text file and the data in different files is linked together with the use of identifiers.

The data conversion for data import is described in XML documents.

3.1.1 Lexicon XML

The starting point for data import is the lexicon XML document (*Lexicon*) which describes the data levels used in the computation; e.g. stand-stratum-tree; and the attributes the objects at each level can have. For each attribute the name, unit of measurement, minimum and maximum values and the description are given in the lexicon.

3.1.2 Data conversion XML

A conversion mapping describes the relationship between the lexicon and the input data (*Data conversion*). With the help of the conversion mapping the structure of the data as well as the attribute names and units are converted to the format understood by SIMO during the data import.

3.2 Simulator module

The simulator module contains the implementation for simulation of forest growth and alternative management regimes. The module consists of the simulator program code, collections of implementations of models describing the forest, and the XML documents used to describe the simulation task.

3.2.1 The simulator implementation

There is no default simulation in the implementation of the simulator. At the initialisation the simulator will read in the data and the content of the XML documents describing the simulation. Therefore, the simulation can be changed by changing the content of the XML documents. During the simulation the simulator will interpret the computation tasks defined in the XML model chains and applies those to the data. The simulator will call the models in the model libraries to compute new attribute values and will emit log messages to the user about the errors in the input data as well as in the model chains.

3.2.2 XML definitions of the simulator

The simulation logic, attributes used and the details of the models are described in a set of XML documents. The basis is again set in the lexicon (*Lexicon*), which sets the way the forest is defined in the simulator. The top level description of a simulation is in a simulation XML document (*Simulation*). It contains the time periods used in the simulation as well as the parameter values and possible initial attribute values. Each simulation period definition also contains the model chains; i.e. the simulation logic; used in the simulation for that period (*Model chain*).

The models used in simulations are divided into two groups described by their own XML documents: the ones used to predict attribute values (*Prediction model*) and the ones describing the forestry operations (*Operation model*). These descriptions contain the model input attributes and parameters, output attributes as well as information about the restrictions for model usage and the data used to derive the model.

The cash flow XML documents (*Cash flow*) contain the income and costs associated with forestry operations.

3.2.3 Model libraries

The programmatic implementations of the prediction and operation models are collected into model libraries, which are shared program libraries (dll files on Windows, so files on Linux/Unix). The model libraries are dynamically attached to a simulation according to the content of the model chain, prediction- and operation model documents. Model libraries can be implemented either in C or in Python (*Prediction model library*, *Operation model library*).

The implemented prediction model library contains static and dynamic prediction model applicable to the forests of Finland. The models include both tree and stand level models. The implemented operation models include models for thinnings and final cuts as well as soil preparation, planting, natural regeneration, seedling tending models.

3.2.4 The steps of simulation

To make simulation with SIMO one has to complete a series of steps:

- create model libraries, i.e. implement the models
- write model chains that utilise the model libraries
- compose a simulation description of the model chains, and set needed parameter values
- simulate

When creating a model library one has to take care that the variables used in the models match the ones defined in the lexicon (*Lexicon*). The models in the libraries must also be documented in the appropriate model description documents (*Prediction model*, *Operation model*).

The model chains decompose the simulation into a series of tasks. Each task is either fulfilled with a model or it is further divided into a series of subtasks. At the end each branch of a model chain ends into a model. The execution of each task in a model chain can be conditional.

The actual simulation is then composed by defining the set of model chains that it consists of. In addition different control parameters for the simulation are set: the start year of the simulation, the length of the simulation period, the number of periods, possible stopping conditions for the simulation and a host of other parameters including the names of the attributes that should be stored into the result database during the simulation. Should one want to change the names of the data levels used in the simulation, it must be done at this phase by modifying the lexicon, model, model chain and simulation xml documents.

3.3 Optimization module

The optimization module contains tools for solving an optimization task. In the optimization the combination of the all simulated alternatives is selected which best fulfills the goals and constraints of the optimization task. The solution is stored in a database for later reporting.

The implementation of the optimization module contains currently two heuristic optimization algorithms and an interface to a linear programming optimization package J.

The optimization task is described in an XML document containing definitions for the goal and the constraints (*Optimization*).

3.4 Reporting module

The role of the reporting module is to produce output from the input, simulated and optimized data. The output can be generated for individual computation units; e.g., stands, or for the whole planning area.

Supported data result text formats are *inlined*, *by_level* and *smt*. For these formats the output attributes are defined in an XML document (*Output constraint*). Format *smt* is meant to be used for single data level data reporting and it conforms to the structure used by the MELA planning system by the Finnish Forest Research Institute.

For aggregating values over time and location, there is an output format *aggregation* which outputs aggregated values (sum, mean, max, min) for attributes. The aggregation rules and output options are defined in an XML document (*Aggregation definition*).

Very close to *aggregation* report format is *expression*, which instead of aggration over simulation units, reports values by simulation unit (see *Expression definition*)

The forestry operation results are reported with the format *operation_result*.

branching_graph format outputs dot text files that can be converted into images with the Graphviz program. Dot files are generated per computation unit and they describe the branching of the simulation results as well as the forestry operation causing each branch generation.

CHAPTER

FOUR

USING SIMO

4.1 Shortcut for the impatient

Your first SIMO run will consist of two commands. The first one will build your SIMO simulator and optimizer and the next command will run a data import, simulation, optimizatio and reporting cycle using those and a demo data set. Later, you'll only need to run the *builder* command if you have changed the XML documents used to describe the simulator & optimizer in SIMO.

4.1.1 On Windows

First, open the Command Shell. From the Windows Start-menu select *Run...* (XP) or the search field (Vista) and type **cmd**

Change to working directory to the SIMO installation directory using the *cd* command, and then in that directory:

```
bin\builder.exe build simulator\builder.ini
bin\runner.exe simulator\runner.ini
```

If you wish to run SIMO as a server, you need to first run the following:

```
bin\builder.exe build simulator\builder.ini
```

Then, all of the following need to be running at the same time:

```
redis-server
bin\server.exe simulator\server_runner.ini
bin\worker.exe
```

After which you can run the following to start runs:

```
bin\control.exe
```

If you're using SIMO source distribution, you won't have the bin -executables and you have to run SIMO using the following commands:

```
python src\builder.py build simulator\builder.ini
python src\runner.py simulator\runner.ini
```

If you wish to run SIMO as a server, you need to first run the following:

```
python src\builder.py build simulator\builder.ini
```

Then, all of the following need to be running at the same time:

```
redis-server
python src\simoserver\simoserver.py simulator\server_runner.ini
python src\simoserver\run_celeryd.py
```

After which you can run the following to start runs:

```
python src\simoserver\simoctrl.py
```

4.1.2 On Linux & OS X

SIMO is run in the terminal from the installation directory:

```
bin/builder build simulator/builder.ini
bin/runner simulator/runner.ini
```

If you wish to run SIMO as a server, you need to first run the following:

```
bin/builder build simulator/builder.ini
```

Then, all of the following need to be running at the same time:

```
bin/redis-server
bin/server simulator/server_runner.ini
bin/server_celeryd
```

After which you can run the following to start runs:

```
bin/server_control
```

4.2 Structure of your SIMO installation

To use SIMO, you have three executables in the *bin* directory in your SIMO installation directory:

```
/bin
   builder.exe or builder
   runner.exe or runner
   logger.exe or logger
```

If you're using the source distribution on Windows, there won't be the *bin* directory. Instead you'll have these files in the *src* directory:

```
/src
   builder.py
   runner.py
   logger.py
```

The three executable files (or Python files) are the modules used for running SIMO. The default setup is to run the programs from the root SIMO directory.

In addition to those, the *simulator* directory in your SIMO installation directory contains important bits and pieces used to construct the simulator and optimizer you'll be using:

```
/simulator
   /db
   /input
   /models
      /aggregation
      /cash_flow
      /geo_table
      /management
      /operation
      /prediction
   /output
   /xml
      /aggregation_models
      /cash_flows
      /conversions
```

```
    /geo_tables
    /management_models
    /model_chains
    /operation_models
    /parameter_tables
    /prediction_models
    /samples
    /schemas
    /translation
  builder.ini
  exporter.ini
  lister.ini
  runner.ini
```

The files *builder.ini* and *runner.ini* are parameter files for controlling the execution of the *builder* and *runner* modules, respectively. The name and the location of the ini-files can naturally be different to the one given above.

In the folder structure above the *./simulator/xml* folder contains the XML documents that the user can modify. The *./simulator/models* folder contains the model libraries for the various model types (dll and py files). The *./simulator/db* folder contains the internal SIMO databases, whereas the *./simulator/input* and *./simulator/output* folders are reserved for the input data and reports.

The above folder structure is given only as illustration of a possible configuration. The specific location of different components is defined in the ini-files.

4.3 Building simulators & optimizers with builder

Builder parses and validates XML documents, constructs SIMO objects from the XML contents and stores the objects into an object database. Builder has three commands; build, list and export; and it's run like this (build command):

```
bin/builder build simulator/builder.ini
```

For source distribution on Windows, which lacks the *bin* versions, the command is:

```
python src/builder.py build simulator/builder.ini
```

You'll need to run the builder build command once at the beginning after a fresh installation and after that only if you make changes to the XML documents in the simulator directory.

4.3.1 Content of builder.ini for building SIMO instances

The content of the builder.ini-file is broken down into separate sections for which details are given below. The file paths in the ini-file can be given either as absolute or relative paths. One can also add comments in the file. The comments must begin with the # sign.

Program execution:

```
[program_execution]
```

The folder which is used a the base folder for the relative path settings used in the ini file:

```
base_folder= C:\Users\Joe\SIMO\simulator
```

Logging:

```
[logging]
```

Whether the log messages during program execution are written on the screen, and if console output is on, what level messages are logged on the screen:

```
console=TRUE
console_level=info
```

If file is given, log messages are written to the file, and if file output is on, what level messages are logged:

```
file=buildlog.txt
file_level=error
```

Database:

```
[simo_database]
```

Path to SIMO object database file:

```
path=db/simo.db
```

Option *create_new* defines whether new database should be created and existing database deleted. Option *create_zip* zips the object database file and creates a .ver file to accompany it. The .ver file contains a timestamp for the zipped object db. Normally you won't be using this, but it gives you an option to build basic versioning for your SIMO dbs:

```
create_new=True
timestamp=False
create_zip=False
```

Typedef:

```
[typedef]
```

Path to SIMO type definition schema document:

```
path=xml/schemas/Typedefs_SIMO.xsd
```

Schema:

```
[schema]
```

Schema section contains the paths to all required SIMO schema documents. Schema documents are used for validating the XML document structure and contents:

```
lexicon=xml/schemas/lexicon.xsd
message_translation=xml/schemas/message_translation_table.xsd
lexicon_translation=xml/schemas/lexicon_translation_table.xsd
text2data=xml/schemas/text2data.xsd
operation2modelchains=xml/schemas/operation2modelchains.xsd
text2operation=xml/schemas/text2operation.xsd
modelchain=xml/schemas/model_chain.xsd
simulation_control=xml/schemas/simulation.xsd
problem_definition=xml/schemas/optimization_task.xsd
output_constraint=xml/schemas/output_constraint.xsd
random_values=xml/schemas/random_variables.xsd
aggregation_definition=xml/schemas/report_definition.xsd
```

Schema.modelbase:

```
[schema.modelbase]
```

SIMO model classes also require schema documents for validating the model interfaces:

```
aggregation=xml/schemas/aggregation_modelbase.xsd
cash_flow=xml/schemas/cash_flow_modelbase.xsd
cash_flow_table=xml/schemas/cash_flow_table.xsd
geo_table=xml/schemas/geo_table.xsd
management=xml/schemas/management_modelbase.xsd
operation=xml/schemas/operation_modelbase.xsd
```

```
parameter_table=xml/schemas/parameter_table.xsd
prediction=xml/schemas/prediction_modelbase.xsd
```

XML:

```
[xml]
```

XML section controls which XML documents builder parses, validates and stores into SIMO object database:

```
# for the xmls several files or folders may be given
# separate the list items with ;
# if you merge lexicon from multiple XML documents, the master lexicon
# should be defined first, and the extra lexicons after that, separated by ;
lexicon=xml/lexicon.xml
message_translation=xml/translation/message_translation.xml
lexicon_translation=xml/translation/lexicon_translation.xml
text2data=xml/conversions/MELA2SIMO.xml
text2operation=xml/conversions/SMU2SIMO.xml
operation2modelchains=xml/operation_models/operation_mapping.xml
problem_definition=xml/samples/optimization_task.xml
output_constraint=xml/samples/result_variables.xml
random_values=
simulation_control=xml/samples/simulation.xml;xml/samples/simulation_price_model.xml
aggregation_definition=xml/samples/aggregation_def.xml
```

XML modelbase:

```
[xml.modelbase]
```

All models must have XML definitions, or interfaces, where model inputs and output etc. are defined. These include XML documents describing the prediction models (*Prediction model*), operation models (*Operation model*), cash flow tables (*Cash flow*), parameter tables (*Parameter table*) and location dependent tables (*Geo table*):::

```
aggregation=xml/aggregation_models/aggregation_models.xml
cash_flow=xml/cash_flows/cash_flow_models.xml
cash_flow_table=xml/cash_flows/cash_flow_table.xml
geo_table=xml/geo_tables/geo_table.xml
management=xml/management_models/management_models.xml
operation=xml/operation_models/operation_model.xml
parameter_table=xml/parameter_tables/parameter_table.xml
prediction=xml/prediction_models/prediction_model.xml
```

XML modelchain:

```
[xml.modelchain]
```

XML.modelchain section controls the paths from where XML model chain documents (*Model chain*) are searched and processed:

```
# Note that these are directories, not individual files
init=xml/model_chains/tree_simulator/init/
simulation=xml/model_chains/tree_simulator/
forced_operation=xml/model_chains/tree_simulator/forced_operation/
operation=xml/model_chains/tree_simulator/operation/
```

Executable modelbase:

```
[executable.modelbase]
```

Executable modelbase contains the paths to directories where the model implementations, or model libraries, are located during runtime. These model implementations include prediction models (*Prediction model library*), operation models (*Operation model library*) and location dependent geo tables (*Geotable*).:

```
aggregation=models/aggregation
cash_flow=models/cash_flow
```

```
cash_flow_table=xml/cash_flows
geo_table=models/geo_table
management=models/management
operation=models/operation
parameter_table=xml/parameter_tables
prediction=models/prediction
```

4.3.2 Content of lister.ini for listing object db content

The beginning of the ini-file used to control the listing of SIMO object database content is the same as for the build command.

Which objects are listed is controlled by setting values for keys in three sections. If the key does not have a value it's names won't be listed. The exact value is irrelevant; e.g. key=1 to export or key= to not:

```
[general]
lexicon=1
message_translation=1
lexicon_translation=1
text2data=1
text2operation=1
operation2modelchains=1
simulation_control=1
problem_definition=1
output_constraint=1
random_values=
aggregation_definition=1

[modelbase]
# The names are listed by type, define which types you want below.
aggregation=1
cash_flow=1
cash_flow_table=1
geo_table=1
management=1
parameter_table=1
prediction=1
operation=1

[modelchain]
# Likewise modelchains are listed by type.
init=1
simulation=1
operation=1
forced_operation=1
```

4.3.3 Content of exporter.ini for object to XML converison of SIMO instances

You can run:

```
./bin/builder export ./simulator/exporter.ini
```

To export the object database content back to XML documents. Again, the ini-file has the same content at the beginning as the ini for build command.

As above, exporting is defined in three sections, in which key-value pairs control whether exporting is done or not. The value for a key should be the path to which the resulting files from that item are to be written. These paths will be prepended with the *result_folder* value in *[program_execution]* section:

```
[general_export]
lexicon=lexicon
message_translation=translation
lexicon_translation=translation
text2data=data_conversion
text2operation=operation_conversion
operation2modelchains=operation_conversion
simulation_control=simulation
problem_definition=optimization
output_constraint=output_constraint
random_values=
aggregation_definition=aggregation_def

[model_export]
# The models are exported by type, define which types you want below.
aggregation=models/aggregation
cash_flow=models/cash_flow
cash_flow_table=models/cash_flow_table
geo_table=models/geo_table
management=models/management
operation=models/operation
parameter_table=models/parameter_table
prediction=models/prediction

[modelchain_export]
# Likewise modelchains are exported by type.
init=modelchains/init
simulation=modelchains/simulation
forced_operation=modelchains/forced_operation
operation=modelchains/operation
```

4.4 Running simulations & optimizations with runner

The actual data import, simulation, optimisation and reporting modules can be run using the **runner** executable:

```
bin/runner simulator/runner.ini
```

For source distribution on Windows, which lacks the *bin* versions, the command is:

```
python src/runner.py simulator/runner.ini
```

4.4.1 Developer option for debugging

You can give -d or –debug option for runner for the program execution to fall to a debugger (pdb) in case there is an unexpected exception during the program run:

```
bin/runner -d simulator/runner.ini
```

4.4.2 Content of runner.ini

The content of the runner.ini-file is broken down into separate sections for which details are given below. The file paths in the ini-file can be given either as absolute or relative paths. One can also add comments in the file. The comments must begin with the # sign.

Program execution:

```
[program_execution]
```

The folder which is used a the base folder for the relative path settings used in the ini file:

```
base_folder= C:\Users\Joe\SIMO\simulator
```

Run ID is used for identifying separate program runs:

```
run_id=demo
```

The following options define which of the four modules are run during the program execution:

```
import_data=TRUE
simulate=TRUE
optimize=TRUE
output_data=FALSE
```

The parameters are boolean values, for which the allowed values are:

- 1, TRUE, True, true, Yes, yes, y, Y, on
- 0, FALSE, False, false, No, no, n, N, off

In some cases, you might want to retain the contents already in the database you're dealing with, especially when dealing with concurrency, so the following options allow you to prevent a wipe of the existing database.

no_wipe_import=FALSE no_wipe_simulate=FALSE no_wipe_optimize=FALSE

A specialized case, which is mostly relevant for parallel execution, requires returning the SQL to be run locally on the server instead of remotely from the workers. These should usually be False (and altered at runtime) and will be harmful in a regular run, as the data is not stored anywhere.

no_exec_import=FALSE no_exec_simulate=FALSE no_exec_optimize=FALSE

Code profiling (boolean value) is normally switched off. It's used in code development for profiling program execution times:

```
code_profiling=off
```

Message logging:

```
[logging]
```

Boolean value indicating whether old log messages should be wiped out from the log database:

```
wipe_log=1
```

Whether the log messages during program execution are written on the screen. If console output is on, what level messages are logged on the screen:

```
console=1
console_level=info
```

Possible values are: debug, info, warning, error, critical

Debug-level contains all the messages possible including info, warning, error and critical messages. Similarly error level contains only error and critical messages.

Message translation:

```
[translation]
```

Log message translation is defined with twoXML documents: message_translation parameter refers to a file containing the message body translations, lexicon_translation contains the data level and attribute name translations. The desired message language setting must match a language code used in the XML documents. The XML document names are given without the xml-extension or path:

```
message_translation=message_translation
lexicon_translation=lexicon_translation
lang=fi
```

Error handling:

```
[error_handling]
```

These settings affect when the simulation is terminated: if max_error_count_total is exceeded during the simulation (including data import), the whole simulation is terminated. By setting this value to negative value, the simulation run is not terminated because of data import or simulation errors. If max_error_count_per_unit is exceeded for any given simulation unit, the simulation for that unit is terminated and the simulaiton proceeds with the next simulation unit:

```
max_error_count_total=100
max_error_count_per_unit=10
```

If the parameter suppress_location is set to true, the error message will not contain description which identifies the location in a model chain the error originated from:

```
suppress_location=False
```

Warning handling:

```
[warning_handling]
```

Despite the logging level setting, the warning messages can be suppressed from the logs by setting the output_warnings to false. The suppress_location setting works similarly to the one for error handling:

```
output_warnings=True
suppress_location=False
```

SIMO database:

```
[simo_db]
```

Path to SIMO object database. SIMO object database is for storing the SIMO's internal data structures constructed from the XML documents:

```
db_path=db/simo.db
```

DATA database:

```
[data_db]
```

The DATA database is used for storing simulation input and result data. The name of the data level having the simulation units (like stands or plots) must be defined here:

```
simulation_level=comp_unit
```

db_type can be either sqlite, postgresql or oracle:

```
db_type=sqlite
```

Spatial database is needed if spatial properties of the data should be included in the simulation. If spatial is set to true, the spatial reference system id for the data needs to be defined. It's the EPSG code for the coordinate system (see: http://spatialreference.org/ref/epsg/). The diminfo-setting needs to be defined if the db_type is oracle. Diminfo is a list of four values separated with ;. The values are the minimum x coordinate in the data, the minimum y coordinate, and maximum x and y coordinates (values in the coordinate system defined with the srid setting:

    spatial=false srid=3067 # diminfo: minx;miny;maxx;maxy diminfo=

Threaded-setting controls whether a threaded connection to the database is used (currently Oracle only), and unlogged-setting whether the database uses unlogged tables (currently PostgreSQL only). WARNING: when usin unlogged tables, the database can't be recovered after a crash or unclean shutdown. This means that _all_ data from the database will be lost; i.e. after opening it again after a db crash, it will be empty.:

```
threaded=false
unlogged=false
```

```
[data_db.sqlite]
```

SQLite specific settings: database in memory (nothing saved to hard drive, but faster), and the location of the on-disk storage of the databases:

```
in_memory_db=false
db_dir=db
log_db=log.db
input_db=input.db
operation_db=operation.db
result_db=simulated.db
optimized_db=optimal.db

[data_db.postgresql]
```

PostgreSQL specific settings: table prefixes for each type of tables (to separate the tables that are stored in the same schema), server connection parameters:

```
log_prefix=log
input_prefix=input
operation_prefix=operation
result_prefix=result
optimized_prefix=optimized
db_host=localhost
db_port=5432
db_user=test
db_pw=test
```

If the database with the given name doesn't exist, it will be created, the same applies for the schema within the database:

```
db_name=simodb
db_schema=test

[data_db.oracle]
```

Oracle specific settings: table prefixes for each type of tables (to separate the tables that are stored in the same schema), server connection parameters:

```
log_prefix=log
input_prefix=input
operation_prefix=operation
result_prefix=result
optimized_prefix=optimized
db_host=localhost:1521/orcl
db_user=test
db_pw=test
```

Data import:

```
[import]
format=inlined
```

Allowed values: *by_level, inlined*

by_level data has data for different data levels in different files, whereas in *inlined* data the different data levels are in the same file.

Text to data conversion mapping file defines how the input data values are imported into SIMO. Mapping definition XML document is given without the xml-extension or path:

```
mapping_definition=MELA2SIMO
```

Boolean value indicating whether date information should be imported from input data:

```
import_date=True
```

Data date must be given regardless of the import_date. Date is given given as d.m.yyyy. If import_date=TRUE, data_date value is used as data date in those cases where date is missing from data. If import_date=FALSE, data_date value is always used.

data_date=1.1.2009

Data delimiter in the input text file. Whitespace as delimiter given as ' '. First line (i.e. header line) is skipped if skip_first_line=FALSE.

data_delimiter=' ' skip_first_line=False

Input data directory and file(s) are defined with options *data_dir* and *data_file*. Multiple data files are delimited with ;. If the format is by_level there are several data files. The files are given in the top-down order of data levels separated by a semicolon; e.g., stand.txt;stratum.txt;tree.txt::

```
data_dir=input
data_file=data.rsu
```

Operation import:

```
[import.operations]
```

So-called *forced operations* can be imported if operation_import=TRUE. Execution of forced operations is predefined in the operation input data:

```
operation_import=TRUE
```

Forced operations can be imported from various formats. The allowed formats are: xml, text, db. Operation input file(s) are defined with *operation_dir* option:

```
operation_format=text
operation_dir=input
```

Operation conversion defined how the operations are imported into SIMO. Only *text* format requires operation_conversion definition, as xml and db are native SIMO operation import formats. More info on operation conversion can be found from operation mapping XML document (*Operation conversion*). Operation implementation logic is defined in model chains and thus *operation_modelchains* option is required. Forced operations input file is defined with option *operation_file*:

```
operation_conversion=SMU2SIMO
operation_modelchains=operation_mapping
operation_file=data.smu
```

Simulation:

```
[simulation]
```

Simulation control XML document (*Simulation*) is the high-level simulation control definition, containing the simulation time span definitions, model chains, initial variable values and output constraints. Simulation control XML document name is given without the xml-extension or path:

```
simulation_control=simulation
```

Simulation *task_type* must be either *simulation* (creates a result db) or *data_processing* (modifies the input db):

```
task_type=simulation
```

In data processing the input data is modified; i.e. the computation will modify the values in the input database. Data processing can contain both init, simulation and forced operation chains. In simulation the data from the input database is taken as is, and a result database is generated.:

```
deterministic=True
track_prices=True
```

Deterministic-setting, if set to True, removes any stochasticity from the results. If track_prices is set to true, the timber assortment unit prices used for each main level object in operations are stored in the result database.

Next four settings relate to forestry operation simulation: when set to False, *do_forced_ops* causes the simulation of forced operations; i.e, operations that are set to happen on the simulation unit on a given date, to be skipped altogether. When dealing with forced operations and Monte Carlo simulations (more than 1 realizations per unit), there are two options: 1. either each unit-iteration has individual set of forced operations, or 2. each unit has a single set of forced operations which should be used for all realizations (iterations). In the former case *copy_ops_to_all_iterations* should be FALSE, and in the latter case the option should be TRUE and forced ops will be copied from iteration 0 to all other iterations.

Option *create_branch_desc* controls whether for each simulation unit, a description of each decision tree branch will be generated or not. The description contains the name of each operation that has caused branching in the decision tree. *cache_dir* is the directory path in which the unit price and stem volume caches will be written to. The 'simulation' level variable *USE_CACHE* defined in the simulation_control (set to 1 to use caching) will control whether the caches will be used or not. See later for examples of simulation control:

```
do_forced_ops=True
copy_ops_to_all_iterations=True
create_branch_desc=True
cache_dir = cache
```

Option *use_data_date_as_init_date* sets whether the date stored in input data will be used as the starting date for the simulation. It's also possible to use the date from the computer's clock as the initial date (*use_today_as_init_date*). If both of these settings are set to false, the date given in *fixed_init_date* is used as the starting date for the simulation. The date is given in d.m.yyyy format. If the task type is 'data_processing' there are two options for how dates are handled during computation, by setting *fix_data_processing_date_to_init* to True the date for the data after the data processing run is the same as the date used for the starting date for the computation. On the other hand if the same option is set to False, the date for the data after the computation will be whatever is dictated by the simulation control used for the computation; i.e., if the simulation control contains a 10 year simulation, the data date after the run will be the starting date + 10 years. Note that for 'simulation' type tasks the results will be date stamped to the end of each simulation period but for 'data processing' type tasks they will be data stamped to the beginning of the next period (unless a fixed date is used). For example for a simulation control with one 1 year period and a starting date of 1.1.2010, the results from a 'simulation' type run will have a date of 31.12.2010. For a 'data processing' type run the result date will be 1.1.2011.:

```
use_data_date_as_init_date=False
use_today_as_init_date=False
fixed_init_date=1.1.2008
fix_data_processing_date_to_init=False
```

The following lines can be modified to improve data execution times. Over- and underestimates can, however, hurt execution time. Option *object_count_multiplier* is a positive non-zero number, *estimated_branch_count* is a positive non-zero integer number, *max_unit_count* is a positive non-zero integer and number that sets the amount of simulation units that are calculated at a time. Option *max_task_count* is a positive non-zero integer number that sets the amount of simulation units that are sent from the server to a worker at once (and has no effect if the program is not in server mode). f.e. max_unit_count 30 max_task_count 300 will have workers working on 10 set groups for faster execution time. Option *max_result_unit_count* control the maximum number of result units from prediction models (e.g. new trees from distribution models):

```
object_count_multiplier=50
estimated_branch_count=1
max_unit_count=1
max_task_count=300
max_result_unit_count=60
```

Simulator can be forced to either always leave a single untouched branch or never leave untouched (no-op) branches. This setting works "globally", meaning that it will consider all branching groups simultaneously. If you DON'T want to leave a branch with no operations (*leave_no_op_branch* = False), but you have, for example, separate branching groups for thinnings and clearcuts, and the stand is at a development state in that is past thinning stage, i.e. only clearcut branches will be simulated. In this case the last clearcut will still leave the untouched branch, as the thinnings haven't been yet simulated. The untouched branch would only 'disappear' if all the possi-

ble thinning and clearcut branches would be simulated. This can be avoided by using *separate_branching_groups* option, which when set to True, will process each branching group separately and you can have multiple branching groups without the problem of "untouched" branches appearing. Option *separate_branching_groups* is set to False by default.

leave_no_op_branch=True separate_branching_groups=False

In cases where branching is very dense, the number of generated branches may cause problems memory- and performance-wise. These problems can be avoided by setting a limit for maximum number of branches, using option *max_branch_limit*.

max_branch_limit=100

If one wants to simulate only a subset of the input data, individual simulation units can be listed in the *id_list* parameter by listing their ids separated by a semicolon. Note that the parameter is commented out in the example. index_sequence parameter can be used to direct a certain sequence of simulation units to simulation, in the example first 10 units will be simulated. Setting the parameter value to 99- all the simulation units from the 100th onwards will be simulated:

```
#id_list=567;773;5367
index_sequence=0-9
```

Optimization:

```
[optimization]
```

Optimisation method should be one of: *hero*, *tabu_sarch*, *jlp*:

```
method=hero
```

Jlp will use linear programming (LP) to find a global optimum. Tabu search and HERO are heuristic optimization methods, which don't necessarily find the global optimum but are more flexible for the optimization problem formulation.

The optimization task is defined in a SIMO optimization task object, constructed from optimization_task XML document (*Optimization*). What is given as the value of the problem_definition setting, will be used as the optimization run id in the database as well:

```
problem_definition=optimization_task
keep_opt_data = False
reuse_opt_data = False
```

Boolean setting *keep_opt_data* controls whether the optimization data file will be left on disk after the optimization run or not. The data file get its name from the *problem_definition* setting. Connected to the *keep_opt_data* setting, the boolean setting *reuse_opt_data* controls whether an existing optimization data file is used or not. You can reuse the data file if you've run the optimization once before and the expressions and constraints in the optimization task have not changed; e.g. when you run the optimization with a different heuristic method or change the parameters for a method. The data file is the one called *opt_task.h5* in the base folder set earlier:

Setting *limit2branches_include* and *limit2branches_exclude* can be used to limit the optimization to just specific branches of the data, e.g.:

```
limit2branches_include=Normal;5 yrs
limit2branches_exclude=10 yrs
```

would only pick those alternative development scenarios that would have the string 'Normal' or '5 yrs' in their branch names, and not the string '10 yrs'; i.e. one has to list the strings wanted to appear in the branch name in *limit2branches_include*, and the strings that must not appear in the branch name in the *limit2branches_exclude*. Strings are separated by semicolon. The match can be partial, i.e. the whole branch name does not have to be given. The branch names can be found in the simulation result database from the table branch_desc in the branch_desc column. They are generated from the name-attributes of the <condition> child elements of <branch_conditions> of the modelchain XML document (*Model chain*) used to describe the alternative branch generation logic in the simulation. Note that a single branch name will contain a combination of the condition names if there are several branching events for the particular branch over the whole simulation period.

Note that even if used branches are limited, the branch 0 is always included in the optimization data, so you should probably set *leave_no_op_branch* to true so that branch 0 does not contain any branching operations.

Normally though, these settings are left empty so that the optimization uses all simulated alternatives for optimal solution candidates:

```
limit2branches=
```

In the case of heuristic optimization, a couple of images illustrating the optimization procedure are generated in the folder defined in the *chart_path* parameter:

```
chart_path=.\logs
```

In the case of heuristic optimization, a couple of images illustrating the optimization procedure are generated in the folder defined in the *chart_path* parameter.

For the LP optimization using J, the installation folder of J must be given as well as the prefix used for all the files that SIMO generates for J. The value of max_objects should be increased if the optimization using J fails to memory related errors. Also, the timeout value is dependent on the problem size. It determines the time in seconds that the optimization is allowed to run with J. This is because at certain corner cases (signifying a bug in J), J can drop to it's internal prompt while executing the optimization thus leading to indefinite optimization execution without the timeout. Use big values for the timeout with large programs, otherwise the optimization will be terminated prematurely due to the timeout.

In case you want to split optimized operations according to the split unit weights generated by J, set the *do_split* parameter to True. OBS! The split weights DO NOT affect simulated data values in any way, only the operations!!! The optimizer will multiple all numerical op_res columns in the optimized database with the given weights and this will cause trouble if you have some categorical variables in your results, such as tree species or timber assortment. This can be avoided by defining the columns you do not want to split in the *skip_weight_splits* parameter:

```
[optimization.jlp]
j_folder=./J
file_prefix=demo
max_objects=2000
timeout=30
do_split=False
skip_weight_splits=species;assortment
```

Because of the iterative nature of optimum search in heuristic methods, the maximum number of iterations used are given. For tabu search also two other parameters are given: for how many iterations is the current solution fixed as the optimum solution (entry_tenure) and for how many iterations is a certain solution kept outside the optimum solution once it's been removed from it (exit_tenure):

```
[optimization.tabusearch]
maximum_iterations=100000
entry_tenure=10
exit_tenure=20

[optimization.hero]
maximum_iterations=100000
```

Reporting:

```
[output]
data_type=result
result_type=optimized
start_date=1.1.2000
end_date=1.1.2010
full_date=false
#id_list=567;773;5367
#index_sequence=10-24
```

The output can be used to output values from input or result data (data_type). For the output format operation_result this setting has no effect as for that format the data is always taken from the result database.

Allowed values: input, result

If the data type is result, the result type can be either simulated or optimized (result_type).

Allowed values: simulated, optimized

It's also possible to limit the output to certain time period, using *start_date* and *end_date*. By leaving these options empty, all the dates are reported. By leaving the end_date value empty, all periods beginning from the start_date are reported.

If *full_date* is true, month and day are printed in the output. Otherwise only the year is printed.

Similarly to simulation, the reporting can be targeted to only specific simulation units (see above for definitions of id_list and index_sequence).

It's possible to select several output formats at the same time by separating the format names with a semicolon. There must be a matching number of output filenames again separated with a semicolon:

```
output_directory=.
output_format=aggregation;inlined;operation_result
output_filename=aggr;stock;oper
output_constraint_file=output_constraints
default_decimal_places=1
archiving=False
```

Allowed output formats:

- *inlined, by_level*: data from different levels and different simulation branches in the same output file
- *smt* - data for a single data level from the first data branch (usually the one generated by optimization module) for the last period in the simulation. Optionally several smt files can be generated, each for a single simulation period.
- *aggregation*: for aggregating values over time and location
- *operation_result*: the forestry operation results
- *branching_graph* format outputs dot text files that can be converted into images with the Graphviz program. Dot files are generated per computation unit and they describe the branching of the simulation results as well as the forestry operation causing each branch generation.

output_constraint_filename is given only for the formats inlined, by_level and smt. It should refer to a SIMO object, constructed from an XML document describing the attributes for different data levels that should be in the report (*Output constraint*). The XML document name is given without the xml-extension or path.

The text output files can be automatically compressed (archiving).

For inlined-format, result padding will align the column names in the header and the data values:

```
[output.inlined]
result_padding = True
```

Aggregation rules and output options are defined in an XML document (*Aggregation definition*). XML document name is given in aggregation_definition_file option without xml-extension or path:

```
[output.aggregation]
aggregation_definition_file=aggregation
aggregation_charts_directory=.
```

For operation_result-format vars-parameter defines which of the operation model result variables should be included in the operation result report. The result variables are described in the operation model XML document (*Operation model*). In addition of those there is an explicit result variable cash_flow. Operation results contain also categorical variables such as tree species and assortment. Operation results can be grouped by these categorical variables by defining the variable names using group_by parameter:

```
[output.operation_result]
vars=cash_flow;Volume
group_by=SP;assortment
```

4.5 What happened in that run? Enter logger

Logger module **logger** is used for retrieving information about simulation and optimisation from log database. Info on logger usage can be get with command:

```
bin/logger --help
```

For source distribution on Windows, which lacks the *bin* versions, the command is:

```
python src/logger.py --help
```

CHAPTER

FIVE

USING THE POSTGRES DATABASE PROVIDED BY BUILDOUT

To initialize Postgres database to store data in simulator/db/pg directory:

```
./parts/postgresql/bin/initdb -D simulator/db/pg
```

and then to start it daemonized:

```
./parts/postgresql/bin/pg_ctl -D simulator/db/pg -l simulator/log/pgsql.log start
```

Next, you'll need to create the Postgres user used in the simulation (default is test). You can do this with the command line SQL interface psql or the createuser command (see ./parts/postgresql/bin/createuser –help):

```
./parts/postgresql/bin/psql postgres
```

and then in psql:

```
CREATE USER test WITH PASSWORD 'test' CREATEDB;
```

Now you're ready to run the test simulation with Postgre as database once you've changed the db_type ini-setting:

```
[data_db]
# db_type must be sqlite or postgresql.
db_type=postgresql
```

CHAPTER

SIX

DOING THINGS IN PARALLEL

First up, you'll need a few more things installed if you wish to use Simo in parallel. First of all, you need to install celery, postgresql (if you didn't already) and a redis server.

Now, in the source folder, there are additional runners, for this, you'll want to use the celery_runner to start up the runner host, which functions mostly as the regular runner, except that instead of running the simulations, it'll pass the jobs to celery workers via amqp.

As for the workers, they are located in the *dev/celery_example* folder. Both the server and the workers are, without any changes, going to be expecting all the pieces to be running on localhost, with postgresql having a user test with password test. For example, if you wish to run postgresql offsite (running server and postgresql at the same place highly recommended for speed), you'll need to change it in the runner.ini, celeryconfig.py (which _must_ be on the path) and celery_tasks (which _also_ needs to be on the path). The workers are started using celeryd, a program provided by the celery installation. No parameters neccessary.

The proper order of starting things up is PostgreSQL, redis, celery_runner, celeryd. For rabbitmq installation and startup instructions for different platforms, see these instructions

So, in case you're on a unix system, and have run buildout (and builder) successfully, you'll need to ensure the rabbitmq and postgresql database are running, then execute:

```
bin/redis-server
bin/python src/celery_runner.py simulator/runner.ini
```

Then, wherever you're running the worker(s):

```
bin/celeryd
```

Do note that you'll need to have a replica of the simulation setup on the worker host; i.e., a matching copy of the SIMO db.

Also note that all the parallel runs use the default celery queues in rabbitmq, so you should really be running a single run per master server, as otherwise the different simulation tasks from different runs will happily mix in the queue and the workers will get tasks from all of the runs. Therefore, if you abort a run, make sure to reset the rabbitmq queues. This is accomplished with:

```
bin/celeryd --purge
```

While the celery_runner, celery_tasks combo is functional, these are more for demonstrative purposes; you should consider making your own server and workers based on them, and your needs, rather than exclusively relying on them.

6.1 Customizing celeryconfig.py

It is quite likely that you'll need to alter *celeryconfig.py* to fit in your environment.

All settings starting **BROKER_** refer to the amqp server, and should match it's settings.

CELERY_RESULT_BACKEND chooses the method via which the workers return their results to the server. We prefer using redis, as it seems to be the most reliable option at this time. The **REDIS_** settings refer to the redis server settings.

CELERY_IMPORTS defines the file from which all the possible tasks are imported, by default *celery_tasks.py*. This needs to be on the path.

CELERYD_CONCURRENCY allows you to define how many workers are on this server. If not set, it defaults to the amount of available cores.

CELERYD_PREFETCH_MULTIPLIER defines how many tasks a worker preloads to itself. This should be set to 1 so that the tasks are executed as workers become available.

CELERY_REDIRECT_STDOUTS allows you to get print statements to the log/output. If it is set to False, all STDOUT output is lost.

CELERYD_LOG_LEVEL defines how detailed output from the workers is printed out. Allowed values are DEBUG, INFO, WARNING and ERROR. STDOUT, if redirected will be on WARNING level of output.

For further reading about this, see the documentation

6.2 Customizing celery_tasks.py

As mentioned above, this is the file pointed by the CELERY_IMPORTS setting. The key things to change are:

```
base_dir -- the location of the simo libraries (usually ./src/simo)
sim_dir -- the local equivalent for the configuration variable
           program_execution.base_folder
pg_host -- address of the postgresql server
pg_port -- port of the postgresql server
pg_user -- username on the postgresql server
pg_pw -- password of the user on the postgresql server
```

The workers use the postgresql information to get the configuration stored there, as opposed to using the local ini file, overriding some key values as stated above.

6.3 When things go wrong

If you need to abort a running parallel job for one reason or another, the unfinished tasks are left in the rabbitmq queue. The clear the queue, you can use celeryd –purge, but to do things directly on rabbitmq you can also:

```
sudo rabbitmqctl stop_app
sudo rabbitmqctl reset
sudo rabbitmqctl start_app
```

CHAPTER

SEVEN

XML DOCUMENTS IN SIMO

This chapter contains descriptions of the XML documents that form the basis of adaptability and extensibility of the SIMO framework. The documents are illustrated through examples. The most detailed description of the XML documents are contained in the XML Schema files, which define the exact structure for the documents. The schema files are available through the SIMO development website at http://trac.simo-project.org/browser/simo/trunk/simulator/xml/schemas.

7.1 Lexicon

The lexicon document defines the data hierarchy and vocabulary used. It's also used to document the system and to enable validity checks of other documents. The lexicon contains definitions for all attributes used in the data and models.

The lexicon is constructed as a hierarchy of data levels, in which the root level is always `simulation`. Each data level contains the definitions for its attributes and possibly its sublevel.

The attributes are divided into two sets: numerical and categorical attributes.

An example of the document contents is given below, for a comprehensive description of all the possible content options see the schema document. Data continuation and abbreviated content expressed as ...:

The root level tag contains a reference to the schema document which is used to validate the content of the XML document:

```
<rootlevel xmlns="http://www.simo-project.org/simo"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:schemaLocation="http://www.simo-project.org/simo ../schemas/lexicon.xsd">
```

Here the simulation level attribute definitions are not shown. The simulation level has as its sublevel the first proper data level, here called `comp_unit`:

```
...
   <name>simulation</name>
   <num_vars>...</num_vars>
   <cat_vars>...</cat_vars>
   <sublevels>
      <sub_level>
         <name>comp_unit</name>
         <type>static</type>
```

An example of a numerical attribute definition: the name is ALT, that will be used to refer to this attribute in model definitions and model chains. The unit is metre, minimum allowed value is 0 m and the maximum value 600 m. If these limits are exceeded in the data, a warning is automatically generated:

```
...
         <num_vars>
            <variable>
               <name>ALT</name>
               <unit>m</unit>
```

```
            <min_value>0</min_value>
            <max_value>600</max_value>
            <description>Altitude above see level</description>
         </variable>
         ...
      </num_vars>
```

comp_unit level has also categorical variables (cat_vars). For categorical variables all the possible values are described in the lexicon. The PEAT attribute has two possible values: 0 meaning mineral soil and 1 meaning peatland:

```
...
      <cat_vars>
         <variable>
            <name>PEAT</name>
            <min_value>0</min_value>
            <max_value>1</max_value>
            <description>Categorical variable referring to peatland
            </description>
            <values>
               <enum>
                  <value>0</value>
                  <description>Mineral soil</description>
               </enum>
               <enum>
                  <value>1</value>
                  <description>Peatland</description>
               </enum>
            </values>
         </variable>
         ...
      </cat_vars>
```

comp_unit has also a sublevel called stratum which has attributes of its own. The structure of the stratum data level definition repeats that of the previous data levels:

```
...
      <sublevels>
         <sublevel>
            <name>stratum</name>
            <type>dynamic</type>
            <num_vars>
               <variable>
                  <name>Age</name>
                  <unit>year</unit>
                  <min_value>0</min_value>
                  <max_value>350</max_value>
                  <description>Biological mean age</description>
               </variable>
            ...
            </num_vars>
            ...
         </sublevel>
      </sublevels>
   </sublevel>
</sublevels>
</rootlevel>
```

7.2 Merging multiple lexicons

Different projects that share the same basic XML components might have some project-specific variables. That means that each project should have it's own lexicon, or that there would be a single lexicon with all of the project-

specific variables. Both of these solutions would cause some problems. A better solution is to have a single master lexicon, with all of the common variables (which includes most of the variables) and a number of project-specific extra lexicons, which are in their own XML documents.

Two or more lexicons can be merged together if the data hierarchies are compatible. This means that the level-names and parent-child relations in the master and extra lexicons are similar. The master lexicon should define the whole data model, but the extra lexicon does not need to contain the whole data hierarchy. For example, if the master lexicon defines a simple data hierarchy with levels "simulation-comp_unit-stratum-tree", the extra lexicons could have only levels "simulation-comp_unit-stratum" or "simulation-comp_unit", if there are no project specific variables in the lower levels.

7.3 Model chain

The model chains define how the models are applied to the data to get the desired results out of the simulation. Each simulation usually consists of several model chains which are applied to the data in the order defined in the simulation XML document (*Simulation*).

An example of the document contents is given below, for a comprehensive description of all the possible content options see the schema document. Data continuation and abbreviated content expressed as . . . :

The root level tag contains a reference to the schema document which is used to validate the content of the XML document:

```
<model_chains xmlns="http://www.simo-project.org/simo"
              xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              xsi:schemaLocation="http://www.simo-project.org/simo ../../schemas/model_chain.xsd">
```

Model chain content is grouped into chain groups. Their function is just to clarify the content of the document. The actual simulation logic is contained in the model chains (model_chain). Each model chain is associated to a data level (evaluate_at); i.e., the tasks of the chain are only applied to the data objects of that data level:

```
...
   <chain_group name="Biomass of growing stock, dry weight in kilograms">
      <model_chain name="calculate biomass of tree" evaluate_at="tree">
```

Each model chain consists of a set of tasks. The name of the task is used if warning or error messages need to generated. The execution of a task can depend on a condition; in the example the task "calculate if trees exist" is executed if the *stratum* level attribute *seedling_below13* value equals (*eq*) 0:

```
...
         <task name="calculate if trees exist">
            <condition>stratum:seedling_below13 eq 0</condition>
```

Each task may have sub tasks; here the "calculate if trees exist" task has a sub task "Calculate biomass of a tree". If a task is not divided into further sub tasks, it has a model associated with it; here *Biomass_tree_JKK*. The name of the model must match the one defined in the prediction model document (*Prediction model*) if its type is *prediction* (for other options see below):

```
...
            <task name="Calculate biomass of a tree">
               <model>
                  <name>Biomass_tree_JKK</name>
                  <prediction/>
               </model>
            </task>
         </task>
      </model_chain>
   </chain_group>
</model_chains>
```

Each model chain can also have number of branching_groups, which are used in generating so-called branches, or, alternative development paths. Each branching_group consists of a number of branch_tasks. A branch_task has attribute 'branching_operations' which contains operations that actually create a new branch.

OBS! Notice that even if you have separate branching groups for thinnings and clearcuts, the branching group name attribute should be equal for the separate groups. Such as "Normal harvest" as in the example below.

A branching group element for regeneration harvests (clearcut & seedtree_position)

```
...
        <branching_group name="Normal harvest">
           <branch_task branching_operations="seedtree_position;clearcut"
              name="Clearcut or seedtree cut">
```

A branching group element for thinnings (thinning & seedtree cut)

```
...
        <branching_group name="Normal harvest">
           <branch_task branching_operations="first_thinning;thinning"
              name="First thinning or normal thinning">
```

Notice that the branching group name, "Normal harvest", is the same for the both branching groups.

Each branch task has normal condition element, number of normal tasks and a branch_conditions -element, which includes a number of normal condition elements. Each of these branch_condition elements can create a new branch if the condition is fulfilled:

```
...
            <branch_conditions>
               <condition name="Normal clearcut">
                   comp_unit:passed_regen_limit gt 0</condition>
               <condition name="Delayed clearcut, 5 years">
                   comp_unit:passed_regen_limit gt 5</condition>
               <condition name="Delayed clearcut, 10 years">
                   comp_unit:passed_regen_limit gt 10</condition>
            </branch_conditions>
            <condition>comp_unit:MAIN_GROUP eq 1 and
               comp_unit:REGENERABLE eq 1</condition>
            <task name="Clearcut">
...
```

7.3.1 Different types of conditions

Condition operators

The allowed values for operators used in conditions are:

Data attribute related:

- gt: greater than
- ge: greater than or equal
- eq: equal
- ue: not equal
- le: less than or equal
- lt: less than
- in: first operand in the values listed by the second operand
- and
- or
- exists: data level/attribute value exists
- not exists: data level/attribute value doesn't exists

Forestry operation related:

- times_eq: the operation has been executed this many times
- times_lt: the operation has been executed less than this many times
- times_gt: the operation has been executed more than this many times
- since_eq: the last execution of the operation was this many time steps ago
- since_ue: the last execution of the operation was not this many time steps ago
- since_lt: the execution of the operation was less than this many time steps ago
- since_gt: the execution of the operation was more than this many time steps ago

Different operands for the expression

The condition consists of expressions that may be nested. Each expression has two operands that are compared with an operator (see above).

There are several possible two-operand combinations:

The most common is the one comparing an attribute value to a fixed value (see above).

The attribute value can also be compared to another attribute value. Here the *D_gM* attribute from *comp_unit* level should have at least equal value to the *regen_dgm* attribute value from the *comp_unit* level:

```
<condition>comp_unit:D_gM ge comp_unit:regen_dgm</condition>
```

A set of allowed values can also be given for an attribute. In this case the operator used is always *ìn*. Here the *SP* attribute from *stratum* level must have one of the values 3, 4, 5, 6, 7, 9:

```
<condition>stratum:SP in [3, 4, 5, 6, 7, 9]</condition>
```

The conditions can also test for operation execution. Here more than 10 years must have passed since the last thinning:

```
<condition>comp_unit:thinning since_gt 10</condition>
```

As for the data attributes, the operation execution can be compared to a attribute value. Here more years than defined by the *simulation* level attribute *REMOVE_SEEDTREES_YEARS_SINCE_REGEN_CUT_PINE* must have passed since the latest *comp_unit* operation *seedtree_position*. Note that if an operation has never been done, the since_eq, since_gt and since_lt will evaluate true:

```
<condition>comp_unit:seedtree_position
           since_gt
           simulation:REMOVE_SEEDTREES_YEARS_SINCE_REGEN_CUT_PINE"
 </condition>
```

The condition can also test for existence of a data level in the data:

```
<condition>stratum exists</condition>
<condition>stratum not exists</condition>
```

Finally, it's possible to construct a more complicated condition by using the *and* and *or* operators and grouping the expressions with parenthesis:

```
((comp_unit:AGE gt 1.0 and comp_unit:SC le 4.0)
  or
  stratum:SP in [1.0, 2.0, 3.0])
and
((comp_unit:thinning since_gt 10
  or comp_unit:thinning since_gt comp_unit:past_thinning)
  or stratum not exists)
```

7.3.2 Different types of models

Six different types of models can be used in model chains.

Prediction model

```
<model>
   <name>Development_class_JKK</name>
   <prediction/>
</model>
```

Aggregation model

When aggregation operand variable and/or weight have default attributes, missing (NaN) values are substituted with the default values before passing the values to the actual aggregation model implementation.

Aggregation model:

```
<model>
   <name>sum</name>
   <aggregation>
      <target>N</target>
      <operands>
         <operand>
            <variable default="1">N</variable>
            <weight default="0">BA</weight>
            <level>stratum</level>
         </operand>
      </operands>
   </aggregation>
</model>
```

Aggregation model targets and operands can be "vectorised", i.e. multiple variables or values instead of single variable or value, separated with whitespace (i.e. a single space). Note that target vector and operand vectors (value, variable and weight) must contain equal number of items, and the used aggregation model must have vector implementation:

```
<model>
   <name>assign_value</name>
   <aggregation>
      <target>N BA V D_gM H_gM</target>
      <operands>
         <operand>
            <value>0 0 0 0 0</value>
         </operand>
      </operands>
   </aggregation>
</model>
```

Special data level 'self'

Conditions can also be used in aggregation models to limit the data that is being passed to the aggregation model. In these conditions a special data level name 'self' must be used to refer to the target model chain evaluation level objects. E.g. the aggregation model below computes the sum of the products of 'ga' and 'n' variables for trees that have bigger diameter ('tree:d') than the tree ('self:d') for which the sum of products is computed for:

```
<task
    name="Calculate relative density factor of trees larger than the
          object tree">
```

```
    <model>
        <name>sum_products</name>
        <aggregation>
            <target>RDF_L</target>
            <through_level>comp_unit</through_level>
            <operands>
                <operand>
                    <variable>ga</variable>
                    <level>tree</level>
                </operand>
                <operand>
                    <variable>n</variable>
                    <level>tree</level>
                </operand>
            </operands>
            <condition>tree:d gt self:d</condition>
        </aggregation>
    </model>
</task>
```

Similarly, this 'comp_unit' level task computes the 'tree' level weighted average for only those threes that have a bigger diameter than the 'comp_unit' level mean diameter (self:D_gM):

```
<task name="Calculate mean crown ratio for dominant trees">
    <model>
        <name>w_average</name>
        <aggregation>
            <target>CR_dom</target>
            <operands>
                <operand>
                    <variable>cr</variable>
                    <weight>n</weight>
                    <level>tree</level>
                </operand>
            </operands>
            <condition>tree:d gt self:D_gM</condition>
        </aggregation>
    </model>
</task>
```

Operation model

erase_memory – will clear the prediction model memory for all objects for which the operation model is executed if set to true. db_names – optional name and group to be saved to result database (as opposed to the name and group of the operation) notes – optional notes on the operation, will be saved to the result database materials – optional materials needed to perform the operation, will be saved to the result database

```
<model>
   <name>clearcut</name>
   <operation>
      <simulation_effects>
         <erase_memory>false</erase_memory>
      </simulation_effects>
      <db_names>
          <name>clearcut</name>
          <group>final_harvest_artificial_regeneration</group>
      </db_names>
      <notes>This is a clearcut</notes>
      <materials>
          <material>Fuel</material>
      </materials>
      <parameters>
```

```
            <parameter>
               <name>LOG_REDUCTION_PINE</name>
               <value>10</value>
            </parameter>
            <parameter>
               <name>LOG_REDUCTION_SPRUCE</name>
               <value>8</value>
            </parameter>
            <parameter>
               <name>LOG_REDUCTION_OTHER</name>
               <value>15</value>
            </parameter>
         </parameters>
         <cash_flow_table>timber_prices</cash_flow_table>
      </operation>
</model>
```

Management model

```
<model desc="Delete all objects from level stratum">
   <name>remove_objects_from_child_level</name>
   <management>
      <parameters>
         <parameter>
            <level>stratum</level>
         </parameter>
      </parameters>
   </management>
</model>
```

Parameter table model

```
<model>
   <name>regeneration_limits</name>
   <parameter_table/>
</model>
```

Geo table model

```
<model>
   <name>dem</name>
   <geo_table>
      <all_vars>true</all_vars>
   </geo_table>
</model>
```

7.4 Prediction model

The prediction model document contains the definitions for the prediction models used in the simulation. It serves two purposes: as a documentation for the models, and as "glue" between the simulator and the model libraries (*Prediction model library*).

An example of the document contents is given below, for a comprehensive description of all the possible content options see the schema document. Data continuation and abbreviated content expressed as . . . :

The root level tag contains a reference to the schema document which is used to validate the content of the XML document:

```
<modelbase xmlns="http://www.simo-project.org/simo"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:schemaLocation="http://www.simo-project.org/simo
           ../schemas/prediction_modelbase.xsd">
```

The models are divided into named model groups, each group containing several model definitions.

The model definition begins with the name of the model. The name must match the name of the function in the model library which implements the model. The implementation details are given after the model name. Possible implementation programming languages (implemented_in) are C and Python:

```
...
   <model_group name="Distribution models">
      ...
      <model>
         <name>Diameter_distribution_spruce_percent_point_KangasMaltamo
         </name>
         <implemented_at>DistributionModels.dll</implemented_at>
         <implemented_in>C</implemented_in>
```

The model definition continues with model metadata, which is not used in the actual simulation:

```
...
         <author>
            <name>Annika Kangas</name>
            <name>Matti Maltamo</name>
         </author>
         <description>Generate trees on mineral soils; stand's
                     basal area weigted median diameter, basal area, age,
                     stems/ha and site class as attributes
         </description>
         <published_in>Kangas, A. and Maltamo, M., 2000.
                      Percentile based basal area diameter distribution
                      models for scots pine, norway spruce and birch
                      species. Silva Fennica 34(4): 371-380.
                      </published_in>
         <species_list>
            <species>2</species>
         </species_list>
         <geographical_coverage>Entire country. Tested with NFI
                               material
         </geographical_coverage>
         <applies_for>Mineral soils. Number of trees measured.
                     BA greater than 0 m2/ha, BA_total greater than 0
                     m2/ha, D_gM greater than 0 cm, Age greater than 0
                     years and N greater than 0 trees per hectare.
         </applies_for>
         <research_material>Values given as (min, mean, max):
                           G (3, 14.8, 34.2), N (75.2, 1062.4, 2944.9),
                           Volume (13.1, 110.2, 282.9),
                           Volume_log (0, 86.1, 282.4),
                           D_gM (6, 19.9, 38), d_min (2, 5.3, 18),
                           d_max (14, 36.6, 66), Age (25, 82.7, 177),
                           total 416 sample plots
         </research_material>
```

Model definition data used in the simulation begins with the model variable definitions. The variable name and data level definitions here must match those found in the lexicon (*Lexicon*). They must also match the function input parameter definitions for the model implementation in the model library (*Prediction model library*). Note that for that the order of the variable definitions is also important. Lower and upper limit values for each variable are optional:

```
...
         <variables>
            <variable>
```

```
            <name>BA</name>
            <level>stratum</level>
            <limits>
               <lower_limit>0</lower_limit>
            </limits>
         </variable>
         <variable>
            <name>BA</name>
            <level>comp_unit</level>
            <limits>
               <lower_limit>0</lower_limit>
            </limits>
         </variable>
         ...
         <variable>
            <name>MAX_CLASS_COUNT</name>
            <level>simulation</level>
         </variable>
      </variables>
```

After the input variables, the result attributes of the model are defined. Usually the value of the `object`-element is `self` meaning that the result values are stored for the object calling the model. In this example, however, the value is `tree` which means that the model generates new objects at the `tree` data level and sets the `d` and `n` attribute values for the new objects to the values returned by the model:

```
...
      <result>
         <object>tree</object>
         <variables>
            <variable>
               <name>d</name>
            </variable>
            <variable>
               <name>n</name>
            </variable>
         </variables>
      </result>
   </model>
</model_group>
</modelbase>
```

7.5 Operation model

The operation model document contains the definitions for the forestry operation models used in the simulation. It serves two purposes: as a documentation for the models, and as "glue" between the simulator and the model libraries (*Operation model library*).

An example of the document contents is given below, for a comprehensive description of all the possible content options see the schema document. Data continuation and abbreviated content expressed as ...:

The root level tag contains a reference to the schema document which is used to validate the content of the XML document:

```
<operation_modelbase xmlns="http://www.simo-project.org/simo"
                     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                     xsi:schemaLocation="http://www.simo-project.org/simo
                     ../schemas/operation_modelbase.xsd">
```

The models are divided into named operation groups, each group containing several model definitions:

```
...
   <operation_group name="thinning">
      <operation>
```

The model definition begins with the name of the model, which must match the name of the model in the model library. In case the type of the model is `model` (see below for alternatives), the model library name and implementation language are given. The cash flow handler may either be the model itself, the simulator or `none`. In the `model`-case the model itself returns the prediction of the cash flow associated with the operation. In case of `simulator` the simulator get's the cash flow prediction from a cash flow table (*Cash flow*), and if the cash flow handler is set to `none` there won't be any cash flow results stored for the operation model. If interaction is set "on", the model can modify the simulation data objects at different levels; i.e., the thinning model removes trees from the simulation. To put it in other words, if interaction is "off", only operation results are processed (not cashflows, simulation effects etc.):

```
...
         <name>low_thinning</name>
         <type>
            <model>
               <implemented_at>SimoOperations.py</implemented_at>
               <implemented_in>python</implemented_in>
               <cash_flow_handler>model</cash_flow_handler>
               <interaction>on</interaction>
            </model>
         </type>
```

Metadata about the operation models consists of the verbal description of the model:

```
...
         <description>Basic implementation of a thinning model. Removes
                      diameter classes systematically starting from the
                      smallest diameter class.
         </description>
```

Next up is the definition about the data that is passed to the operation model. For each data level a group is defined which contains the level name and the level attributes collected for each object on the data level:

```
...
         <data>
            <group>
               <level>comp_unit</level>
               <variables>
                  <variable>
                     <name>Remainlimit_upper</name>
                  </variable>
                  <variable>
                     <name>Remainlimit_lower</name>
                  </variable>
               </variables>
            </group>
            <group>
               <level>stratum</level>
               <variables>
                  <variable>...</variable>
               </variables>
            </group>
            <group>
               <level>tree</level>
               <variables>...</variables>
            </group>
         </data>
```

Another operation model input data category is parameters. The parameter values are set in the model chain documents (*Model chain*); i.e., they do not get their values from the simulation data:

```
...
        <parameters>
           <parameter>
              <name>target_density</name>
           </parameter>
           <parameter>
              <name>track_width</name>
           </parameter>
           ...
        </parameters>
```

Operation models can also get parameter tables as input data. Parameter tables are defined in model interface like this:

```
...
        <parameter_tables>
           <parameter_table>thinning_preference_order</parameter_table>
           <parameter_table>log_reduction</parameter_table>
        </parameter_tables>
```

A weight to be applied to the operation results can also be defined; i.e., it's the multiplier for the result attribute values:

```
...
        <operation_result_weight>
           <name>AREA</name>
           <level>comp_unit</level>
        </operation_result_weight>
```

The operation model results are either operation results (operation_result) or data attribute values (data, see below). The operation results can have classifier values associated with then; e.g., the harvested volume classified by the species and the timber assortment. The classifier values can be taken either from data (variable name, data level) of from the cash flow tables used for the operation model (cashflow_classifier):

```
...
        <results>
           <type>
              <operation_result/>
           </type>
           <variables>
              <variable>
                 <name>Volume</name>
                 <classifiers>
                    <classifier>
                       <variable>SP</variable>
                       <level>stratum</level>
                    </classifier>
                    <classifier>
                       <cashflow_classifier>
                          <variable>assortment</variable>
                       </cashflow_classifier>
                    </classifier>
                 </classifiers>
              </variable>
              ...
           </variables>
        </results>
     </operation>
  </operation_group>
</operation_modelbase>
```

7.5.1 Variation: operation model type

If the model type is *cash_flow*, the model definition is straightforward: the vital details are the model name and the optional result weight. The actual cash flow table (*Cash flow*) used for the model is set in the model chain (*Model chain*):

```
...
        <operation>
           <name>grass_suppression</name>
           <type>
              <cash_flow/>
           </type>
           <description>Mechanical grass_suppression - heinays</description>
           <operation_result_weight>
              <name>AREA</name>
              <level>comp_unit</level>
           </operation_result_weight>
        </operation>
```

7.5.2 Variation: operation result type

If the model result type is *data*, the data level is defined for the results as well as the data attributes that the operation model returns:

```
...
        <results>
           <type>
              <data>
                 <target>existing</target>
                 <level>stratum</level>
              </data>
           </type>
           <variables>
              <variable>
                 <name>stand_value</name>
              </variable>
              <variable>
                 <name>LOGp</name>
              </variable>
              <variable>
                 <name>COMWOOD</name>
              </variable>
              <variable>
                 <name>V_LOG</name>
              </variable>
           </variables>
        </results>
```

7.5.3 Variation: result variable type

Default result variable type is *scalar*, but if operation results should be distributed to different points in time, result variable type should be *date_array*. The result data structure constructed by the simulator will be different for scalar and date_array variables. It is the responsibility of the operation model implementation to return correctly built result data structures back to the simulator.

Below is a scalar result variable (it would be scalar also without the type attribute):

```
...
        <variables>
           <variable type="scalar">
              <name>stand_value</name>
```

```
            </variable>
         <variables>
```

Below is a date_array result variable:

```
...
         <variables>
            <variable type="date_array">
               <name>C_flow</name>
            </variable>
         <variables>
```

7.6 Aggregation model

The aggregarion model document defines the aggregation models used.

An example of the document contents is given below, for a comprehensive description of all the possible content options see the schema document. Data continuation and abbreviated content expressed as . . . :

The root level tag contains a reference to the schema document which is used to validate the content of the XML document:

```
<aggregation_base
   xmlns="http://www.simo-project.org/simo"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.simo-project.org/simo ../schemas/aggregation_modelbase.xsd">
```

An example of aggregation model: the name of the model is *sum*, that will be used to refer to this model in model chains. *simoaggr.py* contains the implementation of the model. <*vector_implementation*> is xs:bool value indicating whether the aggregation model can handle vectors, i.e. multiple target attributes simultaneously:

```
...
   <model>
   <name>sum</name>
   <description>Sums up either values of two operands or values of a single operand (a list of va
   <implemented_at>simoaggr.py</implemented_at>
   <vector_implementation>false</vector_implementation>
   </model>
   ...
</aggregation_base>
```

7.7 Management model

The management model document defines management models used.

An example of the document contents is given below, for a comprehensive description of all the possible content options see the schema document. Data continuation and abbreviated content expressed as . . . :

The root level tag contains a reference to the schema document which is used to validate the content of the XML document:

```
<management_base xmlns="http://www.simo-project.org/simo"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.simo-project.org/simo ../schemas/management_modelbase.xsd">
```

An example of management model named *remove_objects_from_child_level*, that name will be used to refer to this model in model chains. In the model the data level is given to define the level, which objects will be deleted:

```
...
   <model>
      <name>remove_objects_from_child_level</name>
```

```
      <description>For the current evaluation level objects, delete all
           child objects from the given level. Delete also objects from
           sublevels.
      </description>
      <implemented_at>management.py</implemented_at>
      <parameters>
         <parameter>
            <type>level</type>
            <description>delete objects from this level</description>
         </parameter>
      </parameters>
   </model>
   ...
</management_base>
```

7.8 Parameter table

The parameter table document defines parameter models used.

An example of the document contents is given below, for a comprehensive description of all the possible content options see the schema document. Data continuation and abbreviated content expressed as ...:

The root level tag contains a reference to the schema document which is used to validate the content of the XML document:

```
<parameter_tables xmlns="http://www.simo-project.org/simo"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://www.simo-project.org/simo
         ../schemas/parameter_table.xsd">
```

A parameter table includes always classifiers. Here a parameter table named "regeneration_limits_TAPIO" is used as an example. The parameter table has attributes *MANAGEMENT_REGION*, *TAPIO_cultivation* and *SC* as classifiers from `comp_unit` level:

```
...
   <parameter_table name="regeneration_limits_TAPIO" type="parameter_table"
      desc="Regeneration mean diameter limits by geographical area, Tapio
      cultivation alternative and site class">
      <classifiers>
         <classifier>
            <variable>MANAGEMENT_REGION</variable>
            <level>comp_unit</level>
         </classifier>
         <classifier>
            <variable>TAPIO_cultivation</variable>
            <level>comp_unit</level>
         </classifier>
         <classifier>
            <variable>SC</variable>
            <level>comp_unit</level>
         </classifier>
      </classifiers>
```

The last classifier for the table can optionally be of type *float* in which case the values for the target variable values are interpolated based on the classifier values for the last classifier:

```
...
      ...
                  <classifier>
                        <variable>TS</variable>
                        <level>comp_unit</level>
```

```
                    <variable_type>float</variable_type>
                </classifier>
```

Target is data when the result of parameter table is value of data variable. Here the model gives the value of *regen_dgm* attribute in `comp_unit` level as a result:

```
...
      <targets>
         <target>
            <data>
               <variable>regen_dgm</variable>
               <level>comp_unit</level>
            </data>
         </target>
      </targets>
```

Here in the table the regeneration diameter for the certain attribute combinations are given. First three digits are classifiers and the last is the regeneration diamter:

```
...
      <table>
         <value>1 1 1 28</value>
         <value>1 1 2 28</value>
         <value>1 1 3 28</value>
         <value>1 1 4 25</value>
         <value>1 1 5 25</value>
         <value>1 1 6 25</value>
         ...
      </table>
   </parameter_table>
   ...
</parameter_tables>
```

7.8.1 Different types of targets

operation_parameter

Target can also be a operation parameter (operation_parameter) when result of parameter table is used only as parameter of harvest or silvicultural operations. Here operation parameter *N* is given:

```
<target>
   <operation_parameter>
   <parameter>N</parameter>
   <length>1</length>
   </operation_parameter>
</target>
```

7.9 Geo table

The geo table document defines the location related variables used.

An example of the document contents is given below, for a comprehensive description of all the possible content options see the schema document. Data continuation and abbreviated content expressed as . . . :

The root level tag contains a reference to the schema document which is used to validate the content of the XML document:

```
<geo_tables xmlns="http://www.simo-project.org/simo"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.simo-project.org/simo ../schemas/geo_table.xsd">
```

A basic geo table named “dem”:

```
...
   <geo_table name="dem" desc="Digital elevation model">
      <filename>dem.latlonarray</filename>
      <ulx>20.0041666667</ulx>
      <uly>70.0958333333</uly>
      <nrow>1245</nrow>
      <ncol>1394</ncol>
      <xdim>0.00833333333333</xdim>
      <ydim>0.00833333333333</ydim>
      <nvar>1</nvar>
```

The coordinate attributes *GEO_X* and *GEO_Y* from `comp_unit` level are used as input of the geo table model:

```
...
      <coordinate_variables>
         <longitude>
            <variable>GEO_X</variable>
            <level>comp_unit</level>
         </longitude>
         <latitude>
            <variable>GEO_Y</variable>
            <level>comp_unit</level>
         </latitude>
      </coordinate_variables>
```

The result of the model is the value of *ALT* attribute in `comp_unit` level:

```
...
      <targets>
         <target>
            <variable>ALT</variable>
            <level>comp_unit</level>
         </target>
      </targets>
   </geo_table>
...
</geo_tables>
```

7.10 Cash flow

The cash flow document defines the costs and incomes of harvest and sivicultural operations used.

An example of the document contents is given below, for a comprehensive description of all the possible content options see the schema document. Data continuation and abbreviated content expressed as . . . :

The root level tag contains a reference to the schema document which is used to validate the content of the XML document:

```
<cash_flow_tables xmlns="http://www.simo-project.org/simo"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.simo-project.org/simo ../schemas/cash_flow_table.xsd">
```

In the cash flow tables a positive value refers income and negative cost. A Cash flow table named “timber_prices” is an example of table with classifiers:

```
...
   <cash_flow_table name="timber_prices"
                    logged="true"
                    desc="Timber prices in final cutting by trees species,
                          assortment, length class and diameter class">
      <base>
```

The *logged*-attribute controls whether the cash flow values from the table are stored in the database (in case price tracking is on).

The classifiers for the cash flow table are used to give a different value for different variable combinations. A classifier can be a data variable like the *PRICE_REGION* attribute from `comp_unit` level and *SP* attribute from `stratum` level:

```
...
         <classifiers>
            <classifier to_db="true">
               <variable>PRICE_REGION</variable>
               <level>comp_unit</level>
            </classifier>
            <classifier to_db="true">
               <variable>SP</variable>
               <level>stratum</level>
            </classifier>
```

If logging is set on, *to_db*-attribute for each classifier controls whether the classifier value is stored in the database together with the actual cash flow value.

A variable within a operation model can also be used as a classifier. Here the *assortment*, the *diameter* and the *length* attributes are used. The *assortment* attribute can get two different values: *1* or *2*. *1* refers to *log* and *2 pulp* wood:

```
...
            <classifier to_db="true">
               <variable>assortment</variable>
               <level>within-operation</level>
               <labels>
                  <label>
                     <value>1</value>
                     <name>log</name>
                  </label>
                  <label>
                     <value>2</value>
                     <name>pulp</name>
                  </label>
               </labels>
            </classifier>
            <classifier>
               <variable>diameter</variable>
               <level>within-operation</level>
            </classifier>
            <classifier>
               <variable>length</variable>
               <level>within-operation</level>
            </classifier>
         </classifiers>
```

Here in the table the timber prices for the certain attribute combinations are given. First five digits are classifiers and the last is the timber price:

```
...
         <table>
            <cash_flow>1 1 1 160 370 55</cash_flow>
            <cash_flow>1 1 1 160 400 55</cash_flow>
            <cash_flow>1 1 2 70 270 15</cash_flow>
            ...
         </table>
      </base>
   </cash_flow_table>
```

A very basic cash flow table named “scarification” is used as a example of table without classifiers. In the table there is only one digit given. It is the cost of certain soil preparation, scarification:

```
...
   <cash_flow_table name="scarification" desc="Cost of scarification">
      <cash_flow_variable>
         <name/>
         <level/>
      </cash_flow_variable>
      <base>
         <table>
            <cash_flow>-250</cash_flow>
         </table>
      </base>
   </cash_flow_table>
   ...
</cash_flow_tables>
```

7.10.1 Dated tables

It's possible for the base to consist of several tables. In this case the valid time period for each table can be given as attributes for the <table> element:

```
<base>
   <tables>
      <table start="2007-01-01" end="2011-12-31">
         <cash_flow>...</cash_flow>
      </table>
      <table start="2012-01-01" end="2016-12-31">
         <cash_flow>...</cash_flow>
      </table</cash_flow>
      </table>
   </tables>
</base>
```

7.10.2 Trends

Trend element is used for defining a trend over time for cash flow, eg. increase in the average price level over time:

```
<trends>
   <trend>
```

Time period for a trend is given in <start> and <end> elements as xs:date values:

```
...
      <time_period>
         <start>2007-01-01</start>
         <end>2011-12-31</end>
      </time_period>
```

Trends can be classified like the cash flow table values, so that different attribute combinations get different trends:

```
...
      <classifiers>
         <classifier>
            <variable>SP</variable>
            <level>stratum</level>
         </classifier>
         <classifier>
            <variable>assortment</variable>
            <level>within-operation</level>
            <labels>
               <label>
                  <value>1</value>
                  <name>log</name>
```

```
            </label>
            <label>
               <value>2</value>
               <name>pulp</name>
            </label>
         </labels>
      </classifier>
   </classifiers>
```

Cumulative factors for the trends are given in similar way as the cash flow table values. The value for variable y_t at year t is calculated by interpolating between years t_start and t_end, using values y_start and y_start + y_start * trend:

```
...
   <cumulative_factors>
      <factor>1 1 0.05</factor>
      <factor>1 2 -0.02</factor>
      <factor>2 1 0.12</factor>
      <factor>2 2 0.03</factor>
   </cumulative_factors>
</trend>
</trends>
```

7.11 Cash flow model

The cash flow modelbase defines cash flow models that affect different prices and costs over time.

An example of the document contents is given below, for a comprehensive description of all the possible content options see the schema document. Data continuation and abbreviated content expressed as . . . :

The root level tag contains a reference to the schema document which is used to validate the content of the XML document:

```
<cash_flow_modelbase xmlns="http://www.simo-project.org/simo"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.simo-project.org/simo
   ../schemas/cash_flow_modelbase.xsd">
```

Each cash flow model has a unique name and an implementation. The implementation is defined by giving the cash flow model library module, where the actual programmatic implementation is, as well as the programming language for the implementation (at the moment only Python is supported):

```
...
   <cash_flow_model>
      <name>Timber_prices</name>
      <implemented_at>cashflowmodels.py</implemented_at>
      <implemented_in>Python</implemented_in>
```

Each model gets it's reference cash flow values from a cash flow table (*Cash flow*):

```
...
      <reference_values>
         <cash_flow_table>timber_prices</cash_flow_table>
      </reference_values>
```

Cash flow models can also have input variables, which are taken from current data variables:

```
...
      <variables>
         <variable>
            <name>SP</name>
            <level>stratum</level>
         </variable>
      </variables>
```

```
      </cash_flow_model>
   </cash_flow_modelbase>
```

7.11.1 Optional content

In addition to the cash flow table, a date can be specified for the reference values. In that case the cash flow table for the reference values must have the start and end dates specified for each of the value tables it contains (:ref: *cash-flow-table-xml*). The table that overlaps the date specified for the reference values is then used to get the reference values:

```
<reference_values>
   <cash_flow_table>timber_prices</cash_flow_table>
   <date>2010-01-01</date>
</reference_values>
```

7.12 Data conversion

The data conversion XML document is used to convert the input data to match the data model defined in the lexicon document (*Lexicon*).

An example of the inlined and by-level input data formats are given below, for a comprehensive description of all the possible content options for the document see the schema document. Data continuation and abbreviated content expressed as ...

7.12.1 Inlined example

The root level tag contains a reference to the schema document which is used to validate the content of the XML document:

```
<SIMO_conversion_mapping xmlns="http://www.simo-project.org/simo"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.simo-project.org/simo
      ../schemas/conversion_mapping.xsd">
```

Data levels definitions are given nested as in the lexicon. The input data is given Each data row has an belongs to a certain data level implicated by a variable in a given position in the row

The input data has two data levels: `comp_unit` and `stratum`, `comp_unit` being the top level, simulation unit (e.g. stand or sample plot) and `stratum` stratum being the lower level (sub_levels). In the input data the `comp_unit` data is on the rows that have value 1 (rowtype_value) at the position 1 (rowtype_rowpos), and the `stratum` data is on the rows where this value is 2. The id for each object is at the position 1 for `comp_unit` objects and at position 2 for `stratum` objects. The position values begin from 0:

```
...
   <data_levels>
      <level>
         <name>comp_unit</name>
         <id_rowpos>
            <pos>0</pos>
         </id_rowpos>
         <rowtype_rowpos>1</rowtype_rowpos>
         <rowtype_value>1</rowtype_value>
         <date_rowpos>7</date_rowpos>
         <sublevel>
            <name>stratum</name>
            <id_rowpos>
               <pos>2</pos>
            </id_rowpos>
```

```
            <rowtype_rowpos>1</rowtype_rowpos>
            <rowtype_value>2</rowtype_value>
         </sublevel>
      </level>
   </data_levels>
```

Note that the <rowtype_value> tag can contain several values. If several rowtype values are specified, the smallest is taken to indicate the actual data row, others are treated as containing extra information. For example, the date information should be on the main data row.

Missing values in the input data are indicated by either with no data between the delimiters (‘’) or with the value -1 (none_value_indicator). In this case the input data row is rejected during data import if the SIMO variable MAIN_GROUP would get any of the values 4, 5, 6, 7 or 8 (object_rejection). The rejection variable must be from the highest data level; i.e., in this example from the comp_unit level:

```
...
   <none_value_indicator>'' -1</none_value_indicator>
   <object_rejection>
      <SIMO_variable>
         <name>MAIN_GROUP</name>
         <reject_criterion oper="in">
            <enum>4 5 6 7 8</enum>
         </reject_criterion>
      </SIMO_variable>
   </object_rejection>
```

The variable “Pinta-ala” is converted into SIMO variable “AREA” during import. The “Pinta-ala” value in the from-element is only for documenting purposes. It won’t have any effect in the import, because the actual imported value is defined by the row_type and row_position element values. Here the row_type value refers to the ones defined above in the data_levels definitions. It would be possible to give a default value for this variable in case of a missing value (none_to_value). A conversion factor is defined for numerical attributes (conversion_factor). It’s used in data import by multiplying the input data value with it:

```
...
   <variable>
      <name>
         <from>Pinta-ala</from>
         <to>AREA</to>
      </name>
      <row_type>1</row_type>
      <row_position>8</row_position>
      <from_datatype>double</from_datatype>
      <none_to_value/>
      <numerical>
         <conversion_factor>1</conversion_factor>
      </numerical>
   </variable>
```

For categorical values explicit mapping from input values to SIMO attribute values is given (value_mapping). In this case input data value 1 at position 14 for row type 1 is converted to PEAT attribute value 0, and values 2, 3, 4 and 5 are converted to 1:

```
...
   <variable>
      <name>
         <from>Alaryhmä</from>
         <to>PEAT</to>
      </name>
      <row_type>1</row_type>
      <row_position>14</row_position>
      <from_datatype>int</from_datatype>
      <none_to_value/>
      <categorical>
         <value_mapping>
```

```
            <value>
               <from>1</from>
               <to>0</to>
            </value>
            <value>
               <from>2 3 4 5</from>
               <to>1</to>
            </value>
         </value_mapping>
      </categorical>
   </variable>
   ...
```

Besides *numerical* and *categorigal* values, *dates* and *text* can be imported as well. The *epoch_year* for date is either *current* or

```
   ...
      <variable>
         <name> <from></from> <to>inventory_date</to>
         </name> <row_type>1</row_type> <row_position>15</row_position>
         <from_datatype>date</from_datatype> <none_to_value/> <date>
            <epoch_year>current</epoch_year>
         </date>
      </variable> ... <variable>
         <name> <from></from> <to>estate_name</to>
         </name> <row_type>1</row_type> <row_position>16</row_position>
         <from_datatype>string</from_datatype> <none_to_value/> <text/>
      </variable>
   </SIMO_conversion_mapping>
```

7.12.2 By-level example

The conversion format for data in separate files for each data level is similar to the inlined data except for the data level definitions:

```
<data_levels>
   <level>
      <name>comp_unit</name>
      <id_rowpos>
         <pos>1</pos>
      </id_rowpos>
      <link_id_rowpos/>
      <rowtype_value>1</rowtype_value>
      <sublevel>
         <name>stratum</name>
         <id_rowpos>
            <pos>1</pos>
         </id_rowpos>
         <link_id_rowpos>
            <link>
               <level>comp_unit</level>
               <pos>0</pos>
            </link>
         </link_id_rowpos>
         <rowtype_value>2</rowtype_value>
```

```
      </sublevel>
   </level>
</data_levels>
```

Here the linking between the files is defined in the link_id_rowpos-element for the `stratum` data: the first value (pos) in each row in the `stratum` data file will contain an id for the `comp_unit` (level) object the `stratum` belongs to. This id will be identical to the value found in the second position (id_rowpos is 1) in the rows of the `comp_unit` data file.

7.13 Forced operation

There are two alternatives for importing predetermined forestry operation information to the simulation: text files in the SMU file format used in the MELA planning software (see *Operation conversion*), and XML documents in the format specified here.

In the case the planner has decided that a certain forestry operation will be simulated for a certain simulation unit, only that operation will be simulated for that unit and the forestry operation model chains used in the simulation normally will be bypassed.

This data file is accompanied by an operation mapping document (*Operation mapping*) which will tie the operation names used in this document to model chains used in the simulation that will actually implement the desired forced operations.

An example of the document contents is given below, for a comprehensive description of all the possible content options see the schema document. Data continuation and abbreviated content expressed as ...:

The root level tag contains a reference to the schema document which is used to validate the content of the XML document:

```
<forced_operations xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:schemaLocation="http://www.simo-project.org/simo
            ../schemas/forced_operation.xsd">
```

Forced operations are defined for each simulation unit by specifying the id (id) and data level (level) of the object. A set of operations (operations) may be defined. Each operation has a specific target in the data level hierarchy, here the object level is used. Each operation (name) in SIMO belongs to a forestry operation group (group):

```
...
   <simulation_unit>
      <id>6</id>
      <level>comp_unit</level>
      <operations>
         <operation>
            <target>
               <level>comp_unit</level>
            </target>
            <name>low_thinning</name>
            <group>thinning</group>
            <date>2016-01-01</date>
            <parameters/>
         </operation>
      </operations>
   </simulation_unit>
   ...
</forced_operations>
```

7.14 Operation mapping

Operation mapping document contains the information how each of the forestry operations used in the forced operation data files (*Operation conversion* or *Forced operation*) is implemented in the simulation.

An example of the document contents is given below, for a comprehensive description of all the possible content options see the schema document. Data continuation and abbreviated content expressed as . . . :

The root level tag contains a reference to the schema document which is used to validate the content of the XML document:

```
<operation_mapping xmlns="http://www.simo-project.org/simo"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.simo-project.org/simo ../schemas/operation_mapping.xsd">
```

Here the clearcut operation is implemented with two model chain (*Model chain*) documents:

```
...
   <operation>
      <from>
         <name>clearcut</name>
      </from>
      <to>
         <model_chain>Forced clearcut</model_chain>
         <model_chain>Update comp_unit after clearcut</model_chain>
      </to>
   </operation>
   ...
</operation_mapping>
```

7.15 Operation conversion

In the case the planner has decided that a certain forestry operation will be simulated for a certain simulation unit, only that operation will be simulated for that unit and the forestry operation model chains used in the simulation normally will be bypassed.

There are two alternatives for importing predetermined forestry operation information to the simulation: text files in the SMU file format used in the MELA planning software (see *Operation conversion*), and XML documents (see *Forced operation*).

Both formats are accompanied by an operation mapping document (*Operation mapping*) which will tie the operation names used to model chains used in the simulation that will actually implement the desired forced operations.

The SMU format files are accompanied by an XML conversion document illustrated here.

An example of the document contents is given below, for a comprehensive description of all the possible content options see the schema document. Data continuation and abbreviated content expressed as . . . :

The root level tag contains a reference to the schema document which is used to validate the content of the XML document:

```
<operation_conversion xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                      xsi:schemaLocation="http://www.simo-project.org/simo
                      ../schemas/operation_conversion.xsd">
```

The beginning of the conversion definition is mostly similar to the data conversion definition (*Data conversion*): the data level at which the operations are performed (operation_level), value indicating a missing value (none_value_indicator), position of the simulation unit id (id_row_position), and definitions of different types of operations in the data file (operation_type):

```
?
   <operation_level>comp_unit</operation_level>
   <none_value_indicator>0</none_value_indicator>
   <id_row_position>0</id_row_position>
   <operation_type>
      <row_position>2</row_position>
      <mapping>
         <type>
```

```
            <from>1</from>
            <to>harvest</to>
         </type>
         <type>
            <from>2</from>
            <to>silviculture</to>
         </type>
      </mapping>
   </operation_type>
```

Next the actual operation mappings are defined, how the values in the data file are converted to operation names in SIMO. The parameter values associated with an operation are optional. If parameters are defined in operation mapping, the parameter values are added into simulation data just before the actual forced operation is done:

```
...
   <operation_name>
      <row_position>3</row_position>
      <no_operation_value>99</no_operation_value>
      <mapping>
         <operation>
            <type>2</type>
            <from>101</from>
            <to>natural_regeneration</to>
            <parameters>
               <parameter>
                  <name>REGEN_SP</name>
                  <level>comp_unit</level>
                  <value>1</value>
               </parameter>
            </parameters>
         </operation>
         ...
      </mapping>
   </operation_name>
```

The conversion definition ends with the information how the operations are timed, here the value 1 is converted to the first year of the simulation and the value 2 to the fifth year of the simulation:

```
...
   <operation_timing>
      <row_position>5</row_position>
      <time_step>
         <unit>year</unit>
         <mapping>
            <step>
               <from>1</from>
               <to>1</to>
            </step>
            <step>
               <from>2</from>
               <to>5</to>
            </step>
         </mapping>
      </time_step>
   </operation_timing>
</operation_conversion>
```

The timing could also be specified as ‘date’ in which case the row position must contain the date for the operation either in the ISO date format (2009-05-13) or in dd.mm.yyyy format. The element content can be left empty as it won’t be processed:

```
...
   <operation_timing>
      <row_position>5</row_position>
      <date/>
```

```
   </operation_timing>
</operation_conversion>
```

Operation data rows can also include values that affect simulation data. These values are mapped to simulation data using 'variables' mapping. The variables are similar to normal data variables, but their values are added to simulation data only just before the actual forced operation is done, just like operation parameter values:

```
...
   <variable_mapping>
     <variable>
             <name>
                     <from>percentOfArea</from>
                     <to>OPER_AREA_PROP</to>
             </name>
             <level>comp_unit</level>
             <row_position>5</row_position>
             <from_datatype>double</from_datatype>
             <none_to_value/>
             <numerical>
                     <conversion_factor>0.01</conversion_factor>
             </numerical>
     </variable>
   </variable_mapping>
```

7.16 Simulation

The simulation XML document is used to drive the simulation. It contains the simulation span definitions, the model chains (*Model chain*) used in each span, and optionally the alternative stopping logic for the simulation. In addition to those parameter values and initial attribute values are set using the simulation document.

An example of the document contents is given below, for a comprehensive description of all the possible content options see the schema document. Data continuation and abbreviated content expressed as ...:

The root level tag contains a reference to the schema document which is used to validate the content of the XML document:

```
<simulation xmlns="http://www.simo-project.org/simo"
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:schemaLocation="http://www.simo-project.org/simo
            ../xml/schemas/simulation.xsd" desc="forest plan">
```

The definition begins by fixing the simulation data level; i.e., object at the given <main_level> form the simulation units for the simulation. The data levels are defined in the lexicon (*Lexicon*). There are a few built-in attributes: 'time_step' and 'year', which are based on time_span definitions and are automatically updated by the simulator. Attribute 'time_step' is on simulation-level and 'year' on simulation main-level:

```
...
   <main_level>comp_unit</main_level>
   <built_in_attributes>
       <time_step>time_step</time_step>
       <year>year</year>
   </built_in_attributes>
```

The control part defines the start and end of the growth season end date (here 1st june and 1st August each year). If the data date for a simulation unit is past that date during the first simulation year, simulation chains are not executed for the unit. There can be several spans for a simulation. Each has its own time definition: time step for simulation chains (time_step) and forestry operations (operation_step) and how many time steps the span consists of (steps):

```
...
   <control>
      <growth_season_start_date>--06-01</growth_season_start_date>
```

```
      <growth_season_end_date>--08-01</growth_season_end_date>
      <span>
         <time>
            <time_step>2</time_step>
            <operation_step>2</operation_step>
            <time_unit>year</time_unit>
            <steps>1</steps>
         </time>
```

NB! the <time_unit> (year or month) must match the time dependent attribute handling in your modelchains, e.g. if you're assuming in the modelchain that the age of the trees is updated with years, you shouldn't use month-based spans in the simulation control.

Some of the data in a simulation is interesting, but some of it is not. Thus, not all data need to be stored in the result database. The saving of data is controlled with '<save_steps_to_db>' element. The element can have one of three enumerated values: 'all', 'none' or 'last'. These values refer to time steps and which time steps should be saved in the db. Value 'all' saves all time steps of the current time span. Value 'none' means that no data at all is stored from the current time span. Value 'last' means that only the last time step will be saved in the db from the current time span:

```
...
         <save_steps_to_db>all</save_steps_to_db>
```

Note that *branching_graph* output does not work quite right if '<save_steps_to_db>' was 'last' or 'none' in your simulations. This is because in the case of 'last' and 'none' the result database does not include date information. So, if you want to output branching graphs, make sure that '<save_steps_to_db>' is set to 'all'.

There are four different types of model chains for the span, each being optional. init_chains are executed only once at the beginning of the span. simulation_chains are executed at every step (except possibly for the first step, see above). forced_operation_chains contain the chains used to implement the predetermined forestry operations (note that these chains are automatically augmented with the chains defined in the operation mapping document (*Operation mapping*)). operation_chains contain the operations that the simulator will assign to the simulation units automatically in case there are no forced operations. If there are, operation_chains will not be executed. Model chain names are the XML document names, but without the xml-extension:

```
...
         <init_chains>
            <chain>Chain_complete_data</chain>
            <chain>Chain_calculate_missing_attributes</chain>
            <chain>Chain_generate_trees</chain>
            <chain>Chain_calculate_basic_attributes</chain>
         </init_chains>
         <simulation_chains>
            <chain>Chain_regeneration</chain>
            <chain>Chain_grow_seedling_stratum</chain>
            <chain>Chain_grow_little_tree</chain>
            <chain>Chain_grow_big_tree</chain>
            <chain>Chain_update_basic_attributes</chain>
            <chain>Chain_calculate_stand_value_growth</chain>
            <chain>Chain_calculate_stand_productive_value</chain>
         </simulation_chains>
         <forced_operation_chains>
            <chain>Chain_forced_operations</chain>
         </forced_operation_chains>
         <operation_chains>
            <chain>Chain_harvests_branching</chain>
         </operation_chains>
      </span>
      ...
   </control>
```

Simulation can be set to stop when given conditions are met. These stopping conditions can be set with '<stop_logic>' element. The syntax of the stopping conditions is the same as in model chain condition elements. Here's an example how simulation is set to stop after the first regeneration harvest, either clearcut or seedtree

position:

```
...
   <stop_logic>
      comp_unit:clearcut times_gt 0 or
      comp_unit:seedtree_position times_gt 0
   </stop_logic>
```

Initial attribute values, values set for each unit at the beginning of simulation, are set by defining the values at init_variables section. For init_variables the data level is specified in addition to the name and value of the attribute:

```
...
   <init_variables>
      <variable>
         <name>passed_thinning_limit</name>
         <level>comp_unit</level>
         <value>0</value>
      </variable>
      ...
   </init_variables>
```

The last setting in the simulation document applies to the attribute values that are stored in the simulation result database. If the `active`-attribute for the `output_constraints`-element is set to *"off"*, every attribute from every data level is stored in the result database. Since this is usually not what is desired (there are usually a lot of attributes. . .), the ones wanted can be speciefied for each data level:

```
...
   <output_constraints active="on">
      <level>
         <name>comp_unit</name>
         <variables>
            <variable>AREA</variable>
            <variable>TS</variable>
            ...
         </variables>
      </level>
      <level>
         <name>stratum</name>
         <variables>
            <variable>BA</variable>
            <variable>N</variable>
            ...
         </variables>
      </level>
   </output_constraints>
</simulation>
```

7.17 Optimization

The optimization document defines objective function used in optimization task.

An example of the document contents is given below, for a comprehensive description of all the possible content options see the schema document. Data continuation and abbreviated content expressed as . . . :

The root level tag contains a reference to the schema document which is used to validate the content of the XML document:

```
<optimization_task xmlns="http://latitude.mmvar.helsinki.fi/simo"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://www.simo-project.org/simo
         ../schemas/optimization_task.xsd">
```

An example of objective function:

The initial date is either a specific date or "today" and the time step applied will be yearly; i.e., later in the objective function definition the period 1 will refer to dates from 1.1.2007 to 31.12.2007 and the period 2 to dates from 1.1.2008 to 31.12.2008. Thus, it should be noted that the period definitions used in simulation are distinct from the period definitions in the optimization task definition. Let's say that the initial date for simulation is 1.1.2007 and it consists of periods 1 year + 1 year + 1 year + 3 years + 3 years. In that case the simulation period 5 covers dates between 1.1.2011 and 31.12.2013, whereas the period 5 in optimization covers the dates between 1.1.2011 and 31.12.2011 and the period 6 the dates between 1.1.2012 and 31.12.2012 etc.

The data filter <data_filter> is an optional element for targeting only a subset of the whole data in the optimization. The syntax of the data filter is similar to constraint conditions, or to model chain condition and only units that pass the filter will be included in the optimization.

The discount rate is given as a percentage; i.e. 3.5 % as 3.5:

```
...
   <initial_date>2007-01-01</initial_date>
   <period_step>year</period_step>
   <data_filter>comp_unit:SC eq 3</data_filter>
   <discount_rate>3.5</discount_rate>
   <objective_function>
```

If the optimization requires varying discount rates, for example for different geographical regions in the data, conditional discount rates can be defined:

```
...
   <initial_date>2007-01-01</initial_date>
   <period_step>year</period_step>
   <discount_rate>3.5</discount_rate>
   <conditional_discount>
      <rate>
         <condition>comp_unit:SC le 2</condition>
         <value>4.0</value>
      </rate>
      <rate>
         <condition>comp_unit:SC gt 4</condition>
         <value>2.0</value>
      </rate>
   </conditional_discount>
   <objective_function>
```

Here maximization with the *additive* function in *global* scope is used; i.e., the sub objective values are added together in search for a global maximum value. Should the scope be *local*, the maximum value for each simulation unit would be searched for:

```
...
      <type>
         <function>additive</function>
         <target>max</target>
         <scope>global</scope>
      </type>
```

The objective is to maximize the *Income* during all periods. In sum[1:-1](operation:Income:discount), 1 indicates the first period and -1 the last period. 'operation' in 'operation:Income' indicates that the Income variable is an operation result variable. ':discount' indicates that the values of the 'operation:Income' variable will be discounted with the discount rate defined above. If there are several sub objectives, they can be given different weights (<weight>):

```
...
     <sub_objectives>
         <sub_objective>
             <weight>1.0</weight>
             <expression>sum[1:-1](operation:Income:discount)</expression>
             <utility_function>
                 <type>linear</type>
```

```
            </utility_function>
         </sub_objective>
      </sub_objectives>
```

The constraint that the objective function must satisfy is that the discounted 'Income' is greater during the last ten years period than during the first ten years period:

```
...
      <constraints>
         <constraint>
         sum[11:20](operation:Income:discount) gt
         sum[1:10](operation:Income:discount)
         </constraint>
         ...
      </constraints>
   </objective_function>
</optimization_task>
```

The variables in subobjective and constraint expressions can be conditional. Optional conditions for variables are defined after the variable definition. In the following example, the '[operation:SP eq 1 and operation:assortment eq pulp]' defines that only the volume of pine (operation:SP eq 1) pulp wood is taken into account. The classifiers and the classifier values (e.g. assortment having values 'log' and 'pulp') for operation results are defined in the cash flow tables used for each operation:

```
...
     <constraint>
        sum[1:-1](operation:Volume[operation:SP eq 1 and operation:assortment eq pulp])
     </constraint>
```

Operation results can be conditional also to operation name and operation group, as in the two following constraints:

```
...
     <constraint>
        sum[1:-1](operation:Volume[operation:op_name eq thinning]) gt 10500
     </constraint>
     <constraint>
        sum[1:-1](operation:Volume[operation:op_group eq
                         final_harvest_artificial_regeneration]) gt 2000
     </constraint>
```

Data values can also be conditional, as in the following example. In it only the volume of those units from the comp_unit data level that have the MAIN_GROUP attribute value of 4 or greater are added together. In the example the V variable is defined to contain the volume per hectare and therefore the V value is multiplied by the area:

```
...
     <constraint>
        sum[-1:-1](comp_unit:AREA * comp_unit:V[comp_unit:MAIN_GROUP ge 4]) ge 10000
     </constraint>
```

NOTE: When using JLP optimization tools, the format for the constraints is rather limited. JLP constraints should: _not_ have division operators ('/'), have a numerical value on the "right hand" side. A valid JLP constraint would be for example:

```
...
     <constraint>
        sum[6:10](operation:cash_flow:discount) - sum[1:5](operation:cash_flow:discount) gt 0
     </constraint>
```

7.17.1 Different types of objective functions

The objective function type can be 'additive', 'conjunctive' or 'distance', and the value of the objective function is either maximized (<target>max</target>) or minimized (<target>min</target>). For additive objective function

the sub objective values are added together, for conjunctive they are multiplied with each other. The distance based objective function aims to maximize or minimize the distance to the optimal level given for each sub objective:

```
<objective_function>
   <type>
      <function>distance</function>
      <target>max</target>
      <scope>global</scope>
   </type>
   <sub_objectives>
      <sub_objective>
          <weight>1.0</weight>
          <expression>sum[1:-1](operation:Income:discount)</expression>
          <utility_function>
              <type>linear</type>
          </utility_function>
          <optimal_level>100000</optimal_level>
      </sub_objective>
   </sub_objectives>
```

The utility function type given for each sub objective can be only *linear*. Optionally, a midpoint can be defined for the linear utility function defining a turning point for the piecewise linear function:

```
<utility_function>
    <type>linear</type>
    <midpoint>
       <x>0.5</x>
       <y>0.8</y>
    </midpoint>
</utility_function>
```

The midpoint x and y coordinates are defined in the [0..1] range, if no midpoint is given, the utility function grows linearly from 0 to 1. Defining a midpoint for the linear utility function affects how fast or slow the utility value increases as the utility variable value increases. If no midpoint is given, the utility value increases with the same ratio as the utility variable value. If the utility function is concave (y > x), utility value increases faster than the utility variable value on the left side of the midpoint (< x) and slower on the right side (> x). If the utility function is convex (x > y), utility value increases slower than the utility variable value on the left side of the midpoint (< x) and faster on the right side (> x).

7.17.2 Defining expressions for constraints and sub objectives

The expression begins with one of the aggregators 'sum', 'mean', 'min' or 'max'and is followed by the time frame for the aggregation inside square brackets in the form of start period:end period; e.g. sum[1:2]:

```
sum[1:2]
```

1 indicates the first period, and -1 is reserved for the last period. After the aggregator and period definition, the variable for the expression is given inside parenthesis in the form of data level name:variable name:

```
sum[1:2](stratum:BA)
```

In addition to the attribute data (data levels and attributes defined in the *Lexicon*), the data values can also come from the operation results. After the operation variable definition an optional discounting definition follows:

```
sum[1:2](operation:Income:discount)
```

The operation result names are defined in the operation model xml definitions (*Operation model*)

Conditions for each data variable can also be defined giving the condition after the variable definition in square brackets:

```
sum[1:2](operation:Income[operation:SP eq 1]:discount)
```

NOTE! Operation result conditions can be both operation results, or from simulated data. For simulated data, the conditions can be only from simulated, and the same data level. Here's two examples from which the first one is valid and the second one invalid.

Valid:

```
sum[1:2](operation:Income[operation:SP eq 1 and comp_unit:SC eq 3])
```

Invalid:

```
sum[1:2](comp_unit:V[operation:SP eq 1 and comp_unit:SC eq 3])
```

The latter one will cause an error when the simulator is built.

See above for different alternatives for the conditions. The allowed operators for conditions are 'gt', 'ge', 'eq', 'ue', 'le', 'lt' and 'in'. For the 'in'-operator the allowed values are given as a list after the operator:

```
sum[1:2](comp_unit:V[comp_unit:MAIN_SP in (3, 4, 5, 6, 7, 9)])
```

Several conditions can be given by connecting them with the operators 'and' and 'or':

```
sum[1:2](comp_unit:V[comp_unit:MAIN_SP in (3, 4, 5, 6, 7, 9) and DEVEL_CLASS eq 1])
```

The aggregation can also contain several variables combined with the operators '+', '-', '*' and '/':

```
sum[1:2](comp_unit:V * comp_unit:AREA)
```

Several aggregations can be combined in an expression using the operators '+', '-', '*' and '/':

```
sum[1:2](comp_unit:V[comp_unit:SP eq 1] * comp_unit:AREA) /
sum[1:2][comp_unit:V * comp_unit:AREA)
```

Naturally, also constants can be used instead of aggregations:

```
sum[1:2](comp_unit:V[comp_unit:SP eq 1] * comp_unit:AREA) - 10000
```

The for constraints the expression is finalized by defining the comparison operator, which is one of 'gt', 'ge', 'eq', 'ue', 'le' and 'lt':

```
sum[1:2](comp_unit:V * comp_unit:AREA) - 10000 gt 0
```

7.17.3 Constraint violations caused by LP relaxation

Splitting stands in JLP (LP relaxation) can cause constraint violations when SIMO selects the branches with max weights. In this case, SIMO can fall back back to heuristic optimizer that finds a real integer solution that conforms to the constraints.

For each constraint, user can define the maximum violation as an absolute value or percentage. The syntax for this is:

```
<constraint max_violation="10" violation_metric="percentage">sum[1:5](operation:Volume) lt 5000.0
```

To activate these limits on constraint violations, .ini file needs to set optimization.jlp.do_fallback as True

It is a good idea to set the 'keep_opt_data' and 'reuse_opt_data' as true, as that will decrease the optimization times for big data sets considerably.

7.18 Output constraint

The output costraint document defines the levels and variables used in report.

An example of the document contents is given below, for a comprehensive description of all the possible content options see the schema document. Data continuation and abbreviated content expressed as . . . :

The root level tag contains a reference to the schema document which is used to validate the content of the XML document:

```
<output_constraints xmlns="http://latitude.mmvar.helsinki.fi/simo"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.simo-project.org/simo ../schemas/Output_constraint.xsd">
```

Output constraint includes the repoting data levels and variables. Here from level `comp_unit` and `stratum` will be reported attributes *SC MAIN_SP Age* and *SP Age BA*, respectively:

```
...
   <data_level>
       <name>comp_unit</name>
       <var_list>SC MAIN_SP Age</var_list>
   </data_level>
   <data_level>
      <name>stratum</name>
      <var_list>SP Age BA</var_list>
   </data_level>
   ...
</output_constraints>
```

7.19 Aggregation definition

The aggregation definition document defines reporting periods, variables and output types in aggregation output.

An example of the document contents is given below, for a comprehensive description of all the possible content options see the schema document. Data continuation and abbreviated content expressed as ...:

The root level tag contains a reference to the schema document which is used to validate the content of the XML document:

```
<report_definition xmlns="http://latitude.mmvar.helsinki.fi/simo"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://www.simo-project.org/simo
         ../schemas/report_definition.xsd">
```

First, you must define the initial date of the simulation (as YYYY-MM-DD) to get meaningful results:

```
...
   <initial_date>2009-01-01</initial_date>
```

Next, discount rate is defined for outputs that needed discounting, such as net present values of income from harvests etc.:

```
...
   <discount_rate>3.0</discount_rate>
```

Reporting periods can contain a number of periods of given length. Period units can be one of: *year, month, day*:

```
...
   <reporting_periods>
       <period>
           <length>5</length>
           <unit>year</unit>
       </period>
       <period>
           <length>137</length>
           <unit>month</unit>
       </period>
   </reporting_periods>
```

Single report definition can include multiple reports. The syntax for defining the reported variables is similar to optimization task subobjective and constraint syntax (see: *Defining expressions for constraints and sub objectives*).

Below is an example of a very simple report, where harvest volumes from years 1..3 are reported. Note that period counting begins from 1, and the last period has a special indicator -1. You can of course indicate the last period with its direct ordinal number; i.e., for a 30 year simulation the last period is 30:

```
...
   <reports>
      <report>
         <expression>sum[1:3](operation:Volume)</expression>
         <proportion>false</proportion>
      </report>
```

Reported values can be also grouped by given variable values, such as in:

```
...
      <report>
         <expression>sum[1:7](operation:Volume)</expression>
         <proportion>false</proportion>
         <group_by>
            <grouping>
                <level>operation</level>
                <variable>SP</variable>
            </grouping>
         </group_by>
      </report>
```

Here is another example of result grouping (the results will be reported as precentage instead of values, since proportion is set to true).:

```
...
      <report>
         <expression>sum[1:-1](comp_unit:AREA)</expression>
         <proportion>true</proportion>
         <group_by>
            <grouping>
                <level>comp_unit</level>
                <variable>DEVEL_CLASS</variable>
            </grouping>
         </group_by>
      </report>
```

It's also easy to give conditions to certain values, though in that case, it should in most cases be given to all the values so that the expression doesn't fail:

```
...
      <item>
         <expression>avg[1:-1](comp_unit:Age[iteration eq 0] * comp_unit:AREA[iteration eq 0])</ex
         <proportion>true</proportion>
      </item>
```

NOTE! The conditions do not work at the moment as they should (and like in expression output and optimization task)!!!

Report definition has a small extension to optimization task syntax for calculating weighted averages:

```
...
      <report>
         <expression>wavg[1:-1](comp_unit:Age, comp_unit:AREA)</expression>
         <proportion>false</proportion>
      </report>
```

The aggregation expressions can be more complex as in optimization tasks:

```
...
      <report>
         <expression>
            sum[1:60](operation:cash_flow:discount) +
            sum[60:60](comp_unit:PV:discount * comp_unit:AREA) +
            sum[60:60](comp_unit:PV_land:discount * comp_unit:AREA)
```

```
        </expression>
      <proportion>false</proportion>
    </report>
```

7.20 Expression definition

The expression definition document defines reporting variables and output types in expression output.

An example of the document contents is given below, for a comprehensive description of all the possible content options see the schema document. Data continuation and abbreviated content expressed as . . . :

While expression definition is very close to aggregation definition, there are some key differences:

1. Expression definition does not support reporting periods, nor grouping.
2. It is always grouped by id, iteration and branch.
3. If a certain key combination has no value, it is dropped from the results.

The root level tag contains a reference to the schema document which is used to validate the content of the XML document:

```
<expression_definition xmlns="http://latitude.mmvar.helsinki.fi/simo"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://www.simo-project.org/simo
         ../schemas/expression_definition.xsd">
```

First, you must define the initial date of the simulation (as YYYY-MM-DD) to get meaningful results:

```
...
   <initial_date>2009-01-01</initial_date>
```

Next, discount rate is defined for outputs that needed discounting, such as net present values of income from harvests etc.:

```
...
   <discount_rate>3.0</discount_rate>
```

Single expression definition can include multiple expressions. The syntax for defining the reported variables is similar to optimization task subobjective and constraint syntax. Below is an example of a very simple report, where harvest volumes, between first year (1) and last year (-1) are reported:

```
...
   <expressions>
      <item>
         <expression>sum[1:-1](operation:Volume)</expression>
         <label>Harvest volume</label>
      </item>
```

It's also easy to give conditions to certain values, though in that case, it should in most cases be given to all the values so that the expression doesn't fail:

```
...
      <item>
         <expression>avg[1:-1](comp_unit:Age[comp_unit:MAIN_SP eq 1] * comp_unit:AREA[comp_unit:MA
         <label>Average age</label>
      </item>
```

The conditions must be given so that if the result variable is from operation results, the condition variables can be either operation results or simulated data. If the result variable is simulated data (i.e. from level 'comp_unit'), the condition variables can be only from the same level. Here's a valid and invalid example.

Valid:

```
...
      <item>
         <expression>sum[1:-1](operation:Volume[operation:assortment eq 1 and comp_unit:MAIN_SP e
         <label>Average age</label>
      </item>
```

Invalid:

```
...
      <item>
         <expression>sum[1:-1](comp_unit:V[operation:assortment eq 1 and comp_unit:MAIN_SP eq 1])
         <label>Average age</label>
      </item>
```

The expressions can be more complex as in optimization tasks (note that cash_flow, PV and PV_land are discounted in the below syntax):

```
...
      <item>
         <expression>
            sum[1:60](operation:cash_flow:discount) +
            sum[60:60](comp_unit:PV:discount * comp_unit:AREA) +
            sum[60:60](comp_unit:PV_land:discount * comp_unit:AREA)
         </expression>
         <label>NPV</label>
    </item>
```

7.21 Lexicon translation

The lexicon translation document defines translation of vocabulary used.

The lexicon translation has the same data levels as lexicon. The lexicon is constructed as a hierarchy of data levels, in which the root level is always `simulation`. Each data level contains the definitions for its attributes and possibly its sublevel.

An example of the document contents is given below, for a comprehensive description of all the possible content options see the schema document. Data continuation and abbreviated content expressed as ...:

The root level tag contains a reference to the schema document which is used to validate the content of the XML document:

```
<lexicon_translation_table xmlns="http://www.simo-project.org/simo"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.simo-project.org/simo ../schemas/lexicon_translation_table.xsd
```

The translation for `simulation` level. There is not any attributes in simulation level:

```
...
   <level>
      <name>
         <from>simulation</from>
         <to>
            <lang name="fi">simulointi</lang>
         </to>
      </name>
   </level>
```

The translation for `comp_unit` level and attributes on that level:

```
...
   <level>
      <name>
         <from>comp_unit</from>
         <to>
            <lang name="fi">kuvio</lang>
```

```
            </to>
         </name>
         <variables>
            <variable>
               <from>DEVEL_CLASS</from>
               <to>
                  <lang name="fi">kehitysluokka</lang>
               </to>
            </variable>
            ...
         <variables>
         ...
      </level>
</lexicon_translation_table>
```

7.22 Message translation

The message translation document defines translation of messages used.

An example of the document contents is given below, for a comprehensive description of all the possible content options see the schema document. Data continuation and abbreviated content expressed as . . . :

The root level tag contains a reference to the schema document which is used to validate the content of the XML document:

```
<message_translation_table
   xmlns="http://www.simo-project.org/simo"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.simo-project.org/simo ../schemas/message_translation_table.xsd"
```

A message translation is grouped by logging level and logger. Here the logging level is “debug” and the logger is “simulation”. Other alternatives for the logging level are “info”, “warning”, “error” and “exception” , and for the logger “data import”, “optimization”, “output” and “simo”. The message (message) includes original message and the translation of the message for certain language:

```
...
   <log_level name="debug">
      <logger name="simulation">
         <message>
            <from>Variable '(.+)' defined at level '(.+)' in lexicon
                  but not used in the simulation!
            </from>
               <to>
                  <lang name="fi">Muuttuja '$1' on määritelty sanastossa
                        tasolla '$2', mutta sitä ei käytetä simuloinnissa!
                  </lang>
               </to>
         </message>
      </logger>
   </log_level>
   ...
</message_translation_table>
```

CHAPTER

EIGHT

MODEL LIBRARIES

This chapter contains descriptions of the different types of model implementations used in SIMO. Each model type has also an XML definition, or interface, that is used for binding model inputs and outputs into the simulation framework.

8.1 Aggregation model library

An aggregation model takes one attribute as its input parameter. The parameter is an instance of the **AggregationArg** class (see simo.simulation.model.aggregationarg).

The object instance has the following attributes:

- *operands* – the number of operands for the model
- *target_variable* – index of the target variable in the data matrix
- *remove_targets* – indices of the objects for which the aggregation model is not evaluated
- *use_nan_funcs* – boolean indicating NaN values in the data
- *values* – operand values
- *weights* – weight attribute values
- *target_index* – indices of the object for which the model is evaluated
- *oper_target_index* – target index tables for operand(s)
- *oper_data_level* – operand data level indice, int
- *success* – boolean for successfull model evaluation
- *errors* – list of errors encountered during model evaluation
- *deterministic* – boolean that can be used to control whether random values are used in aggregation models or not (among others, random number generation is treated as an aggregation model)
- *results* – a numpy vector for storing aggregation results, NOTE: aggregation model implementation is responsible for making sure that this actually is a numpy array

Below is an example of a very simple **substract** aggregation model:

```
def substract(arg):
    """Substract two operands
    """
    if arg.operands != 2:
        arg.success = False
        arg.errors.append("Two operands required in 'substract'")
    else:
        arg.results = arg.values[0] - arg.values[1]
```

8.2 Cash flow model library

Besides cash flow tables, costs and incomes associated with operations or tree value models can also be defined with cash flow models; i.e., models predicting the unit price changes over time based on some variables.

All cash flow models have the same set of input parameters:

- *table* – cash flow table object (for details, see simo.builder.modelbase.cashflowtable.py)
- *date* – current date as datetime object
- *variables* – cash flow model input variables
- *parameters* – cash flow model parameters
- *errors* – container for returning errors from the model

Below is an example of a stochastic timber assortment price model:

```
def random_prices(table, date, variables, parameters, errors):
    ret = True
    index = (1,2)

    # the initial year is now fixed to 2008
    n = date.year - 2008 + 1

    # load the prices and compute standardized delta (increment), simulate
    # prices for the next n years
    prices, delta_st = _load_prices()
    P = _simulate(prices, delta_st, n, 2)

    # store the prices of the last year
    table.set_values(index, PINELOG, P[-1,0])
    table.set_values(index, SPRUCELOG, P[-1,1])
    table.set_values(index, BIRCHLOG1, P[-1,2])
    table.set_values(index, BIRCHLOG2, P[-1,2])
    table.set_values(index, PINEPULP, P[-1,3])
    table.set_values(index, SPRUCEPULP, P[-1,4])
    table.set_values(index, BIRCHPULP1, P[-1,5])
    table.set_values(index, BIRCHPULP2, P[-1,5])

    return ret
```

8.3 Geotable

Geotable models are used to get variable values by point location. The geotable models operate in a grid spanning the geodetic coordinate space (latitude, longitude).

For Finland there are models in the prediction model library to convert cartographic (northing, easting) point locations into geodetic coordinates.

Geotable inputs and outputs are defined in the XML interface (for example, see (*Geo table*)). The actual geotables can be constructed, for example, with the scripts found in http://svn.simo-project.org/tools/trunk/txt2latlon

8.4 Management Model Library

Management models are used for modifying SIMO data hierarchies, such as removing objects constructing new objects and grouping objects.

All management model functions should have the following parameters:

- *sim* – simulator instance (for details, see simo.simulation.sim.py)

- *depthind* – current model chain depth index as int
- *params* – management model parameters in a dictionary
- *tind* – target object index as a 2d numpy array (for details, see simo.matrix.handler.py)

Below is a simple example of a management model for removing each object from current simulation level. The model implementation can utilize the public object attributes and methods of simulator and data handler:

```
def remove_objects_from_current_level(sim, depthind, params, tind, toremove):
    """Remove objects from current evaluation level

    sim -- simulator instance
    depthind -- current model chain depth index as int
    params -- management model parameters
    tind -- target index
    """
    # get the level that all objects should be removed from
    level = sim.level
    for i in range(tind.shape[0]):
        it, br, o = tind[i,0:3]
        sim.data.del_objects(it, br, level, [o])
```

8.5 Operation model library

Operation models are the implementations of the various forestry operations in SIMO. The operation models can be divided into two classes: cost only operations, and operations that modify the simulation data. Cost only operations have only an associated cost, but do not modify the data; e.g. soil preparation model. Models that modify data have also an associated cash flow (cost or income), but besides that they also modify the data, for example by removing trees (e.g. thinning model) or adding trees (planting model)

Operation models can currently be implemented only in Python in SIMO due to the complex data structures that are passed between simulator and the model implementation. For details on these data structures, see *simo.simulation.caller.operationcaller.py* and *simo.simulation.model.operationarg.py*.

All SIMO operation models share the same set of input parameters. When implementing new operation model, remember to assign the following list of parameters as the function parameters:

- *data* – data by level in a dictionary where *key* = level and *value* = 2d numpy array
- *variables* – operation variable values in a 1d numpy array
- *parameters* – operation parameter values in a 1d numpy array
- *parameter_table_values* – values from parameter tables (simulator handles the table processing)
- *cash_flow_table* – cash_flow structure
- *cash_flow* – size 1 numpy array
- *results* – container (a numpy array for scalars or a list of date value tuples for date-arrays) for returning operation results
- *warnings* – container for returning warnings
- *errors* – container for returning errors

Below is an example of operation model function definition:

```
def clearcut(data, variables, parameters, parameter_table_values,
             cash_flow_table, cash_flow, results, warnings, errors):
    """Remove all trees except a number of reserve trees from a stand.
    """
```

8.5.1 Bucking algorithm implementation

The bucking algorithm used in SIMO operation models is based on Näsberg's (1985) dynamic optimization algorithm for optimal stem cutting. The stem cutting is optimized for maximum value with given price matrix for the various timber assortments. The bucking algorithm is implemented as a dynamically loaded C function.

The price matrix is a 2-dimensional array, where each row represents a single timber assortment class. The columns are: timber assortment indice, minimum diameter, minimum length and value (ie. price in euros). Below is a simple example of price matrix with four timber assortment classes. The assortment indices in the first column are 1=log and 2=pulp:

```
...
1.  160.  370.   55.
1.  160.  400.   57.
1.  160.  430.   59.
2.   70.  300.   17.
...
```

Tree stem profile is a 2-dimensional array where each row represents a single tree stem segment, cut with given divisor. The columns are segment top diameter, segment top height and cumulative volume, starting from the bottom segment. The tree stem and the stem segments are computed using taper curve functions by Laasasenaho (1982), separately for each tree species. Below is a simple example of a tree stem array:

```
...
235.  120.  0.002345
229.  150.  0.002891
222.  180.  0.003210
...
```

Below is a pseudo code description of the bucking algorithm (indexing arrays etc starts from 1):

```
INPUTS:
        div -- tree stem divisor (segment length), double
        n -- number of tree segments, integer
        m -- number of timber assortment price classes, integer
        P -- price matrix as a 2-dimensional array of size m x 4
        T -- tree stem profile as a 2-dimensional array of size n x 3

// Initialize arrays V for storing volumes and C for storing values
// (...and fill the arrays with zero values)
SET V to 1-dimensional double array of size n
SET C to 1-dimensional double array of size n

// Initialize arrays A for storing timber assortment indices and L for
// storing log bottom positions (...and fill the arrays with zero values)
SET A to 1-dimensional integer array of size n
SET L to 1-dimensional integer array of size n

// Initialize some auxiliary variables
SET t to 1
SET d_top to 0.0
SET d_min to 0.0
SET v to 0.0
SET c to 0.0
SET v_tot to 0.0
SET c_tot to 0.0

FOR i = 1 to n
        FOR j = 1 to m
                // Get a location indice t for the top segment of the current stem
                // piece when the location indice for the bottom segment is i
                // (divide the height with the stem divisor)
                t = i + P[j][3] / div
                IF t is less than n THEN
                        // Get the top diameter of the current tree segment and check
```

```
                        // if the diameter is greater than the minimum assortment
                        // diameter defined in the price matrix P
                        d_top = T[top][1]
                        d_min = P[j][1]
                        IF d_top is greater than or equal to d_min THEN
                                // Calculate current tree stem's (bottom at i, top at t)
                            // volume v and the cumulative value c
                                v = T[t][3] - T[i][3]
                                c = v * P[j][4]
                                // Calculate the total volume and value at location i
                                v_tot = v + V[i]
                                c_tot = c + C[i]
                                // If total value, calculated as the value of the current
                                // log plus all the logs below i, is higher than the
                                // existing value in t, overwrite the existing value
                                // with c_tot
                                IF c_tot > C[t] THEN
                                        V[t] = v_tot
                                        C[t] = c_tot
                                        A[t] = P[j][0]
                                        L[t] = i
        ENDFOR
ENDFOR

// Get the location indice for the maximum value in C[t] array by calling
// appropriate function
CALL get_max_indice with C RETURNING maxi

// Define arrays for storing timber assortment volumes and values
SET VOLUMES to 1-dimensional double array with size equal to number of assortments
SET VALUES to 1-dimensional double array with size equal to number of assortments

// define some auxiliary variables
SET a to 1
SET l to 1

// Cut the tree to optimal length logs and divide the values and
// volumes to corresponding assortments, start from the top
WHILE maxi is greater than 0
        a = A[maxi]
        l = L[maxi]
        VOLUMES[a] = VOLUMES[a] + V[maxi] - V[l]
        VALUES[a] = VALUES[a] + C[maxi] - C[l]
        // Move to the next stem piece (under the current piece)
        maxi = l
ENDWHILE

OUTPUTS:
        VOLUMES -- volumes of each timber assortment
        VALUES -- values of each timber assortment
```

References:

Laasasenaho, J. 1982. Taper curve and volume functions for pine, spruce and birch. Seloste: Männyn, kuusen ja koivun runkokäyrä- ja tilavuusyhtälöt. Comm. Inst. Forestalis Fenniae 108. 72 s.

Näsberg, M. 1985. Mathematical programming models for optimal log bucking. Department of Mathematics. Linköping Studies in Science and Technology. Dissertation No. 132. Linköping University, Sweden. 199 p.

8.6 Prediction model library

The prediction model library contains the implementations of prediction models used in the simulation. The model library may be written in C or in Python.

8.6.1 Models implemented in C

The model library implemented in C includes source and header files.

Header file

The header file begins by defining a precompiler macro that directs the building of the shared libraries on different operating systems (dll, dylib, so). Dll is used as a convention for the filetype extension on all platforms. When compiling you should pass the value of the MODE parameter to the compiler (-D option on gcc). MODE==1 is for Windows and MODE==3 is for Mac or Linux. MODE==2 is used on Windows when compiling an aggregation library out of individual model library source files (links between model libraries). In that case none of the functions will be marked for exporting outside the dll, except for the functions in the aggregation library:

```
#if MODE==1
# define DLLEXPORT __declspec (dllexport)
#elif MODE==2
# define DLLEXPORT
#elif MODE==3
# define DLLEXPORT
#endif
```

Also numeric constants should be defined here.:

```
#define PI 3.1415926535897932
```

In the header file all the implemented functions must be defined. The model function definitions follow a simple rule: First define the actual model variables; d, D_domvmi, i_Hdom5, cd, CR_domvmi, RDF_L in the Height_growth_pine_Hynynenym function. The number of these variables naturally varies between models. These variables are follwed by a standard part repeated exatcly same for all the functions: `int *nres` (number of results), `double *modelresult` (the actual modelresult), `char *errors` (any error messages generated during the function call), `int errorCheckMode` (error checking mode - currently unused, but must be present), `double allowedRiskLevel` allowed risk level - currently unused, but must be present) and `double rectFactor` (rectification factor, used to multiply the model result if set, defaults to 1.0). This standard function signature allows dynamic call generation in the simulator core.:

```
DLLEXPORT int Height_growth_pine_Hynynenym (double d, double D_domvmi,
    double i_Hdom5, double cr, double CR_domvmi, double RDF_L,
    int *nres, double *modelresult, char *errors, int errorCheckMode,
    double allowedRiskLevel, double rectFactor);
DLLEXPORT int Height_growth_spruce_Hynynenym (double d, double D_domvmi,
    double i_Hdom5, double cr, double CR_domvmi, double RDF_L,
    int *nres, double *modelresult, char *errors, int errorCheckMode,
    double allowedRiskLevel, double rectFactor);
...
```

Source file

In the actual source file, the included header files are first listed:

```
#include "DynamictreemodelLibrary.h"
#include "SimoUtils.h"
#include <stdio.h>
#include <math.h>
```

```
#include <stdlib.h>
#include <string.h>
```

Followed by the function definitions:

```
int Height_growth_pine_Hynynenym(double d, double D_domvmi, double i_Hdom5,
                                 double cr, double CR_domvmi, double RDF_L,
                                 int *nres, double *modelresult, char *errors,
                                 int errorCheckMode, double allowedRiskLevel,
                                 double rectFactor) {
    int ret = 1;
    //Check for fatal errors
    char errorStr[200] = "";
    if (D_domvmi<=0)
    {
        ret = 0;
        constructErrorMessage (errorStr, "D_domvmi<=0(D_domvmi=",D_domvmi);
    }
    if (CR_domvmi<=0)
    {
        ret = 0;
        constructErrorMessage (errorStr, "CR_domvmi<=0(CR_domvmi=",CR_domvmi);
    }
    if (ret == 0) {
        strcat(errors, errorStr);
        return 0;
    }
    //To avoid problems caused by very little diameter minimum diameter for '
    // pine is 0.1 cm!
    if (d<0.1)
        d=0.1;
    *nres = 1;
    double res =i_Hdom5*pow((d/D_domvmi),(0.80171*pow(i_Hdom5,0.4353)-0.49724*
                (cr/CR_domvmi)-0.64383*cr+0.08193*RDF_L));
    *modelresult = res * rectFactor;
    return ret;
}
```

Returning more complex structures than simple real numbers

TODO: write an example of using structs as model results.

Interfacing the C functions to the Python simulator

The C-functions compiled into dll files must be accompanied by Python wrapper modules for the simulator to be to call the functions. There is an utility script for generating the wrapper module files based on the C header files. If you have the development distribution of SIMO, you can find the script in src/utils folder, it's called hToPyWrap.py. See its content for usage. The generated wrappers should be in the same folder as the model dll-files.

8.6.2 Models implemented in Python

Each model definition must be of the form:

```
def model_name(arg, object_index):
```

in which:

- *arg* – PredictionArg object containing all the data being passed between the simulator and the prediction model

- *object_index* – index number for the object for which the prediction is being made

The arg data object is an instance of **PredictionArg** class (see simo.simulation.model.predictionarg) and has a structure described below with examples on how to access the object attributes.

The values of the input variables of the model. The number and order of these must match with the variable declarations in the variables section of the model definition in the model xml file:

```
arg.variables[variable_index, object_index]
```

The model parameter values. The order in which these are used must match the order given in the parameters section of the model xml file:

```
arg.parameters[parameter_index]
```

Number of result objects. This is always 1 unless the prediction model really creates new objects; e.g. in stem distribution models each diameter or height class is a new object:

```
arg.nres[object_index]
```

Stores the model results. Again the results must be stored here in the same order as defined in the result/variables section of the model xml file:

```
arg.mem[variable_index, result_index]
```

For storing possible error messages. Before actually evaluating the model, you should test for error situations and set error messages detailing why the model can't be evaluated and then exit:

```
arg.errors[object_index]
```

When getting or setting the attribute values of arg, object_index is always used for the object_index; e.g. the value of the first explaining variable of the model is accessed by arg.variables[0][oind]. As can be seen from the example indices always start from 0.

The model must return an integer as its return value. 1 if there were no errors and 0 if errors occurred.

CHAPTER

NINE

SIMULATOR DOCUMENTS IN SIMO

This chapter contains descriptions of the different types of simulators build in SIMO.

The first two simulators are individual tree based, distance independent simulator, and stand level simulator for Finnish conditions. These are described in Finnish.

9.1 Puusimulaattori

9.1.1 Johdanto

Tämä dokumentti sisältää SIMO-järjestelmän puutason simulaattorin kuvauksen. Simulaattorissa käytetään Mela2002 puutason kasvumalleja. Kuvaus on tarkoitettu antamaan sekä järjestelmän käyttäjille että järjestelmän ylläpitäjille ja kehittäjille yleiskuvan niistä tiedoista ja toiminnoista, joita puutason malliketjut sisältävät. Lisäksi on kuvattu malliketjujen rakennetta. Vaikka puhutaan puutason simulaattorista, puu tarkoittaa tässä puujoukkoa. Puujoukolla on edustaja esim. keskipuu, jolla on puutason tunnukset. Tämä keskipuu edustaa puujoukkoa eli tiettyä määrää samassa esim. läpimittaluokassa olevia puita. Kaikki puutasoa koskevat laskennat ja toimenpiteet siis koskevat puujoukkoa eivätkä yksittäistä puuta! Dokumentissa käytetään puutasolla termiä taimi, kun tarkoitetaan alle 1,3 m pituista puuta. Ositetasolla puhutaan taimikko-ositteesta, kun ositteeseen kuuluu vähintään yksi alle 1,3 m pitkä puu. Kuviotasolla puhutaan taimikkokuviosta, jos jokin kuvion osite on taimikko-osite eli sillä on vähintään yksi alle 1,3 m pitkä puu. Näin huomioidaan laskelmissa niiden puiden läsnäolo, joilla ei vielä ole rinnankorkeusläpimittaa. Dummy-muuttujien kohdalla asetettu arvo ilmaistaan numeroin 0 tai 1. Puutason simulointia käytetään yleisesti suunnittelujärjestelmissä. Pitkällä aikavälillä puutason mallit saattavat kuitenkin tuottaa epäluotettavia ennusteita ja toisaalta suuralueiden suunnitteluun ne ovat usein laskennallisesti liian raskaita. Suuraluetasolla suunnittelussa riittäisi vähäisempikin tarkkuus. Tämän vuoksi puutason simulointi sopii parhaiten lyhyen ajan tarkempaan suunnitteluun. Tavoitteina erilaisten simulaattoreiden kehittämiselle voidaan siten nähdä koko metsäsuunnittelun laskelmien keventäminen (tarkoituksenmukaiselle tasolle tavoitteen mukaan) ja luotettavuuden parantaminen.

9.1.2 Malliketjut

Malliketjut:

- Complete data
- Calculate missing attributes
- Generate trees
- Calculate basic attributes

lasketaan vain kerran simuloinnin aluksi aineiston täydentämiseksi vastaamaan simulaattorin tarpeita. Näitä malliketjuja käytetään tavanomaisen kuviottaisen arvioinnin tuloksena tuotetun keskitunnustiedon kanssa. Näiden ketjujen lisäksi täydennyksen yhteydessä voidaan laskea myös normaalisti simuloinnin yhteydessä käytettävät ketjut Calculate biomass, Calculate_stand_value_growth, Calculate_stand_productive_value ja

Chain_calculate_v_value. Simulointiin, nykytilan laskentaan tai metsikön tulevaisuuden tilan ennustamiseen käytetään ketjuja:

- Regeneration
- Regeneration tapio alternatives
- Calculate ingrowth
- Calculate tree survival
- Grow seedling stratum
- Grow little tree
- Grow big tree
- Update basic attributes
- Natural removal
- Calculate biomass
- Calculate stand value growth
- Calculate stand productive value
- Chain_calculate_v_value

Regeneration tapio alternatives -ketjua ei ole erikseen dokumentoitu, koska sen perusrakenne on sama kuin Regeneration -ketjun. Regeneration -ketjua voidaan käyttää myös toimenpidemalliketjujen yhteydessä. Ketjun sijainti riippuun laskennassa käytettävästä aika-askeleesta. Toimenpiteet toteutetaan malliketjuilla:

- Harvests
- Harvests branching
- Harvests branching tapio alternatives
- Silvicultural operations
- Silvicultural operations tapio alternatives

Näistä ketjuista on seuraavassa dokumentoitu vain Harvest branching ja Silvicultural operations. Edellinen sisältää hakkuiden ja jälkimmäinen metsänhoitotoimenpiteiden toteutuksen. Hakkuiden toteutuksessa on mukana simuloinnin haaroitus (branching). Muut toimenpideketjuvaihtoehdot ovat edellä mainittujen ketjujen kanssa perusrakenteeltaan identtisiä. Harvest -toimenpideketjussa on peruskasvatus ilman simuloinnin haaroitusta. Harvest branching tapio alternatives sisältää Metsätalouden kehittämiskeskus Tapion suositusten mukaiset kasvatus- ja toimenpidevaihtoehdot haaroituksineen. Lisäksi toimenpiteiden pakotusta varten on oma malliketjunsa:

- Forced operations

9.1.3 Aineiston täydennysketjut

Complete data

Käytetään aineiston kuvio- ja ositetietojen täydentämiseen ennen kasvatuksessa tarvittavien muuttujien laskentaa. Ketjua käytetään vain kerran kullekin aineistolle.

Chain Complete stand attributes

Käytetään kuvion maantieteellisten, kuvion maaperää ja metsähoidollisten toimenpiteiden tilaa kuvaavien tunnusten laskentaan. Asetetaan kuviotason muuttujien oletusarvoja. Jos ositetta ei ole, asetetaan aukea-muuttujan arvoksi 1. Metsätalousmaakuvioille lasketaan kuntatietoihin perustuen YKJ mukaiset koordinaatit, jos koordinaateilla ei aineistossa alunperin ole arvoa. Lasketaan EUREF_FIN -järjestelmän mukaiset maantieteelliset koordinaatit. Näiden koordinaattien avulla lasketaan sijainnin mukaan määräytyvien muuttujien kuten korkeus merenpinnasta ja lämpösumma arvot. Jos sijainnin perusteella korkeus_merenpinnasta -muuttujalle ei saada arvoa (tyypillisesti sijainnin epätarkkuuden vuoksi osutaan vesistön alueelle), asetetaan muuttujan arvoksi 5 m. Asetetaan hinta-alueeksi 1. Sijainnin mukaan voidaan hintatauluihin määritellä toimenpiteille erilaiset hinnat. Tällä hetkellä

on käytössä vain yhdet hinnat eli hinta-alue 1 tarkoittaa koko maata. Tukkivähennystä varten asetetaan tukkivähennysalue -muuttujan arvoksi sama kuin metsäkeskus -muuttujan arvo. Ahvenanmaa ja Pohjois-Suomi -muuttujien oletusarvoiksi 0. Tämän jälkeen asetetaan Ahvenanmaa -muuttujan arvoksi 1, jos kohde sijaitsee Ahvenanmaan maakunnassa. Pohjois-Suomi -muuttuja saa arvon 1, jos kohde on tiettyjen metsäkeskusten alueella. Edelleen asetetaan maaluokaksi keskikarkea moreeni, jos maaluokkaa ei aineistosta löydy. Oletusarvona asetetaan kantojen nostossa hehtaarille jätettävien kantojen lukumääräksi 25. Jos maaluokka on savi, asetetaan hehtaaria kohti nostamatta jäävien kantojen lukumääksi 50. Bioenergian keruumallia varten asetetaan kantojen_keruu ja oksien_keruu -muuttujille arvoksi 1. Edelleen asetetaan kantojen ja oksien keräämiseen bioenergiatarkoitukseen sopimattomilla kohteilla kantojen_keruu ja oksien_keruu -muuttujille arvoksi 0. Asetetaan niiden kuvioden kehitysluokaksi aukea ala (1) ja ikäluokaksi 0, joilla ei ole ositteita. Lopuksi tarkastellaan vielä muuttujaa, joka kertoo kuluneen ajan viimeisestä ojituksesta kuviolla. Jos aika_ojituksesta -muuttuja puuttuu, mutta ojitusvuosi on sen sijaan tiedossa, lasketaan aika ojituksesta. Jos sekä aika_ojituksesta että ojitusvuosi puuttuvat, asetetaan aika_ojituksesta -muuttujan arvoksi nolla.

Chain Complete stratum variables

Käytetään puulajiositteen alkuarvojen asettamiseen. Asetetaan puulajiositteille muuttujien oletusarvoja, kun kyseessä on puustoinen kuvio. Lopuksi asetetaan runkoluku ja pohjapinta-ala nollaksi, jos muuttujalla ei ole arvoa.

Calculate missing attributes

Käytetään aineiston kuvio- ja ositetietojen täydentämiseen ennen puuston kasvatusta. Ketjua käytetään vain kerran kullekin aineistolle. Tarvittavat lisätiedot riippuvat puulajista (eri puulajeilla on erilaisia malleja, joihin tarvitaan tiettyjä muuttujia).

Chain Assign missing initial variables to stratum

Käytetään ositetason muuttujien täydentämiseen, kun kyseessä on puustoinen kuvio. Lasketaan puulajiositteelle ikälisä, siis aika, joka kuluu puun kasvamiseen rinnankorkeuteen. Jos ositteen keskiläpimitta on suurempi kuin nolla, tarkistetaan, että keskiläpimitan ja keskipituuden suhde on järkevä. Keskipituusarvoa käytetään edelleen valtapituuden määrittämiseen. Edelleen lasketaan runkoluku, jos sitä ei vielä ositteella ole. Lopuksi lasketaan ositteen pohjapinta-ala ja asetetaan vain_runkoluku_mitattu -muuttujan arvoksi 1, jos keskiläpimitta on suurempi kuin nolla, mutta pohjapinta-alan arvoa ei ole.

Chain Count number of strata and calculate stand attributes

Käytetään kuviotason muuttujien täydentämiseen, kun kyseessä on puustoinen kuvio. Lasketaan kuvion runkoluku sekä odotettavissa oleva runkoluku. Edelleen lasketaan ositteiden lukumäärä ja aritmeettinen keskipituus. Aritmeettinen keskipituus tarkoittaa tässä ositteiden keskipituuksien keskiarvoa. Summataan kuvion pohjapinta-ala. Jos pohjapinta-ala on suurempi kuin nolla, lasketaan keskiläpimitta ja keskipituus sekä valtapituus pohjapinta-alalla painottaen. Kuvion pääpuulajiksi ja iäksi määräytyy sen ositteen arvo, jonka pohjapinta-ala on suurin. Jos pohjapinta-ala on nolla, asetetaan myös keskiläpimitta nollaksi. Keski- ja valtapituus lasketaan tällöin runkoluvulla painottaen. Edelleen kuvion pääpuulajiksi ja iäksi määräytyy sen ositteen arvo, jonka runkoluku on suurin. Lopuksi asetetaan pääpuulaji_vanha -muuttujan arvoksi sama kuin pääpuulaji.

Chain Whether stratum belongs to upper storey

Käytetään ositetason muuttujien täydentämiseen, kun kyseessä on puustoinen kuvio. Lasketaan kuuluuko osite ylempään jaksoon. Ositteen kuuluminen ylempään jaksoon tulkitaan vertaamalla kuvion aritmeettista keskipituutta ositteen pohjapinta-alalla painotettuun keskipituuteen.

Chain Calculate attributes if stand is two-storey

Käytetään kuviotason muuttujien täydentämiseen, kun kyseessä on puustoinen kuvio. Ositteiden tietoja käyttäen tulkitaan onko kuvio kaksijaksoinen. Kaksijaksoisilla kuvioilla lasketaan sekä ylempään että alempaan jaksoon kuuluvien ositteiden lukumäärät. Jos eri ositteisiin kuulumisen tulkinnan kanssa on ollut ongelmia eli ylemmän tai alemman jakson ositteiden lukumäärää ei ole laskettu (esim. kaikki ositteet kuuluvat ylempään jaksoon, siis kyseessä on yksijaksoinen kuvio), asetetaan ylempään ja alempaan jaksoon kuuluvien ositteiden lukumääräksi nolla. Tällöin myös kaksijaksoisuudesta kertovan muuttujan arvoksi asetetaan 0 eli kuvio tulkitaan yksijaksoiseksi.

Chain Update upper-storey attribute if problems defining two-storey stand

Käytetään, kun kaksijaksoisuuden määrittelyn kanssa on ollut ongelmia puustoisella kuviolla. Tällöin myös ositetasolla virheellinen tieto kuulumisesta ylempään jaksoon korjataan.

Chain Calculate missing attributes to stand

Käytetään asetettaessa simuloinnissa tarvittavia kuviotason muuttujia puustoisella kuviolla. Aluksi, jos kuvion pääpuulaji on hieskoivu, asetetaan hieskoivumuuttujan arvoksi 1. Jos kyseessä on kaksijaksoinen kuvio, lasketaan sekä ylemmän että alemman jakson keski-ikä. Tämän jälkeen lasketaan jaksojen ikäero. Ikäeroa käytetään ylispuustoisen taimikon määrittämiseen kehitysluokkamallissa. Ennen kuvion kehitysluokan tulkintaa lasketaan uudistamisrajat. Uudistamisrajana sekä keskiläpimitalle että keski-iälle käytetään tunnuksen sallitun vaihteluvälin keskiarvoa (0,5). Jos kehitysluokka on siemenpuu-, suojuspuumetsikkö tai ylispuustoinen taimikko, asetetaan siemenpuukuviomuuttujan arvoksi 1. Lopuksi vielä asetetaan siemenpuukuvio_ilman_taimikkoa -muuttujan arvoksi 1, jos kyseessä on siemenpuu- tai suojuspuumetsikkö.

Chain Calculate missing initial variables to stratum

Käytetään ositetason muuttujien täydentämiseen, kun kyseessä on puustoinen kuvio. Siemenpuuositemuuttujan arvoksi asetetaan 1, jos kyseessä on ylispuustoisen taimikkokuvion ylempi osite tai siemen- tai suojuspuukuvion osite. Jos ositteen keskipituus on pienempi kuin 1,3 m, asetetaan taimikkomuuttujan arvoksi 1. Lasketaan ositteen rinnankorkeusikä. Jos rinnankorkeusikä saa näin negatiivisen arvon, se asetetaan nollaksi. Jos ositteelle lasketun ikälisän seurauksena syntyy tilanne, jossa rinnankorkeusikä on nolla, mutta ositteen keskiläpimitta on kuitenkin suurempi kuin nolla, muutetaan rinnankorkeusikä vastaamaan aineiston arvoja. Lopuksi lasketaan iän ja pituuden kalibrointikerroin ositteille, joilla keski-ikä ja -pituus on suurempi kuin nolla. Kalibroinnilla pyritään siihen, että simuloinnissa käytettävistä kuvauspuista laskettu keski-ikä ja -pituus olisi sama kuin ositteen keski-ikä ja -pituus alunperin.

Chain Calculate still missing initial stand attributes

Käytetään kuviotason muuttujien täydentämiseen, kun kyseessä on puustoinen kuvio. Lasketaan koivun osuus pohjapinta-alasta. Edelleen lasketaan taimikko-ositteiden lukumäärä. Jos kuviolla ei ole yhtään taimikko-ositetta, asetetaan taimikkokuviosta kertovan muuttujan arvoksi 0.

Generate trees

Käytetään, kun halutaan tuottaa metsikön keskitunnustiedoista puutason tietoa kuvauspuiden käyttöä varten. Ketjua käytetään vain kerran kasvatusjakson aluksi kullekin aineistolle.

Chain Clear tree objects

Käytetään puustoiselle kuviolle mahdollisten puutason tietojen tuhoamiseen. Ositteelta (ei taimikko) hävitetään puutason kaikki muuttujat ja niiden tiedot, jotta uuden jakaumamallin tietoja ei sekoitettaisi vanhan jakaumamallin tietoihin, jos sellainen on olemassa (ei pitäisi olla).

Chain Generate trees

Käytetään jakaumamallien muodostamiseen puustoisella kuviolla. Asetetaan mitattu_runkoluku ja vain_runkoluku_mitattu -muuttujien arvoksi 0, jos muuttujalla ei ole vielä arvoa. Malliketjussa tehdään ensin jako keskitunnuksien perusteella iso ja pieni puustoisiin ositteisiin. Pienille puustoille (määritetään keskiläpimitan mukaan) lasketaan pituusjakauma suoraan, mutta varttuneemman puuston kohdalla läpimitta-jakaumamallin valinta riippuu puulajista, maaperästä (mineraali- vai turvemaa), kehitysluokasta, sekä siitä, onko puuston runkoluku mitattu ja onko kyseessä siemenpuuosite. Pienille puille laskettavan pituusjakauman yhteydessä ennustetaan myös puiden läpimitat. Jos ositteen puut muodostetaan pituusjakaumalla, päivitetään taimikko-ositteesta kertovan muuttujan arvo. Tätä varten lasketaan ositteella olevien alle 1,3 m pitkien puiden lukumäärä. Edelleen normaalijakaumamuuttujan arvoksi asetetaan 1. Puiden muodostamisen jälkeen asetetaan puut_generoitu -muuttujan arvoksi 1.

Chain Update seedling_below13 to stand level and calculate age class

Käytetään taimikkokuviomuuttujan päivittämiseen, kun kyseessä on puustoinen kuvio. Lasketaan taimikko-ositteiden lukumäärä. Jos kuviolla on yksikin taimikko-osite, asetetaan taimikkokuviosta kertovan muuttujan arvoksi 1. Lopuksi lasketaan kuvion ikäluokka. Ikäluokan määritystä varten lasketaan kuvion keski-ikä. Jos kyseessä on taimikkokuvio, kuvion keski-iäksi valitaan sen ositteen ikä, jonka runkoluku on suurin, muuten iäksi asetetaan pohjapinta-alaltaan suurimman ositteen ikä.

Calculate basic attributes

Käytetään puiden luonnin jälkeen täydentämään puuston kasvatukseessa tarvittavia puu-, osite- ja kuviotason tietoja simulointia varten. Ketjua käytetään vain kerran kullekin aineistolle!

Chain Calculate number of sample trees

Käytetään kuvauspuiden lukumäärän laskentaan, kun kyseessä on puustoinen kuvio. Lasketaan kuvauspuiden lukumäärä ositteelle, joille puut on jo olemassa (saatu suoraan datasta tai luotu jakaumamalleilla).

Chain Calculate tree attributes

Käytetään puutason muuttujien täydentämiseen, kun kyseessä on puustoinen kuvio. Ensin asetetaan puutason muuttujille alkuarvoja. Edelleen asetetaan taimesta_varttuneeksi -muuttujan oletusarvoksi 0. Malliketjulla lasketaan ei-taimikko-ositteen puille pituudet sekä iät. Lisäksi lasketaan yksittäisen puun sekä läpimittaluokan pohjapinta-ala ja asetetaan taimesta (h < 1,3m) kertovan muuttujan arvoksi 0. Taimikko-ositteelle puiden iäksi asetetaan ositteen keski-ikä. Jos puulla on olemassa läpimitta, lasketaan yksittäisen puun sekä läpimittaluokan pohjapinta-ala. Edelleen lasketaan puulle tilavuus. Jos puun läpimitta on suurempi kuin nolla, lasketaan puun läpimitta kannonkorkeudelta, poikkileikkausala rinnankorkeudelta neliömetreinä sekä minimi kasvutila. Lopuksi lasketaan kutakin puuta suurempien puiden muodostama pohjapinta-alan ja suhteellinen tiheys puulle. Laskutapa riippuu kuvauspuiden lukumäärästä sekä siitä, onko kysessä taimikko- vai ei-taimikkokuvio.

Chain Calculate stratum attributes

Käytetään ositetason muuttujien täydentämiseen, kun kyseessä on puustoinen kuvio. Kun ositteelle on muodostettu jakaumamallilla kuvauspuut, lasketaan ositteen runkoluku, tilavuus, suhteellinen tiheys. Edelleen taimikko-ositteelle lasketaan suhteellisen tiheyden lisäksi pohjapinta-ala ja keski-läpimitta. Ei-taimikko-ositteelle lasketaan lisäksi keskiläpimitta, keskipituus ja pohjapinta-ala. Ositteelle, jolle on muodostettu jakaumamallilla kuvauspuut lasketaan vielä läpimitta kannonkorkeudelta. Lopuksi lasketaan taimikko-ositteelle vuotuinen pituuskasvu ositetason kasvatusta varten ja keskimääräinen rinnankorkeusläpimitan arvo asetettavaksi, kun 1,3 m saavutetaan sekä asetetaan tuotokseksi ositteen tilavuus.

Chain Calculate stand attributes before calculating site index

Käytetään kuviotason muuttujien täydentämiseen, kun kyseessä on puustoinen kuvio. Lasketaan kuvion tilavuus, kuvauspuiden lukumäärä ja runkoluku ositteiden tietoja summaamalla. Edelleen lasketaan niiden ositteiden lukumäärä, joille on jakaumamalleilla tuotettu kuvauspuut. Tämän tiedon perusteella asetetaan kuvion puut_generoitu- ja puhdas_taimikko-muuttujien arvot. Edelleen lasketaan kuvion tuotos. Jos kyseessä ei ole puhdas taimikkokuvio, lasketaan suhteellinen tiheys koko kuviolle sekä puulajeittain. Lisäksi lasketaan suhteellinen tiheys lehtipuille, muille lehtipuille kuin koivulle sekä muille havupuille kuin männylle tai kuuselle.

Chain Calculate site index

Käytetään kasvupaikkaindeksin laskemiseen ositteelle puustoisella kuviolla. Ositteelle, joille puut on jo olemassa (saatu suoraan datasta tai luotu jakaumamalleilla) lasketaan kasvupaikkaindeksi.

Chain Calculate stand attributes

Käytetään täydennettäessä kuviotason tietoa simulaattorin tarpeita vastaavaksi puustoisella kuviolla. Aluksi lasketaan kuviolle ositteiden lukumäärä (ei sisällä sisäänkasvuvaiheessa olevia ositteita eli ositteita, joilla ei ole vielä ikää) ja aritmeettinen keskipituus. Ei-taimikkokuviolle lasketaan pohjapinta-alalla painotettu keskiläpimitta ja -pituus sekä pohjapinta-ala. Lopuksi lasketaan koivun ja kuusen pohjapinta-alojen suhteelliset osuudet koko kuvion pohjapinta-alasta.

Chain Calculate dominant height

Käytetään täydennettäessä valtapituustiedot simulaattorin tarpeita vastaavaksi puustoisen kuvion ositteelle, joilla puut on jo olemassa (saatu suoraan datasta tai luotu jakaumamalleilla). Lasketaan mineraalimaan kuviolla ositteen valtapituuskasvu. Turvemaakuviolla lasketaan ositteen valtapituus.

Chain Calculate crown ratio

Käytetään täydennettäessä puutason tietoa simulaattorin tarpeita vastaavaksi puustoisella kuviolla. Lasketaan puun latvussuhde eli elävän latvuksen suhde puun pituuteen, kun puu on ylittänyt rinnankorkeuden. Jos puulla ei ole rinnankorkeusläpimittaa eli se on vähemmän kuin 1,3 m pitkä, latvussuhdetta ei voida laskea. Edelleen lasketaan tukkivähennys puille, joiden rinnankorkeusläpimitta on suurempi kuin 10 cm.

Chain Calculate crown ratio and diameter for dominant trees

Käytetään täydennettäessä kuviotason tietoa simulaattorin tarpeita vastaavaksi puustoisella kuviolla, joille puut on jo olemassa (saatu suoraan datasta tai luotu jakaumamalleilla). Lasketaan puuston valtapuiden latvussuhde sekä läpimitta. Valtapuiden latvussuhde tarkoittaa keskipuuta paksumpien puiden keskiarvo. Samoin puuston valtapuiden läpimitta tarkoittaa keskipuuta paksumpien puiden keskiarvoa. Laskennan toteutus riippuu siitä onko kyseessä taimikko- vai ei-taimikkokuvio. Lopuksi lasketaan kaikille puustoisille turvemaan kuvioille ojitustarve.

Chain Calculate the value of growing stock on stratum level

Käytetään ositteen puuston arvon sekä puutavaralajijakauman ja bioenergiaksi käytettävien kantojen ja hakkuutähteen määrän laskemiseen. Asetetaan puustoisen kuvion ositteen puuston arvon oletusarvoksi nolla. Jos kuvion kehitysluokka on jokin muu kuin aukea uudistusala tai nuori taimikko ja kyseessä ei ole taimikko-osite, lasketaan ositteen puuston arvo, puutavaralajijakauma sekä bioenergiapuun määrä. Huom! Jos bioenergian määrä halutaan saada tässä selville, on biomassa laskettava ensin puutasolla. Tämä onnistuu malliketjulla Calculate biomass. Normaalisti alkuarvojen yhteydessä biomassaa ei lasketa.

Chain Calculate the value of growing stock on stand level

Käytetään kuvion puuston arvon määrittämiseen. Aluksi asetetaan kuvion puuston oletusarvoksi nolla. Jos kyseessä on puustoinen kuvio, summataan ositteiden puustojen arvot yhteen ja näin saadaan kuvion puuston arvo.

9.1.4 Simulointiketjut

Regeneration

Käytetään puuston uudistamisen toteutukseen ja uudistamisen jälkeisen puuston muuttujien arvojen laskemiseen. Ketjua käytetään tavallisesti jokaisen kasvatusjakson alussa. Sitä voidaan käyttää myös kasvatusjakson lopussa toimenpidemallien jälkeen, jos uudistamistoimenpide halutaan toteuttaa saman vuonna uudistushakkuun kanssa. Tämä tulee kyseeseen erityisesti silloin, kun kasvatusjakso on pitkä (esim. 5 vuotta). Ilman tällaista menettelyä uudistamistoimenpide tulisi toteutettavaksi vasta seuraavalla 5-vuotisjaksolla.

Chain Regenerate clear cutted stand and open area using planting

Käytetään luotaessa avohakatulle alalle tai lähtötiedoissa aukealle alueelle uusi puusto istuttamalla. Uudistettavan kuvion puulaji määräytyy kasvupaikan mukaan. Istutusmallin jälkeen asetetaan kuviotason muuttujille alkuarvoja. Edelleen asetetaan istutuksesta kertovan muuttujan arvoksi 1. Lasketaan runkoluku, ikä, pohjapinta-ala, tilavuus ja kuvion odotettavissa oleva runkoluku sekä pääpuulaji ositetietojen avulla.

Chain Calculate attributes to stratum after planting

Käytetään istutuksen jälkeen ositetietojen täydentämiseen. Lasketaan ositteelle ikälisä eli aika, joka kuluu rinnankorkeuden saavuttamiseen.

Chain Natural regeneration after seed tree cut

Käytetään luontaisen uudistamisen toteutukseen siemenpuu-, suojuspuu- ja kaistalehakkuin käsitellyille kuvioille. Aluksi asetetaan seuraavalla viiden vuoden jaksolla syntyvien puiden määräksi nolla. Uudistushakkuuta jälkeen lasketaan luontaisesti syntyvien puiden lukumäärä, kun hakkuu on tehty siemenpuu- , suojuspuu- tai kaistalehakkuuna tai jos kuvio on määritetty siemen- tai suojuspuukuvioksi, mutta taimiositetta ei vielä ole olemassa. Laskentaa varten tarvitsee kuitenkin määrittää ensin uudistuksessa käytettävä puulaji, jos se puuttuu ja tämän jälkeen laskea keskimääräinen luontaisesti syntyvän puuston runkoluku kasvupaikan ja puulajin mukaan. Jos edellä laskettu syntyvien puiden lukumäärä on suurempi kuin nolla, seuraavaksi määritetään kuvion pääpuulaji, jos sitä ei ole tiedossa (kaistelehakkuussa pääpuulaji puuttuu). Uudistushakkuun jälkeisen luontaisen uudistamisen seurauksena syntyvä uusi puusto kuvataan viidellä uudella kuvauspuulla. Alussa jokainen kuvauspuu saa viidesosan kuviolle syntyvästä runkoluvusta. Jokaiselle kuvauspuulle määräytyy puulaji seuraavasti: Ensimmäisen kuvauspuun puulaji (pääpuulaji) määräytyy kasvupaikan ja maaperän (kivennäismaa/turvemaa) mukaan. Tämän jälkeen neljä muuta kuvauspuuta saavat puulajin satunnaisluvun avulla. Tietyn puulajin todennäköisyys määräytyy pääpuulajin, kasvupaikan ja maaperän (kivennäismaa/turvemaa) mukaan. Samaa puulajia olevat ositteet summataan yhteen, joten luontaisen uudistamisen seurauksena syntyy aina 1-5 uutta ositetta. Luontaisen uudistamisen jälkeen asetetaan kuviotason muuttujille alkuarvoja. Lopuksi päivitetään luontaisesti syntyneiden sisäänkasvuvaiheessa olevien ositteiden yhteenlaskettu runkoluku. Sisäänkasvuvaihe tarkoittaa vaihetta, jossa osite on jo olemassa, mutta puiden syntyminen tapahtuu vasta tulevina vuosina.

Chain Calculate attributes to natural ingrowth stratum

Käytetään luontaisesti uudistetun kuvion ositetason muuttujien täydentämiseen. Samalla kasvatusjaksolla luontaisesti syntyneille ositteille lasketaan taimikon tuleva syntymävuosi. Tätä varten lasketaan ositteille teoreettinen ikä, joka saa arvon -1 ja -5 väliltä (tämä tarkoittaa, että puut syntyvät ositteelle 1 - 5 vuoden kuluttua). Edelleen lasketaan rinnankorkeuden saavuttamiseen kuluva aika. Lopuksi asetetaan luontaisen uudistamisen jälkeen ositetason muuttujille alkuarvoja.

Chain Update N_ingrowth to stratum level

Käytetään luontaisesti syntyneen sisäänkasvuvaiheessa olevan ositteen runkoluvun päivittämiseen. Kun luontaisesti syntyneen ositteen teoreettinen ikä saa arvon nolla, eli on vuosi, jolloin puut syntyvät, siirretään sisäänkasvuvaiheessa olleen ositteen runkoluku ositteen oikeaksi runkoluvuksi. Sisäänkasvuvaiheessa olevan ositteen puiden runkoluku asetetaan nollaksi. Tällöin sisäänkasvuvaiheen haamupuista on tullut oikeita puita.

Chain Update N_ingrowth to stand level

Käytetään luontaisesti syntyneen sisäänkasvuvaiheessa olevan kuvion runkoluvun päivittämiseen. Summataan sisäänkasvuvaiheessa olevien ositteiden runkoluvut kuviotasolle. Päivitetään runkoluku. Lopuksi, jos kuvion runkoluku on suurempi kuin nolla, asetetaan avoimesta alasta (puuttomasta kuviosta) kertovan muuttujan arvoksi 0. Samoin asetetaan taimettomasta siemenpuukuviosta kertovan muuttujan arvoksi 0, jos sisäänkasvuvaiheessa olevien puiden runkoluku on suurempi kuin nolla. Lopuksi asetetaan vanha_pääpuulaji -muuttujan arvoksi sama kuin pääpuulaji.

Calculate ingrowth

Käytetään puiden syntymisen ja sisäänkasvun kuvaamiseen. Sisäänkasvu tarkoittaa puustoiselle kuviolle syntyviä uusia puita. Jättämällä tämän ketjun pois simuloinnista voi sisäänkasvun ennustamisen kytkeä pois käytöstä.

Chain Natural ingrowth

Käytetään laskettaessa kuviolle olemassa olevien ositteiden lisäksi luontaisesti syntyvien uusien puiden lukumäärä, kun kyseessä on puustoinen kuvio. Sisäänkasvu on mahdollista metsämaan kuviolla, kun kyseessä ei ole siemen- tai suojuspuukuvio, jolle ei vielä ole syntynyt taimikkoa. Aluksi asetetaan uusien syntyen puiden lukumäärän kertova muuttuja nollaksi ja lasketaan jo olemassa olevien ositteiden lukumäärä, mukaan lukien vielä sisäänkasvuvaiheessa olevat ositteet. Uusia puita voi syntyä kuviolle, jos siemenpuuhakkuusta on vähintään yksi vuosi ja luontaisessa uudistamisessa on jo syntynyt ositteita tai avohakkuukuviolla avohakkuu on tehty edellisenä vuonna. Puita ei kasvatettavan puuston sekaan enää synny, kun kehitysluokka on uudistuskypsä metsikkö, ainakin yksi harvennushakkuu on kuviolle jo tehty tai, kun kuviolla on jo olemassa viisi ositetta. Uusien puiden lukumäärän määrittämiseksi lasketaan puiden lukumäärän vertailuarvo sekä määritetään syntyneille puille puulajit. Puiden lukumäärän vertailuarvon kertovan mallin valinta riippuu siitä onko kyseessä luontaisesti vai keinollisesti uudistettu kuvio. Kasvatettavan puuston sekaan syntyvä puusto kuvataan yhdellä uudella kuvauspuulla ja lasketaan sille runkoluku. Kuvauspuun puulaji määräytyy seuraavasti: Pääpuulaji määräytyy kasvupaikan ja maaperän (kivennäismaa/turvemaa) mukaan, ellei se ole jo tiedossa. Tämän jälkeen kuvauspuulle määrätään puulaji satunnaisluvun avulla. Tietyn puulajin todennäköisyys määräytyy pääpuulajin, kasvupaikan ja maaperän (kivennäismaa/turvemaa) mukaan. Uuden puulajin syntymistä on rajoitettu, jottei synny epäuskottavan tiheitä puustoja. Lopuksi luontaisesta uudistumisesta kertovan ja taimikko-osite muuttujien arvoksi asetetaan 1 sekä päivitetään sisäänkasvuvaiheessa olevien ositteiden yhteenlaskettu runkoluku.

Chain Calculate initial age to ingrowth stratum

Käytetään kuvion ositetason muuttujien täydentämiseen, kun kyseessä on puustoinen kuvio ja sisäänkasvua on tapahtunut. Samalla kasvatusjaksolla luontaisesti syntyneelle ositteelle lasketaan taimikon tuleva syntymävuosi. Tätä varten lasketaan ositteelle teoreettinen ikä, joka saa arvon -1 ja -5 väliltä (tämä tarkoittaa, että puut syntyvät ositteelle 1 - 5 vuoden kuluttua). Edelleen lasketaan rinnankorkeuden saavuttamiseen kuluva aika. Lopuksi asetetaan sisäänkasvun jälkeen ositetason muuttujille alkuarvoja.

Calculate tree survival

Käytetään puiden itseharvenemisen kuvaamiseen. Jättämällä tämän ketjun pois simuloinnista voi luonnonpoistuman ennustamisen kytkeä pois käytöstä. Tällöin pitää myös Natural removal -ketju jättää pois simuloinnista.

Chain calculate tree survival

Käytetään määrittelemään ei-taimikko-ositteen puiden elossa pysymisen todennäköisyyttä, kun kyseessä on puustoinen kuvio (eli kuvauspuun tapauksessa todennäköisyys kuolla kertoo kuinka monta puuta kuvauspuun edustamasta puujoukosta kuolee). Elossa pysymisen todennäköisyyttä saatetaan skaalata vielä kasvatuksen jälkeen, jos sallittu suurin runkoluku kuviolla ylittyy. Puun elossapysymistä kuvataan puun kilpailuasemasta ja ikääntymisestä kertovilla malleilla. Tässä vaiheessa ennustetaan vain puiden elossapysymistodennäköisyyttä, mutta ei tapeta puita!

Grow seedling stratum

Käytetään taimikko-ositteen kasvatukseen sekä tietojen päivittämiseen kasvatuksen jälkeen.

Chain Grow seedling stratum

Käytetään kasvattamaan taimikkoa, kun kyseessä on puustoinen kuvio. Taimikon kasvatus tehdään ositetasolla, vaikka kasvatus muuten MELA-malleilla tapahtuukin puutasolla. Asetetaan muuttujan puut_juuri_generoitu oletusarvoksi 0. Jos kyseessä on taimikko, jossa puita ei ole vielä generoitu ja teoreettinen ikä on nolla, lasketaan rinnankorkeusläpimitan referenssiarvo, kun rinnankorkeus saavutetaan. Edelleen lasketaan vuotuinen pituuskasvu ja kasvatetaan ositteen keskipituutta. Jos keskipituus saavuttaa 1,3 m, lasketaan ositteelle keskiläpimitta. Seuraavaksi muodostetaan ositteelle pituusjakauma ja lasketaan kuvauspuille läpimitat. Asetetaan muuttujien puut_generoitu, puut_juuri_generoitu ja normaalijakauma arvoiksi 1. Lopuksi lasketaan kuvauspuiden lukumäärä.

Calculate tree variables to just generated tree

Käytetään laskettaessa muuttujien arvoja juuri luoduille puille, kun kyseessä on puustoinen kuvio. Jos ositteelle on juuri luotu puut, asetetaan jokaisen kuvauspuun iäksi ositteen ikä. Edelleen asetetaan puutason muuttujille alkuarvoja. Kasvatetaan ikää lisäämällä ikään aika-askeleen pituus. Lasketaan puun tilavuus. Jos puun läpimitta on suurempi kuin nolla, lasketaan yksittäisen kuvauspuun poikkileikkausala rinnankorkeudelta. Edelleen lasketaan jakauman jokaisen läpimittaluokan poikkileikkausala. Lisäksi lasketaan jokaisen yksittäisen kuvauspuun poikkileikkausala neliömetreinä. Lopuksi lasketaan puun kannonkorkeusläpimitta, jonka jälkeen voidaan laskea puun kasvutilan minimi puulajeittaisilla malleilla.

Chain Update mean diameter to stratum

Käytetään keskiläpimitan päivittämiseen, kun kyseessä on puustoinen kuvio. Ositteelle, jolle on juuri muodostettu puut, lasketaan keskiläpimitta.

Grow little tree

Käytetään pienten (alle 1,3 m pitkien) puiden kasvatukseen sekä tietojen päivittämiseen kasvatuksen jälkeen.

Chain Grow little tree

Käytetään kasvattamaan puustoisen kuvion vähintään vuosi aiemmin generoituja puita, joiden pituus on alle 1,3 m. Kasvatetaan ensin puun biologista ikää ja pituutta. Jos puu saavuttaa näin 1,3 m pituusrajan, lasketaan puulle rinnankorkeusläpimitta, yksittäisen puun sekä koko läpimittaluokan poikkileikkausala. Edelleen asetetaan taimimuuttujan arvoksi 0 (kyseessä on yli 1,3 m pitkä puu) ja rinnankorkeuden ylittämisestä kertovan muuttujan arvoksi 1. Lopuksi lasketaan puun poikkileikkausala neliömetreinä, kannonkorkeusläpimitta sekä minimikasvutila.

Chain Calculate volume to little tree

Käytetään puustoisella kuviolla, kun lasketaan tilavuus pienelle puulle. Lasketaan puulle tilavuus, jos puut on juuri luotu ositteelle tai yksittäinen puu on juuri saavuttanut rinnankorkeuden.

Chain Calculate if there is any seedlings below 1,3 m left stratum

Käytetään päivitettäessä puustoisella kuviolla taimikko-osite -muuttujan arvoa. Ositteelle, joille puut on jo olemassa (saatu suoraan datasta tai luotu jakaumamalleilla) lasketaan alle 1,3 m pitkien puiden lukumäärä. Jos alle 1,3 m puiden lukumäärä on nolla, asetetaan taimikko-ositteesta kertovan muuttujan arvoksi 0.

Chain Calculate if there is any seedlings below 1,3 m left stand

Käytetään päivitettäessä puustoisella kuviolla taimikkokuviomuuttujan arvoa. Lasketaan taimikko-ositteiden lukumäärä. Jos taimikko-ositteiden lukumäärä on nolla, kuviolla ei ole yhtään taimikko-ositetta ja taimikkokuviosta kertovan muuttujan arvoksi asetetaan 0. Edelleen lasketaan niiden ositteiden lukumäärä, joille puut on jo olemassa (saatu suoraan datasta tai luotu jakaumamalleilla). Jos kuviolla on yksikin osite, jolla on puita, asetetaan kuviotasolla puut_generoitu -muuttujan arvoksi 1 ja puhtaasta taimikko-kuviosta kertovan muuttujan arvoksi 0. Lopuksi lasketaan vielä kuvauspuiden lukumäärä kuviolla.

Grow big tree

Käytetään suurten (vähintään 1,3 m pitkien) puiden kasvatukseen sekä tietojen päivittämiseen kasvatuksen jälkeen.

Chain Calculate dominant height

Käytetään täydennettäessä valtapituustiedot simulaattorin tarpeita vastaavaksi puustoisella kuviolla ositteelle, jolla puut on jo olemassa (saatu suoraan datasta tai luotu jakaumamalleilla). Lasketaan mineraalimaan kuvion ositteen valtapituuskasvu. Turvemaalla lasketaan ositteen valtapituus.

Chain Grow big trees on peatland

Käytetään kasvattamaan yksittäin puin kuvattua puustoa turvemaalla. Turvamaakuviolla kasvavalle vähintään 1,3 m pitkälle puulle, joka ei ole samalla aikajaksolla saavuttanut rinnankorkeutta eikä ositteen puita ole muodostettu samalla aikajaksolla, lasketaan pituus ennen läpimitan kasvatusta. Tämän jälkeen lasketaan pohjapinta-alan kasvu ja kasvatetaan puun pohjapinta-alaa. Lopuksi päivitetään puun läpimitta sekä läpimittaluokan pohjapinta-ala.

Chain Update mean diameter and basal area on peatland

Käytetään turvemaan ositetunnusten päivittämiseen puiden läpimitan kasvatuksen jälkeen. Turvemaakuvion ositteelle, jonka puita ei ole muodostettu samalla aikajaksolla lasketaan keskiläpimitta ja pohjapinta-ala.

Chain Grow big trees in mineral soil and update on peatland

Käytetään kasvattamaan yksittäin puin kuvattua puustoa mineraalimaalla ja päivittämään turvemaalla puun pituus. Turvamaakuviolla kasvavalle vähintään 1,3 m pitkälle puulle, joka ei ole samalla aikajaksolla saavuttanut rinnankorkeutta eikä ositteen puita ole muodostettu samalla aikajaksolla, lasketaan pituus läpimitan kasvatuksen jälkeen. Kivennäiskuviolla kasvavalle vähintään 1,3 m pitkälle puulle, joka ei ole samalla aikajaksolla saavuttanut rinnankorkeutta eikä ositteen puita ole muodostettu samalla aikajaksolla, lasketaan pituus- ja pohjapinta-alan kasvu. Edelleen kasvatetaan puun pituutta ja pohjapinta-alaa sekä lasketaan läpimitta. Kivennäismailla puun pituuden ja pohjapinta-alan (läpimitan) kasvumalli perustuu valtapuuston pituuskasvuun. Seuraavaksi lasketaan puun tilavuus ja kasvatetaan sekä biologista että rinnankorkeusikää. Lisäksi lasketaan vielä puun läpimittaluokan pohjapinta-ala, läpimitta kannonkorkeudelta, yksittäisen puun pohjapinta-ala neliömetreinä ja puun minimi kasvutila sekä asetetaan rinnankorkeuden saavuttamisesta kertovan muuttujan arvoksi 0.

Update basic attributes

Käytetään tietojen päivittämiseen kasvatuksen jälkeen.

Chain Calculate tree attributes

Käytetään kohdepuuta suurempien puiden muodostaman pohjapinta-alan ja suhteellisen tiheyden määrittämiseen puulle kasvatuksen jälkeen. Lasketaan, jos kyseessä ei ole puhdas taimikko. Yli 1,3 m pitkälle puulle lasketaan sitä suurempien puiden muodostama pohjapinta-ala sekä suhteellinen tiheys.

Chain Calculate stratum attributes

Käytetään ositetason muuttujien päivittämiseen ositteelle kasvatuksen jälkeen. Ensin täydennetään muuttujien arvoja ositteelle, jolle puut on jo olemassa (saatu suoraan datasta tai luotu jakaumamalleilla). Asetetaan ennen kasvatusta ollut (edellisen aikajakson) tilavuus tilavuus_vanha -muuttujan arvoksi, jonka jälkeen päivitetään ositteen tilavuus ja kuvauspuiden lukumäärä. Lasketaan suhteellinen tiheys, pohjapinta-ala, keskiläpimitta ja -pituus sekä taimikko- että ei-taimikko-ositteelle. Lopuksi ositteelle, jolle on muodostettu jakaumamallilla kuvauspuut lasketaan vielä läpimitta kannonkorkeudelta. Taimikko-ositteelle (voi olla myös puhdas taimikko ilman muodostettuja kuvauspuita) lasketaan vuotuinen pituuskasvu ositetason kasvatusta varten ja keskimääräinen rinnankorkeusläpimitan arvo asetettavaksi, kun 1,3 m pituus saavutetaan.

Chain Calculate stand attributes

Käytetään päivittämään puiden kasvatuksen vuoksi muuttuneita kuviotietoja. Asetetaan ennen kasvatusta ollut tilavuus tilavuus_vanha -muuttujan arvoksi. Tämän jälkeen päivitetään tilavuus ja kuvauspuiden lukumäärä. Lasketaan niiden ositteiden lukumäärä, jolle puut on jo olemassa (saatu suoraan datasta tai luotu jakaumamalleilla). Jos kuvion yhdellekin ositteelle on puut jo olemassa, asetetaan puut_generoitu -muuttujan arvoksi 1. Jos taas kuviolla ei ole yhtään ositetta, jolla olisi puita, asetetaan puhdas_taimikko -muuttujan arvoksi 1. Muulle kuin puhtaalle taimikkokuviolle lasketaan kuvion puuston suhteellinen tiheys (eli kaikki ositteet, puulajit yhteensä). Lisäksi lasketaan puulajeittaiset suhteelliset tiheydet.

Chain Calculate still missing stratum attributes

Käytetään ositetason muuttujien arvojen päivitykseen kasvatuksen jälkeen. Lasketaan kasvupaikkaindeksi ositteelle, jolle puut on jo olemassa (saatu suoraan datasta tai luotu jakaumamalleilla). Kasvatetaan rinnankorkeus-, biologista ja teoreettista ikää. Teoreettista ikää käytetään luontaisen uudistamisen ositteiden syntymishetken määrittämiseen. Lasketaan ositteen valtapituus sekä painomuuttuja (runkoluku*keskipituus) taimikkokuvion keski-iän määritystä varten. Lopuksi päivitetään vielä tilavuuskasvun laskentaan liittyvien muuttujien arvoja. Nollataan sekä absoluuttinen että suhteellinen tilavuuskasvu meneillään olevan aikajakson tilavuuskasvun laskentaa varten. Lasketaan uudet arvot absoluuttiselle ja suhteelliselle tilavuuskasvulle. Lopuksi lasketaan ositteen tuotos.

Chain Calculate still missing stand attributes

Käytetään kuvion muuttujien päivittämiseen kasvatuksen jälkeen. Lasketaan kuvion ositteiden lukumäärä ja aritmeettinen keskipituus. Lukumäärään ei huomioida sisäänkasvuvaiheessa olevia ositteita. Jos kyseessä on muu kuin puhdas taimikkokuvio lasketaan keskiläpimitta, keskipituus, pohjapinta-ala sekä koivun ja kuusen pohjapinta-alojen suhteelliset osuudet koko kuvion pohjapinta-alasta. Edelleen lasketaan runkoluku ja valtapituus sekä valitaan kuvion keski-iäksi ja pääpuulajiksi sen ositteen tiedot, jonka pohjapinta-ala on suurin. Taimikkokuviolle lasketaan keskipituus ja valtapituus sekä valitaan kuvion keski-iäksi sen ositteen tiedot, jonka painomuuttujan arvo (runkoluku*keskipituus) on suurin. Nollataan sekä absoluuttinen että suhteellinen tilavuuskasvu meneillään olevan aikajakson tilavuuskasvun laskentaa varten. Lasketaan uudet arvot absoluuttiselle ja suhteelliselle tilavuuskasvulle. Lasketaan kuvion tuotos. Edelleen asetetaan hieskoivusta kertovan muuttujan arvoksi 1, jos kuvion pääpuulaji on hieskoivu. Lopuksi bioenergian keruumallia varten asetetaan kantojen_keruu ja oksien_keruu -muuttujille arvoksi 0, jos kuvion pääpuulaji ei ole bioenergian keruuseen sopiva.

Chain Calculate crown ratio

Käytetään puun latvussuhteen laskentaan kasvatuksen jälkeen ositteelle, jolle puut on jo olemassa (saatu suoraan datasta tai luotu jakaumamalleilla). Lasketaan latvussuhde puille, joiden rinnankorkeusläpimitta on suurempi kuin nolla sekä tukkivähennys puille, joiden rinnankorkeusläpimitta on suurempi kuin 10 cm.

Chain Whether stratum belongs to upper storey

Käytetään tulkittaessa ei-taimikko-ositteen kuulumista puustojaksoihin kasvatuksen jälkeen. Lasketaan kuuluuko osite ylempään jaksoon. Ositteen kuuluminen ylempään jaksoon tulkitaan vertaamalla kuvion aritmeettista keskipituutta ositteen pohjapinta-alalla painotettuun keskipituuteen.

Chain Calculate missing two-storey stand attributes

Käytetään kuviotason muuttujien päivittämiseen kasvatuksen jälkeen. Sekä taimikko- että ei-taimikkokuviolle, jolle puut on jo olemassa (saatu suoraan datasta tai luotu jakaumamalleilla) lasketaan valtapuiden keskiläpimitta ja latvussuhde. Turvemaakuviolle lasketaan ojitustarve. Tulkitaan onko kyseessä kaksijaksoinen kuvio. Jos kuvio ei ole kaksijaksoinen, nollataan sekä ylempään että alempaan jaksoon kuuluvien ositteiden lukumäärän kertovien muuttujien arvot. Kaksijaksoisilla kuvioilla lasketaan sekä ylempään että alempaan jaksoon kuuluvien ositteiden lukumäärät. Jos eri ositteisiin kuulumisen tulkinnan kanssa on ollut ongelmia eli ylemmän tai alemman jakson ositteiden lukumäärää ei ole laskettu (esim. kaikki ositteet kuuluvat ylempään jaksoon, siis kyseessä on yksijaksoinen kuvio), asetetaan ylempään ja alempaan jaksoon kuuluvien ositteiden lukumääräksi nolla. Tällöin myös kaksijaksoisuudesta kertovan muuttujan arvoksi asetetaan 0 eli kuvio tulkitaan yksijaksoiseksi.

Chain Update upper-storey attribute if problems defining two-storey stand

Käytetään, kun kaksijaksoisuuden määrittelyn kanssa on ollut ongelmia puustoisella kuviolla kasvatuksen jälkeen. Tällöin myös ositetasolla virheellinen tieto kuulumisesta ylempään jaksoon korjataan.

Chain Update development and age class, since thinning and since drainage to comp_unit

Käytetään kehitys- ja ikäluokan sekä harvennuksesta ja ojituksesta kuluneesta ajasta kertovien muuttujien päivitykseen. Lisäksi päivitetään biomassamuutujia. Jos harvennus on toteutettu vähintään kerran ja harvennusta ei ole tehty kuluvalla simulointijaksolla, päivitetään harvennuksesta kuluneen ajan kertova muuttuja. Samoin päivitetään ojituksesta kuluneen ajan kertova muuttuja, jos ojitus on tehty vähintään kerran, mutta ojitusta ei ole tehty kuluvalla simulointijaksolla. Kehitysluokan laskemista varten asetetaan ensin keskiläpimitan arvoksi nolla, jos keskiläpimitta puuttuu. Asetetaan edellisellä simulointijaksolla laskettu kehitysluokkamuuttujan arvo vanha kehitysluokka -muuttujan arvoksi. Jos kuviolla ei ole ositetasoa, on kyseessä avoin uudistusala. Tällöin asetetaan aukeamuuttujan arvoksi 1 ja kehitysluokaksi aukea ala (1). Jos kyseessä on puustoinen kuvio, lasketaan ensin kaksijaksoiselle kuviolle ylempään ja alempaan jaksoon kuuluvien ositteiden keski-iät. Edelleen lasketaan näiden eri jaksojen välinen ikäero kehitysluokkamallia varten. Ikäeroa käytetään tulkittaessa onko kyseessä ylispuustoinen taimikko. Jos kuvio on yksijaksoinen eli ikäeromuuttujalle ei ole laskettu arvoa, asetetaan ikäeromuuttujan arvoksi nolla. Kehitysluokan laskemiseksi määritellään käytettävät uudistamisrajat. Edelleen lasketaan kuvion ikäluokka. Ensin asetetaan kuvion ikä nollaksi, jos se puuttuu. Jos kyseessä on avoin uudistusala, ikäluokka on myös nolla. Tällöin asetetaan myös aukeasta kertovan muuttujan arvoksi 1. Lasketaan puustoisen kuvion ikäluokka. Lopuksi asetetaan hakkuun yhteydessä kuviotasolle tallennettavien biomassatietojen oletusarvoksi 0.

Natural removal

Käytetään luonnonpoistuman ennustamiseen kuviolla vähentämällä kuvauspuun edustamaa runkolukua yksittäisen puun elossapysymistodennäköisyyden perusteella. Tämän jälkeen kuvion runkolukua verrataan edelleen vertailukuvion maksimirunkolukuun. Ylitiheällä kuviolla skaalataan puiden elossapysymistodennäköisyyttä uudelleen ja vähennetään kuvauspuiden edustamaa runkolukua edelleen. Ketjua käytetään myös tietojen päivitykseen luonnonpoistuman laskennan jälkeen. Jättämällä tämän ketjun pois simuloinnista voi luonnonpoistuman ennustamisen kytkeä pois käytöstä. Tällöin pitää myös Calculate tree survival -ketju jättää pois simuloinnista.

Laskentaan käytettävän ketjun osakokonaisuuden valinta riippuu ositteiden määrästä, kuvion jaksoisuudesta, ositteen kuulumisesta ylempään tai alempaan jaksoon sekä ositteiden lukumäärästä ylemmässä ja alemmassa jaksossa. Laskennassa käytetään ketjua ”Adjust number of trees” seuraavissa tapauksissa: Kuviolla on ainoastaan yksi osite. Kaksijaksoisen kuvion ylemmän jakson ositteelle, kun ylemmässä jaksossa on vain yksi osite (esim. ylemmässä jaksossa 1 osite ja alemmassa 2 ositetta). Laskentaan käytetään ketjua ”Adjust number of trees in mixed forest” seuraavissa tapauksissa: Ositteille yksijaksoisella kuviolla, kun ositteita on vähintään kaksi. Kaksijaksoisen vähintään kolme ositetta sisältävän kuvion ylemmän jakson ositteille, jos ositteiden lukumäärä jaksossa on vähintään kaksi. Ositteille, jotka kuuluvat kaksijaksoisen kuvion alempaan jaksoon. Ennen sekapuuston laskentaan tarkoitettua laskentaketjua on laskettava kaikkien kuvion ositteiden puiden maksimilukumäärät. Vasta sen jälkeen voidaan huomioida sekapuustojen vaikutus!

Chain Update stand attributes before scaling

Käytetään päivittämään muuttuneita kuviotietoja. Tilavuus_ennen_slaalausta -muuttujan arvoksi asetetaan kuvion tilavuus luonnonpoistuman laskemista varten. Jos kyseessä ei ole puhdas taimikko, päivitetään kuviolle ennen kasvatusta ollut suurin mahdollinen runkoluku.

Chain Update number of trees

Käytetään kuvauspuun edustaman runkoluvun vähentämiseen elossapysymistodennäköisyysmallin mukaisesti, kun kyseessä ei ole taimikko-osite ja puulle on laskettu elossapysymistodennäköisyys. Lasketaan puulle uusi runkoluku kertomalla runkuloku elossapysymistodennäköisyydellä. Edelleen lasketaan läpimittaluokan pohjapinta-ala.

Chain Update stratum attributes after updating number of trees

Käytetään ositetason muuttujien arvojen päivittämiseen puiden runkoluvun päivityksen jälkeen. Lasketaan ei-taimikko-ositteen keskiläpimitta rinnankorkeudelta, keskipituus, runkoluku, pohjapinta-ala, suhteellinen tiheys ja tilavuus. Lasketaan taimikko-ositteelle pohjapinta-ala, keskipituus- ja -läpimitta. Sekä taimikko- että ei-taimikko-ositteelle lasketaan vielä kannonkorkeusläpimitta. Lopuksi lasketaan ositteen (myös puhtaan taimikko-ositteen) valtapituus ja painomuuttuja (runkoluku * keskipituus) taimikkokuvion keski-iän määrittämistä varten.

Chain Update stand attributes after updating number of trees

Käytetään kuviotason muuttujien arvojen päivittämiseen puiden runkoluvun päivityksen jälkeen. Asetetaan luonnonpoistumamuuttujan arvoksi nolla ennen uuden arvon laskentaa. Päivitetään ei-taimikkokuvion pohjapinta-ala,

keskiläpimitta, keskipituus, tilavuus, valtapuiden latvussuhde ja läpimitta, runkoluku, valtapituus sekä biologinen keski-ikä. Taimikkokuviolle, jolle puut on jo olemassa (saatu suoraan datasta tai luotu jakaumamalleilla) pohjapinta-ala, runkoluku, keski-ikä, keskiläpimitta, keskipituus, valtapituus, tilavuus sekä valtapuiden latvussuhde ja läpimitta.

Chain Calculate maximum stem number

Käytetään päivitettäessä ositteen muuttujien arvoja kasvatuksen jälkeen, kun ositteella on jo olemassa puut (saatu suoraan datasta tai luotu jakaumamalleilla). Ensin lasketaan kannonkorkeusläpimitta ja kasvupaikkaindeksi. Tämän jälkeen lasketaan suurin sallittu runkoluku.

Chain Calculate stand attributes

Käytetään muuttujien arvojen päivittämiseen kuviotasolla. Asetetaan runkoluvun maksimi ja suhteellinen tiheys puulajeittain nollaksi ennen muuttujien arvojen laskentaa. Lasketaan runkoluvun maksimi kuviolla puulajeittain. Edelleen kaksijaksoiselle kuviolle lasketaan maksimi runkoluku puulajeittain sekä ylemmälle että alemmalle jaksolle. Lasketaan puulajeittaiset runkoluvut. Edelleen kaksijaksoisella kuviolla lasketaan ylemmän jakson runkoluvut puulajeittain.

Chain Calculate relative density factor to stand level

Käytetään suhteellisen tiheyden päivittämiseen kuviotasolla. Kun kyseessä ei ole puhdas taimikkokuvio lasketaan suhteellinen tiheys puulajiettain. Lisäksi lasketaan suhteellinen tiheys kaksijaksoisen kuvion ylemmälle jaksolle kokonaisuudessaan (kaikki puulajit yhdessä) ja puulajeittain.

Chain Adjust number of trees

Käytetään määrittämään puuston kuolleisuus siten, että puiden maksimilukumäärä kuviolla ei ylity. Ensin asetetaan skaalauskertoimen oletusarvoksi yksi. Lasketaan ositteelle skaalauskerroin, kun kyseessä ei ole taimikko eikä siemenpuuosite. Kerralla poistettavien puiden määrän rajoittamiseksi lasketaan suurin sallittu runkoluvun vähennys. Tarkastetaan ylittyykö asetettu runkoluvun vähennys. Jos ylittyy, suurimmaksi sallituksi runkoluvuksi tulee todellinen runkoluku vähennettynä suurimmalla sallitulla runkoluvun vähennyksellä. Skaalauskertoimen laskutapa määräytyy ositteen runkoluvun mukaan. Normaalitiheyksisessä puustossa skaalauskerroin lasketaan suoraan puuston suurimman sallitun ja todellisen runkoluvun suhteena. Tiheämmässä puustossa laskenta on monimutkaisempi.

Chain Calculate maximum number of trees in mixed forest

Käytetään suurimman sallitun runkoluvun laskentaan sekapuustoisella kuviolla, kun kyseessä ei ole taimikko- tai siemenpuuosite. Jos ositteiden lukumäärä kuviolla on vähintään kaksi, lasketaan suurin sallittu runkoluku.

Chain Update stand attributes in mixed forest

Käytetään muuttujien arvojen päivittämiseen kuviotasolla sekapuustoisella kuviolla. Lasketaan runkoluvun maksimi sekapuukuviolla puulajeittain. Edelleen kaksijaksoiselle sekapuukuviolle lasketaan maksimi runkoluku puulajeittain sekä ylemmälle että alemmalle jaksolle.

Chain Adjust number of trees in mixed forest

Käytetään määrittämään puuston kuolleisuus siten, että puiden maksimilukumäärä sekapuukuviolla ei ylity. Käytetään lähtötietona edellisessä ketjussa laskettua puiden maksimimäärää. Laskenta tehdään, kun kyseessä ei ole taimikko eikä siemenpuuosite. Lasketaan skaalauskerroin sekapuustoisen kuvion ositteelle. Sekapuuston puulajisuhteet vaikuttavat sallittuun puuston määrään suhteellisen tiheysindeksin painolla. Kerralla poistettavien puiden määrän rajoittamiseksi lasketaan suurin sallittu runkoluvun vähennys. Tarkastetaan ylittyykö asetettu runkoluvun vähennys. Jos ylittyy, suurimmaksi sallituksi runkoluvuksi tulee todellinen runkoluku vähennettynä suurimmalla sallitulla runkoluvun vähennyksellä. Skaalauskertoimen laskutapa määräytyy ositteen runkoluvun mukaan. Normaalitiheyksisessä puustossa skaalauskerroin lasketaan suoraan puuston suurimman sallitun ja todellisen runkoluvun suhteena. Tiheämmässä puustossa laskenta on monimutkaisempi.

Chain Update tree's probability to die

Käytetään toteuttamaan luonnonpoistuma kuviolla kuvauspuiden runkolukua skaalaamalla, kun edellä tehty runkoluvun päivitys elossapysymistodennäköisyysmallin perusteella ei ole riittänyt pudottamaan kuvion runkolukua riittävästi. Jos suurin sallittu runkoluku on ylittynyt, on skaaluskerroin saanut edellisissä malliketjuissa pienemmän arvon kuin yksi. Tällöin skaalataan ei-taimikko-ositteen puuston elossapysymistodennäköisyys siten, että

puiden maksimilukumäärä kuviolla ei ylity. Skaalauksen jälkeen päivitetään elossa olevien (kuvauspuun edustamien) puiden lukumäärä. Näin saavutetaan sallittu runkoluku. Lisäksi lasketaan läpimittaluokan pohjapinta-ala.

Chain Update stratum attributes after scaling

Käytetään päivittämään ositetietoja seuraavan jakson kasvua, hakkuita ja metsänhoitotoimenpiteitä varten, jos skaalusta on käytetty. Asetetaan ensin arvot_skaalattu -muuttujan oletusarvoksi 0. Jos suurin sallittu runkoluku on ylittynyt eli skaalauskerroin on saanut edellisissä malliketjuissa pienemmän arvon kuin yksi, asetetaan arvot_skaalattu -muuttujan arvoksi 1. Päivitetään ositteen, jolle puut on jo olemassa (saatu suoraan datasta tai luotu jakaumamalleilla) tietoja. Lasketaan ei-taimikko-ositteen keskiläpimitta rinnankorkeudelta, keskipituus, runkoluku, pohjapinta-ala, suhteellinen tiheys ja tilavuus. Lasketaan taimikko-ositteelle pohjapinta-ala, keskipituus- ja -läpimitta. Sekä taimikko- että ei-taimikko-ositteelle lasketaan vielä kannonkorkeusläpimitta. Lopuksi lasketaan ositteen (myös puhtaan taimikko-ositteen) valtapituus ja painomuuttuja (runkoluku * keskipituus) taimikkokuvion keski-iän määrittämistä varten.

Chain Update stand attributes after scaling

Käytetään päivittämään muuttuneita kuviotietoja seuraavan jakson kasvua, hakkuita ja metsänhoitotoimenpiteitä varten. Asetetaan ensin arvot_skaalattu -muuttujan oletusarvoksi 0. Summataan arvot_skaalattu -muuttujan arvot ositteilta. Jos kuvion yhdelläkin ositteella on arvoja skaalattu, asetetaan arvot_skaalattu -muuttujan arvoksi 1. Kuviolla, jolla runkoluku on skaalattu päivitetään ei-taimikkokuvion pohjapinta-ala, keskiläpimitta, keskipituus, tilavuus, valtapuiden latvussuhde ja läpimitta, runkoluku, valtapituus sekä biologinen keski-ikä. Taimikkokuviolle, jolle puut on jo olemassa (saatu suoraan datasta tai luotu jakaumamalleilla) pohjapinta-ala, runkoluku, keski-ikä, keskiläpimitta, keskipituus, valtapituus, tilavuus sekä valtapuiden latvussuhde ja läpimitta. Lopuksi lasketaan toteutunut luonnonpoistuma kuviolle.

Calculate biomass

Käytetään kuvion puuston biomassan määrittämiseen. Biomassojen laskennan saa kytkettyä pois päältä jättämällä tämän ketjun pois simuloinnista.

Chain Calculate biomass of tree

Käytetään puun eri osien biomassan määrittämiseen. Jos kyseessä on ei-taimikko-osite, lasketaan puun eri osien biomassat.

Calculate stand value growth

Käytetään kuvion puuston arvon määrittämiseen ja sitä kautta arvokasvun laskemiseen. Tuottaa samalla myös puutavaralajijakauman mukaanlukien oksista ja latvoista sekä kannoista ja juurista kertyvän bioenergian. Tätä ketjua voidaan käyttää myös aineiston täydennysketjujen yhteydessä ilman puuston kasvatusta. Arvokasvun laskennan saa kytkettyä pois päältä jättämällä tämän ketjun pois simuloinnista.

Chain Calculate the value of growing stock on stratum level

Käytetään ositteen puuston arvon sekä puustavaralajijakauman ja bioenergian määrän laskemiseen. Jos kuvion kehitysluokka on jokin muu kuin aukea uudistusala tai nuori taimikko ja kyseessä ei ole taimikko-osite, lasketaan ositteen puuston arvo ja puutavaralajijakauma sekä bioenergiapuun määrä.

Chain Calculate the stand value growth

Käytetään kuvion puuston arvokasvun määrittämiseen. Aluksi asetetaan kuvion arvokasvu, suhteellinen arvokasvu, keskimääräinen suhteellinen arvokasvu sekä keskimääräinen arvokasvu nollaksi ja edellisellä laskentajaksolla laskettu puuston arvo puuston arvo ennen kasvatusta -muuttujan arvoksi. Jos kuvion kehitysluokka on varttunut taimikko, nuori- tai varttunut kasvatusmetsikkö tai uudistuskypsä metsikkö, päivitetään laskentahetken puuston arvo summaamalla ositteiden puustojen arvot yhteen. Lopuksi lasketaan absosuuttinen, suhteellinen ja viiden vuoden keskimääräinen arvokasvu.

Calculate stand productive value

Käytetään kuvion tuottoarvon laskentaan. Tätä ketjua voidaan käyttää myös aineiston täydennysketjujen yhteydessä ilman puuston kasvatusta. Tuottoarvon laskennan saa kytkettyä pois päältä jättämällä tämän ketjun pois simuloinnista.

Chain Calculate productive value

Käytetään kuvion maan ja puuston tuottoarvon laskentaan. Kun kyseessä on puustoinen kuvio ja luontaisen uudistamisen sisäänkasvuvaihe ei ole käynnissä, nollataan puulajeittaiset pohjapinta-alat ja runkoluvut kuviolla kertovien muuttujien arvot. Tämän jälkeen lasketaan näille muuttujille uudet arvot. Tuottoarvo lasketaan puulajeittaisilla pohjapinta-aloilla painottaen. Asetetaan taimikon tuottoarvon kertovan muutujan oletusarvoksi nolla. Lasketaan taimikon ja varttuneen metsän puuston tuottoarvo sekä paljaan maan tuottoarvo. Mineraali ja turvemaalle laskenta tehdään samalla mallilla, mutta turvemaan tuottoarvoa on mahdollista laskea kertoimella. Puuttoman kuvion puuston tuottoarvoksi asetetaan nolla.

Calculate v-value

Käytetään kuvion puuston v-arvon määrittämiseen. V-arvon laskennan saa kytkettyä pois päältä jättämällä tämän ketjun pois simuloinnista.

Chain Calculate V_value

Käytetään kuvion v-arvon määrittämiseen, kun kuvion kehitysluokka on jokin muu kuin aukea uudistusala tai nuori taimikko ja kuvion puuston arvo on tiedossa edelliseltä laskentavuodelta. Ennen laskentaa asetetaan v-arvon oletusarvoksi nolla.

9.1.5 Toimenpideketjut

Harvests branching

Käytetään hakkuiden toteuttamiseen. Optimointia varten simuloidaan useita erilaisia käsittelyvaihtoehtoja. Erilaisia käsittelyvaihtoehtoja voivat muodostaa esim. viivästetyt harvennus- tai päätehakkuut, harvennusvoimakkuus tai kokonaan toisenlainen käsittelyketju. Seuraavassa on kuvattu tavanomainen optimointiin tähtäävä haaroitettu simulointi. Yleisesti ottaen kaikki halutut käsittelyvaihtoehdot on määriteltävä malliketjuissa. Tällä tavalla vältytään turhien toteuttamiskelvottomien vaihtoehtojen simuloinnilta ja jatkokäsittelyltä optimoinnissa. Näin optimointiin tuotetaan vain todella toteutettavissa olevia käsittelyketjuja.

Chain Final harvests

Käytetään päätehakkuiden toteuttamiseen simuloinnissa, kun kyseessä on puustoinen kuvio, mutta ei puhdas taimikko. Ensin lasketaan keinollisesti syntyneen kuvion keski-ikä, jos taimikonhoitoa tai harvennusta ei ole vielä tehty. Edelleen, jos kyseessä ei ole siemenpuu- tai kaistalehakkuukuvio, lasketaan ensin uudistamisrajat ja tulkitaan tämän jälkeen, onko kuvio uudistuskypsä. Uudistamisrajat voidaan määrittää Metsätalouden kehittämiskeskus tapion uudistussuositusten keskiläpimitta- ja ikätunnuksen ala- ja ylärajan välille. Jos kuvio on uudistuskypsä, asetetaan uudistuskypsyydestä kertovan muuttujan arvoksi 1. Päivitetään muuttujan arvo, joka kertoo kuinka monella vuodella uudistamisraja on ylitetty. Tämän muuttujan avulla toteutetaan viivästetty uudistushakkuu optimointia varten. Jos uudistamiskypsyydestä kertovalla muuttujalla ei ole vielä arvoa tai, jos pääpuulaji on vaihtunut kasvatusjakson aikana, asetetaan arvoksi 0. Simuloinnin haaroitus päätehakkuussa on kuvattu heti ketjun alussa: viivästetty päätehakkuu tehdään, kun uudistuskypsyyden saavuttamisesta on kulunut tietty aika esim. 5, 10, 15, 20 ja 30 vuotta. Näin ollen haaroja syntyy kuusi kappalta. Laskenta menee kaikilla haaroilla samalla tavalla. Päätehakkuu tehdään, jos kuvio on metsämaata, uudistushakkuukelpoinen, valtapituus on riittävä suhteessa kasvupaikkaan ja edellisestä hakkuusta on kulunut riittävästi aikaa. Päätehakkuu tehdään avohakkuuna, jos kyseessä on rehevä (MT+) kivennäismaan kuvio pääpuulajista riippumatta tai vähäravinteinen (VT-) pääpuulajiltaan muu kuin mäntykuvio (todennäköisesti kasvupaikalle sopimaton puulaji). Kun hakkuu on tehty, uudistuskypsästä kuviosta kertovan muuttujan arvoksi asetetaan 0. Turvemaiden reheville (MT+) korpi tai lettokuusikoille tehdään avohakkuu. Avohakkuun jälkeen asetetaan uudistuskypsästä kuviosta kertovan muuttujan arvoksi 0. Kivennäismaiden vähäravinteisille (VT-) kuviolle ja turvemaan keskiravinteisille ja huonommille (MT-) räme- ja nevakuviolle tehdään siemenpuuhakkuu, kun pääpuulaji on mänty. Uudistettavaksi puulajiksi asetetaan

mänty (1). Kun hakkuu on tehty, asetetaan uudistushakkuukypsästä kuviosta kertovan muuttujan arvoksi 0 ja määritetään uudistettava puulaji. Jos edelleenkään uudistuskypsää turvemaakuviota ei ole uudistushakattu epätavallisen pääpuulaji/kasvupaikkayhdistelmän vuoksi, kuvio avohakataan. Lopuksi vielä asetetaan uudistuskypsä -muuttujan arvoksi 0. Tukkivähennystä voidaan malleissa käyttää kuvaamaan laatuvikojen vuoksi tukkipuutavaralajin siirtymistä kuiduksi. vaihtoehtoina on käyttää puutasolla määritettyä tai/ja alueellista tukin tasokorjausta.

Chain Update comp_unit after clearcut or stripcut

Käytetään kuviotietojen päivittämiseen avo- tai kaistalehakkuun jälkeen, kun kyseessä on puustoinen kuvio. Samalla simulointijaksolla avohakkuun kanssa nollataan kuviontason muuttujien arvoja. Poistetaan vielä kuviolle mahdollisesti jääneet jättöpuut. Edelleen asetetaan aukea-muuttujan arvoksi 1, kehitysluokaksi avoin_uudistusala ja kuvion puuston arvoksi nolla. Lopuksi, jos hakkuu tehtiin kaistalehakkuuna, tulee uudistamistavaksi luontainen uudistaminen.

Chain Update stratum after seedtree or sheltertree cut

Käytetään ositetietojen päivittämiseen siemen- tai suojuspuuhakkuun jälkeen, kun kyseessä on puustoinen kuvio. Jos siemen- tai suojuspuuhakkuu on tehty samalla simulointijaksolla, päivitetään ositteen pohjapinta-ala, keskiläpimitta, keskipituus, runkoluku, tilavuus, kuvauspuiden lukumäärä ja ositteen puuston arvo, puutavaralajijakauma sekä bioenergiapuun määrä. Lopuksi asetetaan siemenpuuositteesta kertovan muuttujan arvoksi 1 ja ylemmästä jaksosta kertovan muuttujan arvoksi 0.

Chain Update comp_unit after seedtree or sheltertree cut

Käytetään kuviotietojen päivittämiseen siemen- tai suojuspuuhakkuun jälkeen, kun kyseessä on puustoinen kuvio. Jos siemen- tai suojuspuuhakkuun on tehty samalla simulointijaksolla, päivitetään kuvion pääpuulaji, pohjapinta-ala, tilavuus, keskiläpimitta, keskipituus, runkoluku, koivun osuus pohjapinta-alasta, valtapuiden keskiläpimitta ja latvussuhde. Edelleen asetetaan siemenpuukuviomuuttujan arvoksi 1, nollataan kuviotason muuttujien arvoja sekä asetetaan taimettomasta siemenpuukuviosta kertovan muuttujan arvoksi 1. Uudistamistavaksi tulee luontainen uudistaminen. Edelleen päivitetään kuvion kehitysluokka. Sitä varten asetetaan ennen toimenpiteen toteutusta ollut kehitysluokka vanha_kehitysluokka -muuttujan arvoksi, eri jaksojen ikäeroksi 0 ja lasketaan uudistamisrajat. Lopuksi päivitetään kuvion puuston arvo.

Chain Remove seed trees

Käytetään siemenpuiden poistoon simuloinnissa, kun kyseessä on puustoinen kuvio. Sekä männyn siemenpuuettä kuusen suojuspuukuvioilta poistetaan siemen- ja suojuspuut, kun uudistushakkuusta on kulunut tietty aika. Jos simulointiaineistossa on alussa siemenpuukuviota, joilla on jo taimiainesta, siemenpuut poistetaan, kun kuviolla on taimiaineista tietty määrä.

Chain Update comp_unit after removing seed trees

Käytetään kuviotietojen päivittämiseen siemenpuiden poiston jälkeen, kun kyseessä on puustoinen kuvio. Samalla simulointijaksolla siemenpuiden poiston kanssa asetetaan ensin puhtaasta taimikkokuviosta kertovan muuttujan oletusarvoksi 1. Jos kuitenkin kuviolla on muitakin kuin taimikko-ositteita, asetetaan puhdas taimikko -muuttujan arvoksi 0. Siemenpuukuvio -muuttujan arvoksi asetetaan 0. Päivitetään kuvion runkoluku, odotettavissa oleva runkoluku, ikä, ositteiden lukumäärä ja aritmeettinen keskipituus. Edelleen nollataan kuviotason muuttujien arvoja ja päivitetään taimikkokuviosta kertovan muuttujan arvo. Jos siemenpuiden poiston jälkeen kuviolla on puhdas taimikko, päivitetään pääpuulaji, keski- ja valtapituus sekä nollataan kuviotason muuttujien arvoja. Jos jäljelle jäävä puusto ei ole puhdas taimikko, päivitetään pohjapinta-ala, pääpuulaji, tilavuus, keskiläpimitta, keski- ja valtapituus, koivun osuus pohjapinta-alasta, kuvauspuiden lukumäärä, suhteellinen tiheys sekä valtapuiden keskiläpimitta ja latvussuhde. Edelleen päivitetään kuvion kehitysluokka. Sitä varten asetetaan ennen toimenpiteen toteutusta ollut kehitysluokka vanha_kehitysluokka -muuttujan arvoksi, eri jaksojen ikäeroksi 0 ja lasketaan uudistamisrajat. Lopuksi asetetaan kuvion puuston arvoksi nolla.

Chain Thinnings

Käytetään harvennusten toteuttamiseen simuloitaville kuvioille, kun kyseessä on puustoinen kuvio ja uudistushakkuun viisästys ei ole toiminnassa. Harvennusmallia varten lasketaan leimaus- ja jäävän puuston rajat, kun kyseessä ei ole puhdas taimikko ja puuston valtapituus on riittävä suhteessa kasvupaikkaan. Harvennusmallin valinta riippuu maantieteellisestä sijainnista (Etelä-, Väli- ja Pohjois-Suomi), kasvupaikasta ja pääpuulajista. Harvennuksen leimausrajaa ja jäävän puuston rajaa on mahdollista säätää Metsätalouden kehittämiskeskus tapion uusien harvennusmallien vaihteluvälin sisällä. Jos kuvion pohjapinta-ala on suurempi kuin harvennusmallin leimausraja, päivitetään muuttuja, joka kertoo kuinka monta vuotta on kulunut harvennusrajan ylittymisestä. Tämän muuttujan

avulla toteutetaan viivästetty harvennus optimointia varten. Simuloinnin haaroitus harvennushakkuissa: Seuraavassa on kuvattu malliketjun osat, joissa harvennushakkuuta 5 tai 10 vuotta viivästyttämällä tuotetaan erilaisia vaihtoehtoja simulointiin. 5 vuotta viivästetty harvennushakkuu tehdään, kun harvennusmallin leimausrajan ylittymisestä on kulunut 5 vuotta. Vastavasti 10 vuotta viivästetty harvennushakkuu tehdään, kun harvennusmallin leimausrajan ylittymisestä on kulunut 10 vuotta. Näin ollen haaroja syntyy 3 kappaletta. Bioenergiamallia varten asetetaan kantojen_keruu -muuttujan arvoksi 0 eli estetään kantojen nosto harvennuksessa. Ennen varsinaista harvennusta asetetaan juuri harvennetusta kuviosta kertovan muuttujan arvoksi 0. Jos kuviota ei ole vielä harvennettu tai jos harvennuksista ei ole tietoa ja puusto on vielä ensiharvennuskokoista, toteutetaan ensiharvennus. Toteutetun harvennuksen jälkeen asetetaan juuri harvennettu -muuttujan arvoksi 1. Edelleen, jos puusto ei ollut ensiharvennuskelpoinen, harvennus toteutetaan muuna harvennuksena. Toteutetun harvennuksen jälkeen asetetaan juuri harvennettu -muuttujan arvoksi 1. Puuston ensiharvennus-/harvennuskelpoisuus tulkitaan valtapituuden ja kasvupaikan mukaan.

Chain Update tree after thinning

Käytetään päivitettäessä puutason muuttujia harvennuksen jälkeen, kun kyseessä on puustoinen kuvio. Samalla simulointijaksolla harvennuksen kanssa päivitetään läpimittaluokan pohjapinta-ala.

Chain Update stratum after thinning

Käytetään päivitettäessä ositetason muuttujia harvennuksen jälkeen, kun kyseessä on puustoinen kuvio. Samalla simulointijaksolla harvennuksen kanssa päivitetään ositteen pohjapinta-ala, keskiläpimitta, keskipituus, valtapituuden kasvu, runkoluku, kuvauspuiden lukumäärä sekä tilavuus. Lopuksi lasketaan harvennuksen jälkeinen puuston arvo, puutavaralajijakauma sekä bioenergiapuun määrä.

Chain Update comp_unit after thinning

Käytetään päivitettäessä kuviotason muuttujia harvennuksen jälkeen, kun kyseessä on puustoinen kuvio. Samalla simulointijaksolla harvennuksen kanssa päivitetään kuvion pääpuulaji, pohjapinta-ala, tilavuus, keskiläpimitta, keski- ja valtapituus, runkoluku, odotettu runkoluku, koivun osuus pohjapinta-alasta, kuvauspuiden lukumäärä sekä valtapuiden keskiläpimitta ja latvussuhde. Edelleen nollataan harvennusrajan ylittämisestä kuluneen ajan, aika harvennuksesta ja kaksijaksoisuudesta kertovien muuttujien arvot sekä lasketaan kuvion puuston arvo. Lopuksi bioenergiamallia varten asetetaan kantojen_keruu -muuttujan oletusarvoksi 1. Jos kohde ei sovellu kantojen nostoon, asetetaan kantojen_keruu -muuttujan arvoksi 0.

Silvicultural operations

Käytetään metsänhoitotoimenpiteiden toteuttamiseen.

Chain Soil preparations after regeneration harvest

Käytetään maanmuokkauksen kustannusten kohdentamiseen uudistushakatuille kuvioille sinä vuonna, kun uudistushakkuu on tehty. Laikutus tehdään kivennäismaiden reheville (MT+) kuviolle ja yhdelle kolmasosalle turvemaiden keski- ja vähäravinteisista (MT-CT) männyn siemenpuukuvioista. Laikkumätästys tehdään turvemaiden reheville (MT+) avohakkuukuvioille. Äestys tehdään vähäravinteisille (VT-) kivennäismaakuvioille. Toteutetun maanmuokkauksen jälkeen maanmuokkauksesta kertovan muuttujan arvoksi asetetaan 1.

Chain Tending of young stands

Käytetään taimikonhoidon toteuttamiseen tietyssä puuston pituus- ja ikävaiheessa, kun kyseessä on puustoinen kuvio. Malliketju sisältää sekä varhais- että varsinainen taimikonhoidon toteutuksen. Varhaistaimikonhoito (heinäys ja perkaus) toteutetaan varhaisessa pituusvaiheessa kuusen ja männyn taimikoille. Lasketaan taimikonhoitoraja eli tiheys, jota tiheämmät taimikot harvennetaan. Varsinainen taimikonhoito tehdään, jos taimikon runkoluku on suurempi kuin taimikonhoitoraja. Taimikonhoito voidaan uusia tietyn ajan kuluttua edellisestä taimikonhoidosta. Kuitenkin kasvupaikalle on määritelty taimikonhoitokertojen maksimiarvo, jota ei voi ylittää. Taimikonhoidon toteutusvaihe määräytyy kasvupaikan, pääpuulajin ja puuston pituuden mukaan.

Chain Update stratum after tending of young stand

Käytetään päivitettäessä ositetietoja taimikonhoidon jälkeen samalla simulointijaksolla taimikonhoidon kanssa, kun kyseessä on puustoinen kuvio. Jos ositteen pohjapinta-ala on suurempi kuin nolla, päivitetään pohjapinta-ala, keskiläpimitta, keskipituus, runkoluku, tilavuus ja kuvauspuiden lukumäärä. Lopuksi lasketaan alle 1,3 m pitkien

puiden lukumäärä. Jos alle 1,3 m puiden lukumäärä on nolla, asetetaan taimikko-ositteesta kertovan muuttujan arvoksi 0.

Chain Update comp_unit after tending of young stand

Käytetään päivitettäessä kuviotietoja taimikonhoidon jälkeen samalla simulointijaksolla taimikonhoidon kanssa, kun kyseessä on puustoinen kuvio. Jos puutaso on olemassa, päivitetään kuvion pääpuulaji, pohjapinta-ala, tilavuus, keskiläpimitta, keskipituus, runkoluku, odotettu runkoluku, ositteiden lukumäärä, koivun osuus pohjapinta-alasta, kuvauspuiden lukumäärä, valtapuiden keskiläpimitta ja latvussuhde. Edelleen lasketaan taimikko-ositteiden lukumäärä. Jos kuviolla ei ole yhtään taimikko-ositetta, asetetaan taimikkokuviosta kertovan muuttujan arvoksi 0. Jos puutaso puuttuu, päivitetään keskipituus, runkoluku, odotettu runkoluku ja ositteiden lukumäärä. Tulkitaan ositteiden tietoja käyttäen onko kuvio kaksijaksoinen. Jos kuvio on yksijaksoinen, nollataan ylempään ja alempaan jaksoon kuuluvien ositteiden lukumäärät. Jos kuvio on kaksijaksoinen, lasketaan sekä ylempään että alempaan jaksoon kuuluvien ositteiden lukumäärät. Jos eri osiitteisiin kuulumisen tulkinnan kanssa on ollut ongelmia eli ylemmän tai alemman jakson ositteiden lukumäärää ei ole laskettu (esim. kaikki ositteet kuuluvat ylempään jaksoon, siis kyseessä on yksijaksoinen kuvio), asetetaan ylempään ja alempaan jaksoon kuuluvien ositteiden lukumääräksi nolla. Tällöin myös kaksijaksoisuudesta kertovan muuttujan arvoksi asetetaan 0 eli kuvio tulkitaan yksijaksoiseksi.

Forced operations

Käytetään toimenpiteiden pakottamiseen puuston kasvun simuloinnin yhteydessä tai ennen sitä. Sisältää kaikki järjestelmällä toteutettavat toimenpiteet, niin hakkuut kuin hoitotoimenpiteetkin. Pakotetut toimenpiteet saadaan metsätaloussuunnitelman toimenpide-ehdotuksista. Jokaista toimenpidemallia varten on oma malliketjukokonaisuutensa.

Forced thinnings

Chain Calculate thinning limits

Käytetään harvennuksen leimausrajan ja harvennuksen jälkeen jäävän puuston ala- ja ylärajan määrittämiseen.

Chain Forced first thinning

Käytetään pakotetun ensiharvennuksen toteuttamiseen. Harvennusmallin valinta riippuu maantieteellisestä sijainnista (Etelä-, Väli- ja Pohjois-Suomi) ja pääpuulajista. Ennen varsinaista harvennusta asetetaan juuri harvennetusta kuviosta kertovan muuttujan arvoksi 0 ja bioenergian keruumallia varten kantojen_keruu -muuttujan arvoksi 0. Toteutetun harvennuksen jälkeen asetetaan juuri harvennettu -muuttujan arvoksi 1.

Chain Forced low thinning

Käytetään pakotetun harvennuksen toteuttamiseen alaharvennuksena. Harvennusmallin valinta riippuu maantieteellisestä sijainnista (Etelä-, Väli- ja Pohjois-Suomi) ja pääpuulajista. Ennen varsinaista harvennusta asetetaan juuri harvennetusta kuviosta kertovan muuttujan arvoksi 0 ja bioenergian keruumallia varten kantojen_keruu -muuttujan arvoksi 0. Toteutetun harvennuksen jälkeen asetetaan juuri harvennettu -muuttujan arvoksi 1.

Chain Forced low thinning with removed volume

Käytetään pakotetun harvennuksen toteuttamiseen alaharvennuksena annetulla harvennuspoistumalla. Harvennusmallin valinta riippuu maantieteellisestä sijainnista (Etelä-, Väli- ja Pohjois-Suomi) ja pääpuulajista. Ennen varsinaista harvennusta asetetaan juuri harvennetusta kuviosta kertovan muuttujan arvoksi 0 ja bioenergian keruumallia varten kantojen_keruu -muuttujan arvoksi 0. Toteutetun harvennuksen jälkeen asetetaan juuri harvennettu -muuttujan arvoksi 1.

Chain Update tree after forced thinning

Käytetään päivitettäessä puutason muuttujia harvennuksen jälkeen. Samalla simulointijaksolla harvennuksen kanssa päivitetään läpimittaluokan pohjapinta-ala.

Chain Update stratum after forced thinning

Käytetään päivitettäessä ositetason muuttujia harvennuksen jälkeen. Samalla simulointijaksolla harvennuksen kanssa päivitetään ositteen pohjapinta-ala, keskiläpimitta, keskipituus, valtapituuden kasvu, runkoluku, kuvauspuiden lukumäärä sekä tilavuus. Lopuksi lasketaan harvennuksen jälkeinen puuston arvo, puutavaralajijakauma sekä bioenergiapuun määrä.

Chain Update comp_unit after forced thinning

Käytetään päivitettäessä kuviotason muuttujia harvennuksen jälkeen. Samalla simulointijaksolla harvennuksen kanssa päivitetään kuvion pääpuulaji, pohjapinta-ala, tilavuus, keskiläpimitta, keski- ja valtapituus, valtapituus, runkoluku, odotettu runkoluku, koivun osuus pohjapinta-alasta, kuvauspuiden lukumäärä sekä valtapuiden keskiläpimitta ja latvussuhde. Lopuksi nollataan harvennusrajan ylittämisestä kuluneen ajan, aika harvennuksesta ja kaksijaksoisuudesta kertovan muuttujan arvot sekä lasketaan kuvion puuston arvo.

Chain Calculate productive value

Käytetään kuvion maan ja puuston tuottoarvon laskentaan. Kun kyseessä on puustoinen kuvio, nollataan puulajeittaiset pohjapinta-alat ja runkoluvut kuviolla kertovien muuttujien arvot. Tämän jälkeen lasketaan näille muuttujille uudet arvot. Tuottoarvo lasketaan puulajeittaisilla pohjapinta-aloilla painottaen. Asetetaan taimikon tuottoarvon kertovan muutujan oletusarvoksi nolla. Lasketaan taimikon ja varttuneen metsän puuston tuottoarvo sekä paljaan maan tuottoarvo. Mineraali ja turvemaalle laskenta tehdään samalla mallilla, mutta turvemaan tuottoarvoa on mahdollista laskea kertoimella. Puuttoman kuvion puuston tuottoarvoksi asetetaan nolla.

Forced clearcut and strip cut

Chain Forced clearcut

Käytetään pakotetun avohakkuun toteuttamiseen.

Chain Forced stripcut

Käytetään pakotetun kaistalehakkuun toteuttamiseen. Asetetaan uudistettavaksi puulajiksi kuusi (2). Kaistalehakkuun toteuttamisen jälkeen asetetaan kaistalehakkuukuviosta ja taimettomasta luontaisen uudistamisen kuviosta kertovien muuttujien arvoksi 1.

Chain Update comp_unit after forced clearcut or strip cut

Käytetään kuviotietojen päivittämiseen pakotetun avo- tai kaistalehakkuun jälkeen. Samalla simulointijaksolla avohakkuun kanssa nollataan kuviontason muuttujien arvoja. Edelleen asetetaan aukea-muuttujan arvoksi 1, kehitysluokaksi avoin_uudistusala ja kuvion puuston arvoksi nolla. Poistetaan vielä kuviolle mahdollisesti jääneet jättöpuut.

Chain Calculate productive value

Käytetään kuvion maan ja puuston tuottoarvon laskentaan. Kun kyseessä on puustoinen kuvio, nollataan puulajeittaiset pohjapinta-alat ja runkoluvut kuviolla kertovien muuttujien arvot. Tämän jälkeen lasketaan näille muuttujille uudet arvot. Tuottoarvo lasketaan puulajeittaisilla pohjapinta-aloilla painottaen. Asetetaan taimikon tuottoarvon kertovan muutujan oletusarvoksi nolla. Lasketaan taimikon ja varttuneen metsän puuston tuottoarvo sekä paljaan maan tuottoarvo. Mineraali ja turvemaalle laskenta tehdään samalla mallilla, mutta turvemaan tuottoarvoa on mahdollista laskea kertoimella. Puuttoman kuvion puuston tuottoarvoksi asetetaan nolla.

Forced seedtree position, sheltertree, covertree position and seed/sheltertree removal

Chain Forced seedtree position

Käytetään pakotetun siemenpuuhakkuun toteuttamiseen. Ennen siemenpuuhakkuun totetuttamista lasketaan valtapuiden keskiläpimitta ja asetetaan uudistettavaksi puulajiksi mänty (1). Siemenpuuhakkuun jälkeen asetetaan kuvion kehitysluokaksi siemenpuumetsikkö (7).

Chain Forced sheltertree position

Käytetään pakotetun suojuspuuhakkuun toteuttamiseen. Ennen suojuspuuhakkuun totetuttamista lasketaan valtapuiden keskiläpimitta ja asetetaan uudistettavaksi puulajiksi kuusi (2). Suojuspuuhakkuun jälkeen asetetaan kuvion kehitysluokaksi suojuspuumetsikkö (8).

Chain Forced covertree position

Käytetään pakotetun verhopuuhakkuun toteuttamiseen. Ennen verhopuuhakkuun totetuttamista lasketaan valtapuiden keskiläpimitta ja asetetaan uudistettavaksi puulajiksi kuusi (2). Verhopuuhakkuun jälkeen asetetaan kuvion kehitysluokaksi suojuspuumetsikkö (8).

Chain Update stratum after forced seed, shelter or covertree cut

Käytetään ositetietojen päivittämiseen siemen-, suojus- tai verhopuuhakkuun jälkeen, kun kyseessä on puustoinen kuvio. Jos siemen-, suojus- tai verhopuuhakkuu on tehty samalla simulointijaksolla, päivitetään ositteen

pohjapinta-ala, keskiläpimitta, keskipituus, runkoluku, tilavuus, kuvauspuiden lukumäärä ja ositteen puuston arvo, puutavaralajijakauma sekä bioenergiapuun määrä. Lopuksi asetetaan siemenpuuositteesta kertovan muuttujan arvoksi 1 sekä nollataan ositteen puuston arvo ja asetetaan ylemmästä jaksosta kertovan muuttujan arvoksi 0.

Chain Update comp_unit after forced seed, shelter or covertree cut

Käytetään kuviotietojen päivittämiseen siemen-, suojus- tai verhopuuhakkuun jälkeen, kun kyseessä on puustoinen kuvio. Jos siemen-, suojus- tai verhopuuhakkuu on tehty samalla simulointijaksolla, päivitetään kuvion pääpuulaji, pohjapinta-ala, tilavuus, keskiläpimitta, keskipituus, runkoluku, koivun osuus pohjapinta-alasta, valtapuiden keskiläpimitta ja latvussuhde. Edelleen asetetaan siemenpuukuviomuuttujan arvoksi 1, nollataan kuviotason muuttujien arvoja sekä asetetaan taimettomasta siemenpuukuviosta kertovan muuttujan arvoksi 1. Edelleen päivitetään kuvion kehitysluokka. Sitä varten asetetaan ennen toimenpiteen toteutusta ollut kehitysluokka vanha_kehitysluokka -muuttujan arvoksi, eri jaksojen ikäeroksi 0 ja lasketaan uudistamisrajat. Lopuksi päivitetään kuvion puuston arvo.

Chain Assign seedtree_stratum to 1 for Upper_st stratum

Käytetään kaksijaksoisen kuvion ylemmän jakson määrittämiseen siemenpuuositteeksi.

Chain Forced remove seed trees

Käytetään pakotetun siemenpuiden poiston toteuttamiseen, kun kyseessä on kaksijaksoinen kuvio.

Chain Update comp_unit after forced removing seed trees

Käytetään kuviotietojen päivittämiseen pakotetun siemenpuiden poiston jälkeen. Samalla simulointijaksolla siemenpuiden poiston kanssa asetetaan ensin puhtaasta taimikkokuviosta kertovan muuttujan oletusarvoksi 1. Jos kuitenkin kuviolla on muitakin kuin taimikko-ositteita, asetetaan puhdas taimikko -muuttujan arvoksi 0. Siemenpuukuvio -muuttujan arvoksi asetetaan 0. Päivitetään kuvion runkoluku, odotettavissa oleva runkoluku, ikä, ositteiden lukumäärä ja aritmeettinen keskipituus. Edelleen nollataan kuviotason muuttujien arvoja ja päivitetään taimikkokuviosta kertovan muuttujan arvo. Jos siemenpuiden poiston jälkeen kuviolla on puhdas taimikko, päivitetään pääpuulaji, keski- ja valtapituus sekä nollataan kuviotason muuttujien arvoja. Jos jäljelle jäävä puusto ei ole puhdas taimikko, päivitetään pohjapinta-ala, pääpuulaji, tilavuus, keskiläpimitta, keski- ja valtapituus, koivun osuus pohjapinta-alasta, kuvauspuiden lukumäärä, suhteellinen tiheys sekä valtapuiden keskiläpimitta ja latvussuhde. Edelleen päivitetään kuvion kehitysluokka. Sitä varten asetetaan ennen toimenpiteen toteutusta ollut kehitysluokka vanha_kehitysluokka -muuttujan arvoksi, eri jaksojen ikäeroksi 0 ja lasketaan uudistamisrajat. Lopuksi asetetaan kuvion puuston arvoksi nolla.

Chain Calculate productive value

Käytetään kuvion maan ja puuston tuottoarvon laskentaan. Kun kyseessä on puustoinen kuvio, nollataan puulajeittaiset pohjapinta-alat ja runkoluvut kuviolla kertovien muuttujien arvot. Tämän jälkeen lasketaan näille muuttujille uudet arvot. Tuottoarvo lasketaan puulajeittaisilla pohjapinta-aloilla painottaen. Asetetaan taimikon tuottoarvon kertovan muutujan oletusarvoksi nolla. Lasketaan taimikon ja varttuneen metsän puuston tuottoarvo sekä paljaan maan tuottoarvo. Mineraali ja turvemaalle laskenta tehdään samalla mallilla, mutta turvemaan tuottoarvoa on mahdollista laskea kertoimella. Puuttoman kuvion puuston tuottoarvoksi asetetaan nolla.

Forced regeneration

Chain Forced planting

Käytetään pakotetun istutuksen toteuttamiseen. Istutuksen jälkeen asetetaan kuviotason muuttujille alkuarvoja. Edelleen asetetaan istutuksesta kertovan muuttujan arvoksi 1. Lasketaan runkoluku, ikä, pohjapinta-ala, tilavuus, kuvion odotettavissa oleva runkoluku ja pääpuulaji ositetietojen avulla.

Chain Forced seeding

Käytetään pakotetun kylvön toteuttamiseen. Kylvön jälkeen asetetaan kuviotason muuttujille alkuarvoja. Edelleen asetetaan istutuksesta kertovan muuttujan arvoksi 0. Lasketaan runkoluku, ikä, pohjapinta-ala, tilavuus, kuvion odotettavissa oleva runkoluku ja pääpuulaji ositetietojen avulla.

Chain Forced natural regeneration

Käytetään pakotetun luontaisen uudistamisen toteuttamiseen.

Chain Calculate attributes to stratum after planting or seeding

Käytetään ositetietojen täydentämiseen pakotetun istutuksen tai kylvön jälkeen. Lasketaan rinnankorkeuden saavuttamiseen kuluva aika. Arvo määräytyy sijainnin (kunta), kasvupaikan ja puulajin mukaan. Lisäksi metsämaalle sekä kitu- ja joutomaalle on omat mallinsa. Edelleen kasvatetaan istutetun ositteen keskipituutta ja ikää. Tätä varten lasketaan taimikon vuotuinen pituuskasvu.

Chain Calculate attributes to comp_unit after planting

Käytetään ositetietojen täydentämiseen pakotetun istutuksen jälkeen. Päivitetään kuvion keskipituus ja ikä.

Forced regeneration check

Chain Forced supplemental_planting

Käytetään pakotetun täydennysistutuksen toteuttamiseen. Lasketaan puulajeittaiset runkoluvut. Täydennysistutuksen jälkeen asetetaan istutuksesta kertovan muuttujan arvoksi 1. Lasketaan runkoluku, ikä, pohjapinta-ala, tilavuus, kuvion odotettavissa oleva runkoluku ja pääpuulaji ositetietojen avulla.

Chain Calculate attributes to stratum after supplemental planting

Käytetään ositetietojen täydentämiseen pakotetun täydennysistutuksen jälkeen. Lasketaan rinnankorkeuden saavuttamiseen kuluva aika. Arvo määräytyy sijainnin (kunta), kasvupaikan ja puulajin mukaan. Lisäksi metsämaalle sekä kitu- ja joutomaalle on omat mallinsa. Edelleen kasvatetaan istutetun ositteen keskipituutta ja ikää. Tätä varten lasketaan taimikon vuotuinen pituuskasvu.

Chain Calculate attributes to comp_unit after supplemental planting

Käytetään kuviotietojen täydentämiseen pakotetun täydennysistutuksen jälkeen. Päivitetään kuvion keskipituus ja ikä.

Chain Forced grass suppression

Käytetään pakotetun mekaanisen heinäntorjunnan toteuttamiseen.

Chain Forced chemical grass suppression

Käytetään pakotetun mekaanisen heinäntorjunnan toteuttamiseen.

Forced soil preparation

Chain Forced scarification

Käytetään pakotetun maanmuokkauksen toteuttamiseen laikuttamalla. Maanmuokkauksen toteuttamisen jälkeen asetetaan maanmuokkauksesta kertovan muuttujan arvoksi 1.

Chain Forced mounding

Käytetään pakotetun maanmuokkauksen toteuttamiseen laikkumätästämällä. Maanmuokkauksen toteuttamisen jälkeen asetetaan maanmuokkauksesta kertovan muuttujan arvoksi 1.

Chain Forced harrowing

Käytetään pakotetun maanmuokkauksen toteuttamiseen äestämällä. Maanmuokkauksen toteuttamisen jälkeen asetetaan maanmuokkauksesta kertovan muuttujan arvoksi 1.

Forced tending and management of seedling and young stands

Chain Forced tending of young stands

Käytetään pakotetun taimikonhoidon toteuttamiseen. Jos kuviolla on olemassa puita, taimikonhoito tehdään aidosti puita poistamalla, muuten kyseessä on taimikon perkaus.

Chain Forced early tending of young stands

Käytetään pakotetun varhaistaimikonhoidon toteuttamiseen. Jos kuviolla on olemassa puita, varhaistaimikonhoito tehdään aidosti puita poistamalla, muuten kyseessä on taimikon perkaus.

Chain Forced cleaning

Käytetään pakotetun perkauksen toteuttamiseen.

Chain Forced improvement_of_young_stand

Käytetään pakotetun nuoren metsän hoidon toteuttamiseen.

Chain Update stratum after forced tending or improvement of young stand

Käytetään päivitettäessä ositetietoja taimikonhoidon tai nuoren metsän kunnostuksen jälkeen samalla simulointijaksolla. Jos ositteen pohjapinta-ala on suurempi kuin nolla, päivitetään pohjapinta-ala, keskiläpimitta, keskipituus, runkoluku, tilavuus ja kuvauspuiden lukumäärä.

Chain Update comp_unit after forced tending or improvement of young stand

Käytetään päivitettäessä kuviotietoja taimikonhoidon tai nuoren metsän kunnostuksen jälkeen samalla simulointijaksolla. Jos puutaso on olemassa, päivitetään kuvion pääpuulaji, pohjapinta-ala, tilavuus, keskiläpimitta, keskipituus, runkoluku, odotettu runkoluku, ositteiden lukumäärä, koivun osuus pohjapinta-alasta, kuvauspuiden lukumäärä, valtapuiden keskiläpimitta ja latvussuhde. Jos puutaso puuttuu, päivitetään keskipituus, runkoluku, odotettu runkoluku ja ositteiden lukumäärä. Tulkitaan ositteiden tietoja käyttäen onko kuvio kaksijaksoinen. Jos kuvio on yksijaksoinen, nollataan ylempään ja alempaan jaksoon kuuluvien ositteiden lukumäärät. Jos kuvio on kaksijaksoinen, lasketaan sekä ylempään että alempaan jaksoon kuuluvien ositteiden lukumäärät. Jos eri ositteisiin kuulumisen tulkinnan kanssa on ollut ongelmia eli ylemmän tai alemman jakson ositteiden lukumäärää ei ole laskettu (esim. kaikki ositteet kuuluvat ylempään jaksoon, siis kyseessä on yksijaksoinen kuvio), asetetaan ylempään ja alempaan jaksoon kuuluvien ositteiden lukumääräksi nolla. Tällöin myös kaksijaksoisuudesta kertovan muuttujan arvoksi asetetaan 0 eli kuvio tulkitaan yksijaksoiseksi.

Forced clearing

Chain Forced clearing

Käytetään pakotetun raivauksen toteuttamiseen.

Chain Forced clearing_before_harvest

Käytetään pakotetun ennakkoraivauksen toteuttamiseen.

Forced pruning

Chain Forced pruning

Käytetään pakotetun pystykarsinnan toteuttamiseen.

Forced ditching

Chain Forced ditch cleaning

Käytetään pakotetun kunnostusojituksen toteuttamiseen.

Forced fertilization

Chain Forced fertilization

Käytetään pakotetun kasvatuslannoituksen toteuttamiseen.

Chain Forced vitality fertilization

Käytetään pakotetun terveyslannoituksen toteuttamiseen.

9.2 Metsikkösimulaattori

9.2.1 Johdanto

Tämä dokumentti sisältää SIMO-järjestelmän metsikkötason simulaattorin kuvauksen. Simulaattorissa käytetään Vuokilaisen ja Väliahon, Mielikäisen, Oikarisen sekä Saramäen metsikkötason kasvumalleja. Kuvaus on tarkoitettu antamaan sekä järjestelmän käyttäjille että järjestelmän ylläpitäjille ja kehittäjille yleiskuvan niistä tiedoista ja toiminnoista, joita metsikkötason malliketjut sisältävät. Lisäksi on kuvattu malliketjujen rakennetta. Metsikkötason kasvumallit tarkoittavat sitä, että koko puujoukon kasvua kuvataan muutaman sen edustaman tunnuksen avulla (pohjapinta-alan ja valtapituuden kasvu). Toimenpiteiden toteutuksen yhteydessä puulajiositteille luodaan lisäksi kuvauspuut apteerausta varten. Muuten puutason tietoa ei käytetä. Ositetasolla puhutaan taimikkoositteesta, kun ositteeseen kuuluu vähintään yksi alle 1,3 m pitkä puu. Kuviotasolla puhutaan taimikkokuviosta, jos jokin kuvion osite on taimikko-osite eli sillä on vähintään yksi alle 1,3 m pitkä puu. Näin huomioidaan laskelmissa niiden puiden läsnäolo, joilla ei vielä ole rinnankorkeusläpimittaa. Dummy-muuttujien kohdalla

asetettu arvo ilmaistaan numeroin 0 tai 1. Metsikkötason simulointia ei tällä hetkellä käytetään suunnittelujärjestelmissä, vaan käytössä ovat puutason kasvumallit. Pitkällä aikavälillä puutason mallit saattavat kuitenkin tuottaa epäluotettavia ennusteita ja toisaalta suuralueiden suunnitteluun ne ovat usein laskennallisesti liian raskaita. Tällaisissa tapauksissa metsikkötason kasvumalleille voisi olla tarvetta. Tavoitteina erilaisten simulaattoreiden kehittämiselle voidaan siten nähdä koko metsäsuunnittelun laskelmien keventäminen (tarkoituksenmukaiselle tasolle tavoitteen mukaan) ja luotettavuuden parantaminen.

9.2.2 Malliketjut

Malliketju:

- Complete data
- Calculate missing variables

lasketaan vain kerran simuloinnin aluksi aineiston täydentämiseksi vastaamaan simulaattorin tarpeita. Näitä malliketjuja käytetään tavanomaisen kuviottaisen arvioinnin tuloksena tuotetun keskitunnustiedon kanssa. Simulointiin, nykytilan laskentaan tai metsikön tulevaisuuden tilan ennustamiseen käytetään ketjuja:

- Regeneration
- Grow seedling stratum
- Grow stratum
- Update basic attributes
- Generate trees
- Generate tree attributes
- Calculate biomass
- Calculate stand value growth
- Calculate stand productive value

Toimenpiteet toteutetaan malliketjulla:

- Harvests
- Silvicultural operations

Harvests sisältää hakkuiden ja Silvicultural operations metsänhoitotoimenpiteiden toteutuksen. Lisäksi toimenpiteiden pakotusta varten on oma malliketjunsa:

- Forced operations

9.2.3 Aineiston täydennysketjut

Complete data

Käytetään aineiston kuvio- ja ositetietojen täydentämiseen ennen kasvatuksessa tarvittavien muuttujien laskentaa. Ketjua käytetään vain kerran kullekin aineistolle.

Chain Complete stand attributes

Käytetään kuvion maantieteellisten, kuvion maaperää ja metsähoidollisten toimenpiteiden tilaa kuvaavien tunnusten laskentaan. Asetetaan kuviotason muuttujien oletusarvoja. Jos ositetta ei ole, asetetaan aukea-muuttujan arvoksi 1. Metsätalousmaakuvioille lasketaan kuntatietoihin perustuen YKJ mukaiset koordinaatit, jos koordinaateilla ei aineistossa alunperin ole arvoa. Lasketaan EUREF_FIN -järjestelmän mukaiset maantieteelliset koordinaatit. Näiden koordinaattien avulla lasketaan sijainnin mukaan määräytyvien muuttujien kuten korkeus merenpinnasta ja lämpösumma arvot. Asetetaan hinta-alueeksi 1. Sijainnin mukaan voidaan hintatauluihin määritellä toimenpiteille erilaiset hinnat. Tällä hetkellä on käytössä vain yhdet hinnat eli hinta-alue 1 tarkoittaa koko maata. Edelleen asetetaan maaluokaksi keskikarkea moreeni, jos maaluokkaa ei aineistosta löydy. Oletusarvona asetetaan kantojen nostossa hehtaarille jätettävien kantojen lukumääräksi 25. Jos maaluokka on savi, asetetaan hehtaaria

kohti nostamatta jäävien kantojen lukumääksi 50. Bioenergian keruumallia varten asetetaan kantojen_keruu ja oksien_keruu -muuttujille arvoksi 1. Edelleen asetetaan kantojen ja oksien keräämiseen bioenergiatarkoitukseen sopimattomilla kohteilla kantojen_keruu ja oksien_keruu -muuttujille arvoksi 0. Asetetaan niiden kuvioden kehitysluokaksi aukea ala (1) ja ikäluokaksi 0, joilla ei ole ositteita. Lopuksi tarkastellaan vielä muuttujaa, joka kertoo kuluneen ajan viimeisestä ojituksesta kuviolla. Jos aika_ojituksesta -muuttuja puuttuu, mutta ojitusvuosi on sen sijaan tiedossa, lasketaan aika ojituksesta. Jos sekä aika_ojituksesta että ojitusvuosi puuttuvat, asetetaan aika_ojituksesta -muuttujan arvoksi nolla.

Chain Complete stratum variables

Käytetään puulajiositteen alkuarvojen asettamiseen. Asetetaan puulajiositteille muuttujien oletusarvoja, kun kyseessä on puustoinen kuvio.

Calculate missing attributes

Käytetään aineiston kuvio- ja ositetietojen täydentämiseen ennen puuston kasvatusta. Ketjua käytetään vain kerran kullekin aineistolle. Tarvittavat lisätiedot riippuvat puulajista (eri puulajeilla on erilaisia malleja, joihin tarvitaan tiettyjä muuttujia).

Chain Assign missing initial variables to stratum

Käytetään ositetason muuttujien täydentämiseen, kun kyseessä on puustoinen kuvio. Lasketaan puulajiositteelle ikälisä, siis aika, joka kuluu puun kasvamiseen rinnankorkeuteen. Jos ositteen keskiläpimitta on suurempi kuin nolla, tarkistetaan, että keskiläpimitan ja keskipituuden suhde on järkevä. Keskipituusarvoa käytetään edelleen valtapituuden määrittämiseen. Lasketaan runkoluku, jos sitä ei vielä ositteella ole. Lopuksi lasketaan ositteen pohjapinta-ala, jos keskiläpimitta on suurempi kuin nolla, mutta pohjapinta-alan arvoa ei ole. Määritetään onko kyseessä taimikko-osite. Jos kyseessä on taimikko-osite asetetaan puuttuvia arvoja - ei kuitenkaan mäntyositteelle, koska sille on käytössä myös taimikkomallit. Lasketaan ositteen kuoreton pohjapinta-ala sekä rinnakorkeusikä puulajeittaisilla malleilla. Lehtipuille lasketaan lisäksi kannonkorkeusikä. Jos kannonkorkeusikä saa negatiivisen arvon, sen arvoksi asetetaan nolla. Lopuksi asetetaan muotoluvun oletusarvoksi 0,5 ja lasketaan ositteen tilavuus.

Chain Count number of strata and calculate stand attributes

Käytetään kuviotason muuttujien täydentämiseen, kun kyseessä on puustoinen kuvio. Lasketaan kuvion runkoluku sekä odotettavissa oleva runkoluku. Edelleen lasketaan ositteiden lukumäärä ja aritmeettinen keskipituus. Aritmeettinen keskipituus tarkoittaa tässä ositteiden keskipituuksien keskiarvoa. Summataan kuvion pohjapinta-ala, lasketaan koivun osuus pohjapinta-alasta sekä kuoreton pohjapinta-ala. Jos pohjapinta-ala on suurempi kuin nolla, lasketaan keskiläpimitta ja keskipituus pohjapinta-alalla painottaen. Kuvion valtapituudeksi, pääpuulajiksi ja iäksi määräytyy sen ositteen arvo, jonka pohjapinta-ala on suurin. Jos pohjapinta-ala on nolla, asetetaan myös keskiläpimitta nollaksi. Keskipituus lasketaan tällöin runkoluvulla painottaen. Edelleen kuvion valtapituudeksi, pääpuulajiksi ja iäksi määräytyy sen ositteen arvo, jonka runkoluku on suurin. Lasketaan tilavuus, jos se puuttuu. Lopuksi lasketaan vielä kasvupaikkaindeksi puulajeittaisilla malleilla.

Chain Whether stratum belongs to upper storey

Käytetään ositetason muuttujien täydentämiseen, kun kyseessä on puustoinen kuvio. Lasketaan kuuluuko osite ylempään jaksoon. Ositteen kuuluminen ylempään jaksoon tulkitaan vertaamalla kuvion aritmeettista keskipituutta ositteen pohjapinta-alalla painotettuun keskipituuteen.

Chain Calculate attributes if stand is two-storey

Käytetään kuviotason muuttujien täydentämiseen, kun kyseessä on puustoinen kuvio. Ositteiden tietoja käyttäen tulkitaan onko kuvio kaksijaksoinen. Kaksijaksoisilla kuvioilla lasketaan sekä ylempään että alempaan jaksoon kuuluvien ositteiden lukumäärät. Jos eri ositteisiin kuulumisen tulkinnan kanssa on ollut ongelmia eli ylemmän tai alemman jakson ositteiden lukumäärää ei ole laskettu (esim. kaikki ositteet kuuluvat ylempään jaksoon, siis kyseessä on yksijaksoinen kuvio), asetetaan ylempään ja alempaan jaksoon kuuluvien ositteiden lukumääräksi nolla. Tällöin myös kaksijaksoisuudesta kertovan muuttujan arvoksi asetetaan 0 eli kuvio tulkitaan yksijaksoiseksi.

Chain Update upper-storey attribute if problems defining two-storey stand

Käytetään, kun kaksijaksoisuuden määrittelyn kanssa on ollut ongelmia puustoisella kuviolla. Tällöin myös ositetasolla virheellinen tieto kuulumisesta ylempään jaksoon korjataan.

Chain Calculate missing attributes to stand

Käytetään asetettaessa simuloinnissa tarvittavia kuviotason muuttujia puustoisella kuviolla. Aluksi, jos kuvion pääpuulaji on hieskoivu, asetetaan hieskoivumuuttujan arvoksi 1. Jos kyseessä on kaksijaksoinen kuvio, lasketaan sekä ylemmän että alemman jakson keski-ikä. Tämän jälkeen lasketaan jaksojen ikäero. Ikäeroa käytetään ylispuustoisen taimikon määrittämiseen kehitysluokkamallissa. Ennen kuvion kehitysluokan tulkintaa lasketaan uudistamisrajat. Uudistamisrajana sekä keskiläpimitalle että keski-iälle käytetään tunnuksen sallitun vaihteluvälin keskiarvoa (0,5). Jos kehitysluokka on siemenpuu-, suojuspuumetsikkö tai ylispuustoinen taimikko, asetetaan siemenpuukuviomuuttujan arvoksi 1 ja uudistuskypsästä metsiköstä kertovan muuttujan arvoksi 0. Lopuksi vielä asetettaan siemenpuukuvio_ilman_taimikkoa -muuttujan arvoksi 1, jos kyseessä on siemenpuu- tai suojuspuumetsikkö.

Chain Calculate missing initial variables to stratum

Käytetään ositetason muuttujien täydentämiseen, kun kyseessä on puustoinen kuvio. Siemenpuuositemuuttujan arvoksi asetetaan 1, jos kyseessä on ylispuustoisen taimikkokuvion ylempi osite tai siemen- tai suojuspuukuvion osite. Jos ositteen keskipituus on pienempi kuin 1,3 m, asetetaan taimikkomuuttujan arvoksi 1. Lopuksi lasketaan iän ja pituuden kalibrointikerroin ositteille, joilla keski-ikä ja -pituus on suurempi kuin nolla. Kalibroinnilla pyritään siihen, että simuloinnissa käytettävistä kuvauspuista laskettu keski-ikä ja -pituus olisi sama kuin ositteen keski-ikä ja -pituus alunperin.

Chain Calculate still missing initial stand attributes

Käytetään kuviotason muuttujien täydentämiseen, kun kyseessä on puustoinen kuvio. Lasketaan koivun osuus pohjapinta-alasta. Edelleen lasketaan taimikko-ositteiden lukumäärä. Jos kuviolla ei ole yhtään taimikko-ositetta, asetetaan taimikkokuviosta kertovan muuttujan arvoksi 0.

9.2.4 Simulointiketjut

Regeneration

Käytetään puuston uudistamisen toteutukseen ja uudistamisen jälkeisen puuston muuttujien arvojen laskemiseen. Ketjua käytetään tavallisesti jokaisen kasvatusjakson alussa. Sitä voidaan käyttää myös kasvatusjakson lopussa toimenpidemallien jälkeen, jos uudistamistoimenpide halutaan toteuttaa saman vuonna uudistushakkuun kanssa. Tämä tulee kyseeseen erityisesti silloin, kun kasvatusjakso on pitkä (esim. 5 vuotta). Ilman tällaista menettelyä uudistamistoimenpide tulisi toteutettavaksi vasta seuraavalla 5-vuotisjaksolla.

Chain Regenerate clear cutted stand and open area using planting

Käytetään luotaessa avohakatulle alalle tai lähtötiedoissa aukealle alueelle uusi puusto istuttamalla. Uudistettavan kuvion puulaji määräytyy kasvupaikan mukaan. Istutusmallin jälkeen asetetaan kuviotason muuttujille alkuarvoja. Edelleen asetetaan istutuksesta kertovan muuttujan arvoksi 1. Lasketaan runkoluku, ikä, pohjapinta-ala, tilavuus, kuvion odotettavissa oleva runkoluku ja pääpuulaji ositetietojen avulla.

Chain Calculate attributes to stratum after planting

Käytetään istutuksen jälkeen ositetietojen täydentämiseen. Lasketaan ositteelle ikälisä eli aika, joka kuluu rinnankorkeuden saavuttamiseen. Lasketaan ositteen kuoreton pohjapinta-ala puulajeittaisilla malleilla.

Chain Update variables after planting

Käytetään kasvupaikkaindeksin laskentaan avohakkuun jälkeen. Lasketaan kasvupaikkaindeksi puulajeittaisilla malleilla.

Chain Natural regeneration after seed tree cut

Käytetään luontaisen uudistamisen toteutukseen siemenpuu-, suojuspuu- ja kaistalehakkuin käsitellyille kuvioille. Aluksi asetetaan seuraavalla viiden vuoden jaksolla syntyvien puiden määräksi nolla. Uudistushakkuuta jälkeen lasketaan luontaisesti syntyvien puiden lukumäärä, kun kuvio on määritetty siemenpuu- tai kaistalehakkuukuvioksi, mutta taimiositetta ei vielä ole olemassa. Laskentaa varten tarvitsee kuitenkin määrittää ensin uudistuksessa käytettävä puulaji, jos se puuttuu ja tämän jälken laskea keskimääräinen luontaisesti syntyvän puuston runkoluku kasvupaikan ja puulajin mukaan. Jos edellä laskettu syntyvien puiden lukumäärä on suurempi kuin nolla, seuraavaksi määritetään kuvion pääpuulaji, jos sitä ei ole tiedossa (kaistelehakkuussa pääpuulaji puuttuu). Uudistushakkuun jälkeisen luontaisen uudistamisen seurauksena syntyvä uusi puusto kuvataan viidellä uudella kuvauspuulla. Alussa jokainen kuvauspuu saa viidesosan kuviolle syntyvästä runkoluvusta. Jokaiselle

kuvauspuulle määräytyy puulaji seuraavasti: Ensimmäisen kuvauspuun puulaji (pääpuulaji) määräytyy kasvupaikan ja maaperän (kivennäismaa/turvemaa) mukaan. Tämän jälkeen neljä muuta kuvauspuuta saavat puulajin satunnaisluvun avulla. Tietyn puulajin todennäköisyys määräytyy pääpuulajin, kasvupaikan ja maaperän (kivennäismaa/turvemaa) mukaan. Samaa puulajia olevat ositteet summataan yhteen, joten luontaisen uudistamisen seurauksena syntyy aina 1-5 uutta ositetta. Luontaisen uudistamisen jälkeen asetetaan kuviotason muuttujille alkuarvoja. Lopuksi päivitetään luontaisesti syntyneiden sisäänkasvuvaiheessa olevien ositteiden yhteenlaskettu runkoluku. Sisäänkasvuvaihe tarkoittaa vaihetta, jossa osite on jo olemassa, mutta puiden syntyminen tapahtuu vasta tulevina vuosina.

Chain Calculate attributes to natural ingrowth stratum

Käytetään luontaisesti uudistetun kuvion ositetason muuttujien täydentämiseen. Samalla kasvatusjaksolla luontaisesti syntyneille ositteille asetetaan ositetason muuttujien alkuarvoja. Edelleen lasketaan taimikon tuleva syntymävuosi. Tätä varten lasketaan ositteille teoreettinen ikä, joka saa arvon -1 ja -5 väliltä (tämä tarkoittaa, että puut syntyvät ositteelle 1 - 5 vuoden kuluttua). Edelleen lasketaan rinnankorkeuden saavuttamiseen kuluva aika. Lopuksi asetetaan kuoreton pohjapinta-ala nollaksi.

Chain Update N_ingrowth to stratum level

Käytetään luontaisesti syntyneen sisäänkasvuvaiheessa olevan ositteen runkoluvun päivittämiseen. Kun luontaisesti syntyneen ositteen teoreettinen ikä saa arvon nolla, eli on vuosi, jolloin puut syntyvät, siirretään sisäänkasvuvaiheessa olleen ositteen runkoluku ositteen oikeaksi runkoluvuksi. Sisäänkasvuvaiheessa olevan ositteen puiden runkoluku asetetaan nollaksi. Tällöin sisäänkasvuvaiheen haamupuista on tullut oikeita puita.

Chain Update N_ingrowth to stand level

Käytetään luontaisesti syntyneen sisäänkasvuvaiheessa olevan kuvion runkoluvun päivittämiseen. Summataan sisäänkasvuvaiheessa olevien ositteiden runkoluvut kuviotasolle. Päivitetään runkoluku ja odotettavissa oleva runkoluku. Lopuksi, jos kuvion runkoluku on suurempi kuin nolla, asetetaan avoimesta alasta (puuttomasta kuviosta) kertovan muuttujan arvoksi 0. Samoin asetetaan taimettomasta siemenpuukuviosta kertovan muuttujan arvoksi 0, jos sisäänkasvuvaiheessa olevien puiden runkoluku on suurempi kuin nolla. Tämän avulla voidaan päätellä, että kuviolle on syntymässä puita.

Grow seedling stratum

Käytetään taimikko-ositteen kasvatukseen sekä tietojen päivittämiseen kasvatuksen jälkeen.

Chain Grow seedling stratum

Käytetään kasvattamaan taimikkoa. Asetetaan muuttujan puut_juuri_generoitu oletusarvoksi 0. Asetetaan tilavuus_vanha muuttujan arvoksi ositteen tilavuus ennen kasvatusta. Määritetään onko kyseessä taimikko-osite. Jos kyseessä on taimikko, jossa puita ei ole vielä generoitu ja teoreettinen ikä on nolla, asetetaan ei mäntyositteelle muuttujan arvoja. Edelleen lasketaan mäntyositteelle, jonka keskipituus on alle 1,3 m rinnankorkeusläpimitan referenssiarvo, kun rinnankorkeus saavutetaan. Edelleen lasketaan vuotuinen pituuskasvu ja kasvatetaan ositteen keskipituutta ja päivitetyn keskipituuden perusteella lasketaan valtapituus. Muulle kuin männyn taimikko-ositteelle lasketaan kasvuprosentti ja kasvatetaan sen jälkeen ositteen valtapituutta, pohjapinta-alaa, tilavuutta, keskiläpimittaa ja -pituutta sekä lasketaan kuoreton pohjapinta-ala (koivulla kasvatetaan kuorellista pohjapinta-alaa ja lasketaan tämän jälkeen kuoreton pohjapinta-ala, kuusella kasvatetaan myös kuorellista pohjapinta-alaa, mutta lasketaan ensin kuoren pohjapinta-ala ja sitä kautta kuoreton pohjapinta-ala). Jos keskipituus saavuttaa 1,3 m, lasketaan ositteelle keskiläpimitta. Seuraavaksi muodostetaan ositteelle pituusjakauma ja lasketaan kuvauspuille läpimitat. Asetetaan muuttujien puut_generoitu, puut_juuri_generoitu ja normaalijakauma arvoiksi 1. Lopuksi lasketaan kuvauspuiden lukumäärä.

Calculate tree variables to just generated tree

Käytetään laskettaessa muuttujien arvoja juuri luoduille puille. Jos ositteelle on juuri luotu puut, asetetaan jokaisen kuvauspuun rinnakorkeusikä ositteen iän ja rinnankorkeuteen kasvamiseen kuluvan ajan perusteella. Jos rinnankorkeusikä saa negatiivisen arvon, asetetaan sen arvoksi nolla. Edelleen asetetaan puutason muuttujille alkuarvoja. Jos puun läpimitta on suurempi kuin nolla, lasketaan yksittäisen kuvauspuun poikkileikkausala rinnankorkeudelta. Edelleen lasketaan jakauman jokaisen läpimittaluokan poikkileikkausala. Lisäksi lasketaan jokaisen yksittäisen kuvauspuun poikkileikkausala neliömetreinä. Edelleen lasketaan puun kannonkorkeusläpimitta, jonka jälkeen voidaan laskea puun kasvutilan minimi puulajeittaisilla malleilla. Lopuksi lasketaan puun tilavuus.

Chain Update attribute values in stratum level

Käytetään mäntyositteen muuttujien arvojen päivitykseen, kun ositteelle on juuri luotu puut. Lasketaan keskiläpimitta, pohjapinta-ala, kuoreton pohjapinta-ala, tilavuus ja rinnakorkeusikä.

Chain Update attribute values in comp_unit level

Käytetään kuviotietojen päivitykseen taimikon kasvatuksen jälkeen. Päivitetään kuoreton pohjapinta-ala. Asetetaan puhdas_taimikko -muuttujan oletusarvoksi 0. Lasketaan puut_generoitu -muuttujan summa. Jos yhdellekin kuvion ositteista on jo luotu puut, asetetaan puut_generoitu -muutujan arvoksi 1. Jos taas puita ei ole luotu vielä yhdellekään ositteista, asetetaan puhdas_taimikko -muuttujan arvoksi 1. Asetetaan tilavuus_vanha muuttujalle arvoksi ositteen tilavuus ennen tilavuuden päivitystä.

Grow stratum

Käytetään ositteen puuston kasvattamiseen, kun ositteen keskipituus on ylittänyt rinnankorkeuden.

Chain Grow stratum

Käytetään puuston ositetunnusten kasvun ennustamiseen puulajeittaisilla malleilla, kun ositteelle ei ole juuri luotu puita. Tätä ennen kuitenkin asetetaan valtapituus_ennen_kasvua -muuttujan arvoksi sen hetkinen valtapituus. Kasvumallit on jaettu ositteen varhaiskasvatusmalleihin ja varttuneen puuston kasvumalleihin. Männylle käytetään varhaiskasvatukseen ennustemalleja, mutta muilla puulajeilla on käytössä tavoitetaimikkomallit. Jokaisen puulajin kohdalla ensin määritetään onko kyseessä varttunut taimikko-osite. Varttuneelle taimikko-ositteelle lasketaan kasvuprosentti, kun puulaji on muu kuin mänty, ja kasvatetaan sen jälkeen ositteen valtapituutta, pohjapinta-alaa, tilavuutta, keskiläpimittaa ja -pituutta sekä lasketaan kuoreton pohjapinta-ala (koivulla kasvatetaan kuorellista pohjapinta-alaa ja lasketaan tämän jälkeen kuoreton pohjapinta-ala, kuusella kasvatetaan myös kuorellista pohjapinta-alaa, mutta lasketaan ensin kuoren pohjapinta-ala ja sitä kautta kuoreton pohjapinta-ala). Varttuneen männyn taimikko-ositteen tapauksessa lasketaan valtapituus, pohjapinta-ala, valtapuiden keskiläpimitta, tilavuus, kuoreton pohjapinta-ala, keskiläpimitta ja keskipituus. Jos kyseessä on varttunut puusto-osite lasketaan havupuulla kasvu kasvupaikoittain valtapituudelle, jonka jälkeen päivitetään keskipituus sekä asetetaan ennen kasvatusta ollut pohjapinta-ala pohjapinta-ala_vanha -muuttujan arvoksi. Edelleen lasketaan pohjapinta-alan kasvu (havupuilla lasketaan kuorettoman ja lehtipuilla kuorellisen pohjapinta-alaa kasvu) ja lasketaan kuoren pohjapinta-ala, muotoluku sekä lopuksi tilavuus. Lehtipuilla asetetaan ennen kasvun laskentaa ollut pohjapinta-ala pohjapinta-ala_vanha -muuttujan arvoksi ja sen jälkeen lasketaan pohjapinta-alan, tilavuuden ja valtapituuden kasvu. Edelleen lasketaan keskipituus.

Update basic attributes

Käytetään tietojen päivittämiseen kasvatuksen jälkeen.

Chain Update stratum variables

Käytetään ositetason muuttujien arvojen kasvatukseen. Varhaiskasvatusvaiheen ohittaneelle ositteelle päivitetään eri muuttujien arvoja riippuen puulajista. Havupuilla päivitetään valtapituus, kuoreton pohjapinta-ala ja lopuksi lasketaan vielä kuorellinen pohjapinta-ala. Lehtipuilla päivitetään valtapituus, kuorellinen pohjapinta-ala ja lopuksi vielä lasketaan kuoreton pohjapinta-ala. Lehtipuilla lisäksi päivitetään tilavuus. Lopuksi kaikille puulajeille lasketaan vielä keskiläpimitta.

Chain Update comp_unit variables

Käytetään kuviotason muuttujien arvojen päivitykseen kasvatuksen jälkeen. Kun kyseessä ei ole puhdas taimikkokuvio, päivitetään keskiläpimitta ja -pituus, pohjapinta-ala, kuoreton pohjapinta-ala, koivun osuus pohjapinta-alasta, runkoluku, valtapituus, ikä ja pääpuulaji. Puhtaalle taimikkokuviolle päivitetään ikä, keskipituus ja valtapituus. Lopuksi päivitetään kaikille kuviolle tilavuus, aritmeettinen keskipituus, kuvauspuiden lukumäärä ja valtapuiden keskiläpimitta sekä nollataan keskiläpimitta, pohjapinta-ala ja kuoreton keskiläpimitta, jos keskipituus on alle 1,3 m.

Chain Update percentage of birch and Age to stratum level

Käytetään ositetason muuttujien arvojen päivitykseen kasvatuksen jälkeen. Päivitys tehdään hieman eri tavalla puulajista riippuen. Jos kyseessä on koivuosite, päivitetään koivun osuus koko kuvion pohjapinta-alasta. Lasketaan rinnankorkeusikä ja kasvatetaan ikää lisäämällä siihen simuloinnin aika-askeleen pituus. Nollataan ikä rinnankorkeudelta, jos sen arvo laskennassa on mennyt negatiiviseksi. Lehtipuuositteelle lasketaan ikä kannonkorkeudelta ja kasvatetaan ikää lisäämällä siihen simuloinnin aika-askeleen pituus. Nollataan ikä kannonkorkeudelta, jos sen arvo laskennassa on mennyt negatiiviseksi. Edelleen kasvatetaan biologista ja teoreettista ikää lisäämällä ikään simuloinnin aika-askeleen pituus. Päivitetään bioenergian keruussa ositteesta keräämättä jätettävien kantojen lukumäärä ositteen ja kuvion pohjapinta-alan suhteen perusteella. Lasketaan painotusmuuttuja runkoluku kertaa keskipituus. Lopuksi nollataan suhteellisen ja absoluuttisen tilavuuskasvun kertovien muuttujien arvot ja lasketaan näille muuttujille uudet arvot sekä asetetaan runkoluku_vanha -muuttujan arvoksi ositteeen runkoluku taimikonhoitomallia varten.

Chain Update stand Age

Käytetään biologisen iän sekä tilavuuskasvun päivittämiseen kuviotasolle. Asetetaan kuvion biologiseksi iäksi sen ositteen ikä, jonka pohjapinta-ala on suurin. Lasketaan kuvion ositteiden lukumäärä. Nollataan suhteellisen ja absoluuttisen tilavuuskasvun kertovien muuttujien arvot ja lasketaan näille muuttujille uudet arvot. Edelleen asetetaan hieskoivusta pääpuulajina kertovan muuttujan arvoksi 1, jos pääpuulaji on hieskoivu. Lopuksi asetetaan bioenergian keruuta varten kerää_kannot ja kerää_oksat -muuttujien arvoksi 0, jos kuvion pääpuulaji ei ole bioenergian keruuseen sopiva.

Chain Whether stratum belongs to upper storey

Käytetään tulkittaessa ei-taimikko-ositteen kuulumista puustojaksoihin kasvatuksen jälkeen. Lasketaan kuuluuko osite ylempään jaksoon. Ositteen kuuluminen ylempään jaksoon tulkitaan vertaamalla kuvion aritmeettista keskipituutta ositteen pohjapinta-alalla painotettuun keskipituuteen.

Chain Calculate attributes if stand is two-storey

Käytetään kuviotason muuttujien täydentämiseen, kun kyseessä on puustoinen kuvio. Ositteiden tietoja käyttäen tulkitaan onko kuvio kaksijaksoinen. Kaksijaksoisilla kuvioilla lasketaan sekä ylempään että alempaan jaksoon kuuluvien ositteiden lukumäärät. Jos eri ositteisiin kuulumisen tulkinnan kanssa on ollut ongelmia eli ylemmän tai alemman jakson ositteiden lukumäärää ei ole laskettu (esim. kaikki ositteet kuuluvat ylempään jaksoon, siis kyseessä on yksijaksoinen kuvio), asetetaan ylempään ja alempaan jaksoon kuuluvien ositteiden lukumääräksi nolla. Tällöin myös kaksijaksoisuudesta kertovan muuttujan arvoksi asetetaan 0 eli kuvio tulkitaan yksijaksoiseksi.

Chain Update upper-storey attribute if problems defining two-storey stand

Käytetään, kun kaksijaksoisuuden määrittelyn kanssa on ollut ongelmia puustoisella kuviolla kasvatuksen jälkeen. Tällöin myös ositetasolla virheellinen tieto kuulumisesta ylempään jaksoon korjataan.

Chain Update development and age class, since thinning and since drainage to comp_unit

Käytetään kehitys- ja ikäluokan sekä harvennuksesta ja ojituksesta kuluneesta ajasta kertovien muuttujien päivitykseen. Jos harvennus on toteutettu vähintään kerran ja harvennus ei ole tehty kuluvalla simulointijaksolla, päivitetään harvennuksesta kuluneen ajan kertova muuttuja. Samoin päivitetään ojituksesta kuluneen ajan kertova muuttuja, jos ojitus on tehty vähintään kerran, mutta ojitus ei ole tehty kuluvalla simulointijaksolla. Kehitysluokan laskemista varten asetetaan ensin keskiläpimitan arvoksi nolla, jos keskiläpimitta puuttuu. Asetetaan edellisellä simulointijaksolla laskettu kehitysluokkamuuttujan arvo vanha kehitysluokka -muuttujan arvoksi. Jos kuviolla ei ole ositetasoa, on kyseessä avoin uudistusala. Tällöin asetetaan aukeamuuttujan arvoksi 1 ja kehitysluokaksi aukea ala (1). Jos kyseessä on puustoinen kuvio, lasketaan ensin kaksijaksoiselle kuviolle ylempään ja alempaan jaksoon kuuluvien ositteiden keski-iät. Edelleen lasketaan näiden eri jaksojen välinen ikäero kehitysluokkamallia varten. Ikäeroa käytetään tulkittaessa onko kyseessä ylispuustoinen taimikko. Jos kuvio on yksijaksoinen eli ikäeromuuttujalle ei ole laskettu arvoa, asetetaan ikäeromuuttujan arvoksi nolla. Kehitysluokan laskemiseksi määritellään käytettävät uudistamisrajat. Edelleen lasketaan kuvion ikäluokka. Ensin asetetaan kuvion ikä nollaksi, jos se puuttuu. Jos kyseessä on avoin uudistusala, ikäluokka on myös nolla. Tällöin asetetaan myös aukeasta kertovan muuttujan arvoksi 1. Lopuksi lasketaan puustoisen kuvion ikäluokka.

Generate trees

Käytetään, kun halutaan tuottaa metsikön keskitunnustiedoista puutason tietoa kuvauspuiden käyttöä varten toimenpidemalleissa.

Chain Clear tree objects

Käytetään puustoiselle kuviolle mahdollisten puutason tietojen tuhoamiseen. Ositteelta hävitetään puutason kaikki muuttujat ja niiden tiedot, jotta uuden jakaumamallin tietoja ei sekoitettaisi vanhan jakaumamallin tietoihin, jos sellainen on olemassa.

Chain Generate trees

Käytetään jakaumamallien muodostamiseen puustoisella kuviolla. Asetetaan runkoluku_mitattu -muuttujan arvoksi 0, jos muuttujalla ei vielä ole arvoa. Malliketjussa tehdään jako keskitunnuksien perusteella iso- ja pienipuustoisiin ositteisiin. Pienille puustoille (määritetään keskiläpimitan mukaan) lasketaan pituusjakauma suoraan, mutta varttuneemman puuston kohdalla läpimittajakaumamallin valinta riippuu puulajista, maaperästä (mineraali- vai turvemaa), kehitysluokasta, sekä siitä, onko puuston runkoluku mitattu. Pienille puille laskettavan pituusjakauman yhteydessä ennustetaan myös puiden läpimitat. Jos ositteen puut muodostetaan pituusjakaumalla, päivitetään taimikko-ositteesta kertovan muuttujan arvo. Tätä varten lasketaan ositteella olevien alle 1,3 m pitkien puiden lukumäärä. Edelleen normaalijakaumamuuttujan arvoksi asetetaan 1. Läpimittajakauman muodostamisen jälkeen asetetaan taimikko-ositteesta kertovan muuttujan arvoksi 0. Puiden muodostamisen jälkeen asetetaan puut_generoitu -muuttujan arvoksi 1. Lopuksi asetetaan vielä runkoluku_mitattu -muutujan arvoksi 1, jotta seuraavalla laskentakerralla runkoluku pysyisi samana.

Chain Update seedling_below13 to stand level and calculate age class

Käytetään taimikkokuviomuuttujan päivittämiseen, kun kyseessä on puustoinen kuvio. Lasketaan taimikko-ositteiden lukumäärä. Jos kuviolla on yksikin taimikko-osite, asetetaan taimikkokuviosta kertovan muuttujan arvoksi 1. Lopuksi lasketaan kuvion ikäluokka. Ikäluokan määritystä varten lasketaan kuvion keski-ikä. Jos kyseessä on taimikkokuvio, kuvion keski-iäksi valitaan sen ositteen ikä, jonka runkoluku on suurin, muuten iäksi asetetaan pohjapinta-alaltaan suurimman ositteen ikä.

Generate tree attributes

Käytetään puiden luonnin jälkeen täydentämään toimenpiteiden yhteydessä apteerauksessa tarvittavia puutason tietoja.

Chain Generate tree attributes

Käytetään puutason muuttujien täydentämiseen kuvauspuiden luonnin jälkeen. Malliketjulla lasketaan puulajeittaisilla malleilla läpimittajakaumalla luoduille kuvauspuulle pituus ja ikä. Lisäksi lasketaan yksittäisen puun ja läpimittaluokan pohjapinta-ala sekä kannonkorkeusläpimitta. Edelleen asetetaan rinnankorkeusikä ositteen keski-iän ja rinnankorkeuden saavuttamiseen kuluvan ajan perusteella. Päivitetään rinnankorkeusikä. Jos tämän jälkeen rinnankorkeusikä on negatiivinen, se asetetaan nollaksi. Lopuksi lasketaan puun tilavuus.

Chain Calculate volume

Käytetään tilavuuden summaamiseen ositetasolle. Ensin asetetaan valtapituus_ennen_kasvatusta -muuttujan arvoksi ositteen valtapituus. Muuttujaa käytetään kasvumallien valintaan. Ositteen tilavuus lasketaan, jos ositteen puusto on sellainen, että kasvatukseen tullaan käyttämään taimikon kasvumalleja tavoitetaimikkokasvatuksen sijasta. Tällä hetkellä tällaiset mallit on olemassa vain männylle.

Chain Calculate the value of growing stock on stratum level

Käytetään ositteen puuston arvon sekä puustavaralajijakauman ja bioenergiaksi käytettävien kantojen ja hakkuutähteen määrän laskemiseen. Asetetaan kuvion ositteen puuston oletusarvoksi nolla. Jos kuvion kehitysluokka on jokin muu kuin aukea uudistusala tai nuori taimikko ja kyseessä ei ole taimikko-osite, lasketaan ositteen puuston arvo, puutavaralajijakauma sekä bioenergiapuun määrä.

Chain Calculate the value of growing stock on stand level

Käytetään kuvion puuston arvon määrittämiseen. Aluksi asetetaan kuvion puuston oletusarvoksi nolla. Tämän jälkeen summataan ositteiden puustojen arvot yhteen ja näin saadaan kuvion puuston arvo.

Calculate stand value growth

Käytetään kuvion puuston arvon määrittämiseen ja sitä kautta arvokasvun laskemiseen. Tuottaa samalla myös puutavaralajijakauman mukaanlukien oksista ja latvoista sekä kannoista ja juurista kertyvän bioenergian. Tätä ketjua voidaan käyttää myös aineiston täydennysketjujen yhteydessä ilman puuston kasvatusta. Arvokasvun laskennan saa kytkettyä pois päältä jättämällä tämän ketjun pois simuloinnista.

Chain Calculate the value of growing stock on stratum level

Käytetään ositteen puuston arvon sekä puustavaralajijakauman ja bioenergian määrän laskemiseen. Jos kuvion kehitysluokka on jokin muu kuin aukea uudistusala tai nuori taimikko ja kyseessä ei ole taimikko-osite, lasketaan ositteen puuston arvo ja puutavaralajijakauma sekä bioenergiapuun määrä.

Chain Calculate the stand value growth

Käytetään kuvion puuston arvokasvun määrittämiseen. Aluksi asetetaan kuvion arvokasvu, suhteellinen arvokasvu sekä keskimääräinen suhteellinen arvokasvu nollaksi ja edellisellä laskentajaksolla laskettu puuston arvo puuston arvo ennen kasvatusta -muuttujan arvoksi. Jos kuvion kehitysluokka on varttunut taimikko, nuori- tai varttunut kasvatusmetsikkö tai uudistuskypsä metsikkö, päivitetään laskentahetken puuston arvo summaamalla ositteiden puustojen arvot yhteen. Lopuksi lasketaan kasvu absosuuttisena sekä suhteellisena sekä viiden vuoden keskimääräinen arvokasvu.

Calculate stand productive value

Käytetään kuvion tuottoarvon laskentaan. Tätä ketjua voidaan käyttää myös aineiston täydennysketjujen yhteydessä ilman puuston kasvatusta. Tuottoarvon laskennan saa kytkettyä pois päältä jättämällä tämän ketjun pois simuloinnista.

Chain Calculate productive value

Käytetään kuvion maan ja puuston tuottoarvon laskentaan. Kun kyseessä on puustoinen kuvio ja luontaisen uudistamisen sisäänkasvuvaihe ei ole käynnissä, nollataan puulajeittaiset pohjapinta-alat ja runkoluvut kuviolla kertovien muuttujien arvot. Tämän jälkeen lasketaan näille muuttujille uudet arvot. Tuottoarvo lasketaan puulajeittaisilla pohjapinta-aloilla painottaen. Lasketaan sekä paljaan maan että puuston tuottoarvo. Mineraali ja turvemaalle laskenta tehdään samalla mallilla, mutta turvemaan tuottoarvoa on mahdollista laskea kertoimella. Puuttoman kuvion puuston tuottoarvoksi asetetaan nolla.

Calculate biomass

Käytetään kuvion puuston biomassan määrittämiseen. Biomassojen laskennan saa kytkettyä pois päältä jättämällä tämän ketjun pois simuloinnista.

Chain Calculate biomass of tree

Käytetään puun eri osien biomassan määrittämiseen. Jos kyseessä on ei-taimikko-osite, lasketaan puun eri osien biomassat.

9.2.5 Toimenpideketjut

Harvests

Käytetään hakkuiden toteuttamiseen.

Chain Final harvests

Käytetään päätehakkuiden toteuttamiseen simuloinnissa, kun kyseessä on puustoinen kuvio, mutta ei puhdas taimikko. Ensin lasketaan keinollisesti syntyneen kuvion keski-ikä, jos taimikonhoitoa tai harvennusta ei ole vielä tehty. Edelleen, jos kyseessä ei ole siemenpuu- tai kaistalehakkuukuvio, lasketaan ensin uudistamisrajat ja tulkitaan tämän jälkeen, onko kuvio uudistuskypsä. Uudistamisrajat voidaan määrittää Metsätalouden kehittämiskeskus tapion uudistussuositusten keskiläpimitta- ja ikätunnuksen ala- ja ylärajan välille. Jos kuvio on uudistuskypsä, asetetaan uudistuskypsyydestä kertovan muuttujan arvoksi 1. Päivitetään muuttujan arvo, joka

kertoo kuinka monella vuodella uudistamisraja on ylitetty. Tämän muuttujan avulla on mahdollista toteuttaa viivästetty uudistushakkuu optimointia varten. Jos uudistamiskypsyydestä kertovalla muuttujalla ei ole vielä arvoa, asetetaan arvoksi 0. Päätehakkuu tehdään, jos kuvio on metsämaata, uudistushakkuukelpoinen, valtapituus on riittävä suhteessa kasvupaikkaan ja edellisestä hakkuusta on kulunut riittävästi aikaa. Ennen hakkuuta lasketaan kuvauspuiden lukumäärä ja valtapuiden keskiläpimitta. Päätehakkuu tehdään avohakkuuna, jos kyseessä on rehevä (MT+) kivennäismaan kuvio pääpuulajista riippumatta tai vähäravinteinen (VT-) pääpuulajiltaan muu kuin mäntykuvio (todennäköisesti kasvupaikalle sopimaton puulaji). Kun hakkuu on tehty, uudistuskypsästä kuviosta kertovan muuttujan arvoksi asetetaan 0. Turvemaiden reheville (MT+) korpi tai lettokuusikoille tehdään avohakkuu. Avohakkuun jälkeen asetetaan uudistuskypsästä kuviosta kertovan muuttujan arvoksi 0. Kivennäismaiden vähäravinteisille (VT-) kuviolle ja turvemaan keskiravinteisille ja huonommille (MT-) räme- ja nevakuviolle tehdään siemenpuuhakkuu, kun pääpuulaji on mänty. Kun hakkuu on tehty, asetetaan uudistushakkuukypsästä kuviosta kertovan muuttujan arvoksi 0 ja määritetään uudistettava puulaji. Jos edelleenkään uudistuskypsää turvemaakuviota ei ole uudistushakattu epätavallisen pääpuulaji/kasvupaikkayhdistelmän vuoksi, kuvio avohakataan. Lopuksi vielä asetetaan uudistuskypsä -muuttujan arvoksi 0.

Chain Update comp_unit after clearcut or stripcut

Käytetään kuviotietojen päivittämiseen avo- tai kaistalehakkuun jälkeen, kun kyseessä on puustoinen kuvio. Samalla simulointijaksolla avohakkuun kanssa nollataan kuviontason muuttujien arvoja. Lisäksi nollataan kuoreton pohjapinta-ala. Poistetaan vielä kuviolle mahdollisesti jääneet jättöpuut. Edelleen asetetaan aukea-muuttujan arvoksi 1, kehitysluokaksi avoin_uudistusala ja kuvion puuston arvoksi nolla. Lopuksi, jos hakkuu tehtiin kaistalehakkuuna, tulee uudistamistavaksi luontainen uudistaminen.

Chain Update stratum after seedtree or sheltertree cut

Käytetään ositetietojen päivittämiseen siemen- tai suojuspuuhakkuun jälkeen, kun kyseessä on puustoinen kuvio. Jos siemen- tai suojuspuuhakkuu on tehty samalla simulointijaksolla, päivitetään ositteen pohjapinta-ala, keskiläpimitta, keskipituus, runkoluku, tilavuus, kuvauspuiden lukumäärä, ositteen puuston arvo, puutavaralajijakauma sekä bioenergiapuun määrä ja kuoreton pohjapinta-ala. Lopuksi asetetaan siemenpuuositteesta kertovan muuttujan arvoksi 1 ja asetetaan ylemmästä jaksosta kertovan muuttujan arvoksi 0.

Chain Update comp_unit after seedtree or sheltertree cut

Käytetään kuviotietojen päivittämiseen siemen- tai suojuspuuhakkuun jälkeen, kun kyseessä on puustoinen kuvio. Jos siemen- tai suojuspuuhakkuun on tehty samalla simulointijaksolla, päivitetään kuvion pääpuulaji, pohjapinta-ala, tilavuus, keskiläpimitta, keskipituus, runkoluku, koivun osuus pohjapinta-alasta, valtapuiden keskiläpimitta ja kuoreton pohjapinta-ala. Edelleen asetetaan siemenpuukuviomuuttujan arvoksi 1, nollataan kuviotason muuttujien arvoja sekä asetetaan taimettomasta siemenpuukuviosta kertovan muuttujan arvoksi 1. Uudistamistavaksi tulee luontainen uudistaminen. Edelleen päivitetään kuvion kehitysluokka. Sitä varten asetetaan ennen toimenpiteen toteutusta ollut kehitysluokka vanha_kehitysluokka -muuttujan arvoksi, eri jaksojen ikäeroksi 0 ja lasketaan uudistamisrajat. Lopuksi päivitetään kuvion puuston arvo.

Chain Remove seed trees

Käytetään siemenpuiden poistoon simuloinnissa, kun kyseessä on puustoinen kuvio. Sekä männyn siemenpuuettä kuusen suojuspuukuvioilta poistetaan siemen- ja suojuspuut, kun uudistushakkuusta on kulunut tietty aika. Jos simulointiaineistossa on alussa siemenpuukuviota, joilla on jo taimiainesta, siemenpuut poistetaan, kun kuviolla on taimiaineista tietty määrä.

Chain Update comp_unit after removing seed trees

Käytetään kuviotietojen päivittämiseen siemenpuiden poiston jälkeen, kun kyseessä on puustoinen kuvio. Samalla simulointijaksolla siemenpuiden poiston kanssa asetetaan ensin puhtaasta taimikkokuviosta kertovan muuttujan oletusarvoksi 1. Jos kuitenkin kuviolla on muitakin kuin taimikko-ositteita, asetetaan puhdas taimikko -muuttujan arvoksi 0. Siemenpuukuvio -muuttujan arvoksi asetetaan 0. Päivitetään kuvion runkoluku, kuoreton pohjapinta-ala, odotettavissa oleva runkoluku, ikä, ositteiden lukumäärä ja aritmeettinen keskipituus. Edelleen nollataan kuviotason muutujien arvoja ja päivitetään taimikkokuviosta kertovan muuttujan arvo. Jos siemenpuiden poiston jälkeen kuviolla on puhdas taimikko, päivitetään pääpuulaji, keski- ja valtapituus sekä nollataan kuviotason muuttujien arvoja. Jos jäljelle jäävä puusto ei ole puhdas taimikko, päivitetään pohjapinta-ala, pääpuulaji, tilavuus, keskiläpimitta, keski- ja valtapituus, koivun osuus pohjapinta-alasta, kuvauspuiden lukumäärä ja suhteellinen tiheys. Edelleen päivitetään valtapuiden keskiläpimitta ja latvussuhde, jos kuvion pääpuulaji on mänty. Edelleen päivitetään kuvion kehitysluokka. Sitä varten asetetaan ennen toimenpiteen toteutusta ollut kehitysluokka vanha_kehitysluokka -muuttujan arvoksi, eri jaksojen ikäeroksi 0 ja lasketaan uudistamisrajat. Lopuksi

asetetaan kuvion puuston arvoksi nolla.

Chain Thinnings

Käytetään harvennusten toteuttamiseen simuloitaville kuvioille, kun kyseessä on puustoinen kuvio. Harvennusmallia varten lasketaan leimaus- ja jäävän puuston rajat, kun kyseessä ei ole puhdas taimikko ja puuston valtapituus on riittävä suhteessa kasvupaikkaan. Harvennusmallin valinta riippuu maantieteellisestä sijainnista (Etelä-, Väli- ja Pohjois-Suomi), kasvupaikasta ja pääpuulajista. Harvennuksen leimausrajaa ja jäävän puuston rajaa on mahdollista säätää Metsätalouden kehittämiskeskus tapion uusien harvennusmallien vaihteluvälin sisällä. Jos kuvion pohjapinta-ala on suurempi kuin harvennusmallin leimausraja, päivitetään muuttuja, joka kertoo kuinka monta vuotta on kulunut harvennusrajan ylittymisestä. Tämän muuttujan avulla on mahdollista toteuttaa viivästetty harvennus optimointia varten. Harvennusmalliketjuun mennään, jos kuvion pohjapinta-ala on suurempi kuin harvennusmallin leimausraja ja on kulunut tietty aika edellisestä harvennuksesta. Tällöin ensin tehdään edeltävä raivaus tietyllä todennäköisyydellä erilaisille kohteille. Ennakkoraivaus tehdään 90 % turvemaan rehevistä (MT+) ensiharvennuskohteista. Kivennäismaalla rehevien (MT+) kuusikoiden ensiharvennuskohteista ennakkoraivataan 20 % ja männiköiden kohteista kaksi kolmasosaa. Bioenergiamallia varten asetetaan kantojen_keruu -muuttujan arvoksi 0 eli estetään kantojen nosto harvennuksessa. Jos kuviota ei ole vielä harvennettu tai jos harvennuksista ei ole tietoa ja puusto on vielä ensiharvennuskokoista, toteutetaan ensiharvennus. Toteutetun harvennuksen jälkeen asetetaan juuri harvennettu -muuttujan arvoksi 1. Edelleen, jos puusto ei ollut ensiharvennuskelpoinen, harvennus toteutetaan muuna harvennuksena. Toteutetun harvennuksen jälkeen asetetaan juuri harvennettu -muuttujan arvoksi 1. Puuston ensiharvennus-/harvennuskelpoisuus tulkitaan valtapituuden ja kasvupaikan mukaan. Harvennusvoimakkuutta ja harvennuksen kohdistusta läpimittajakauman eri kohtiin (ala-/yläharvennus) voidaan säätää harvennusmallin parametrien avulla.

Chain Update stratum after thinning

Käytetään päivitettäessä ositetason muuttujia harvennuksen jälkeen, kun kyseessä on puustoinen kuvio. Samalla simulointijaksolla harvennuksen kanssa päivitetään ositteen pohjapinta-ala, keskiläpimitta, runkoluku, kuvauspuiden lukumäärä, tilavuus ja kuoreton pohjapinta-ala. Keskiläpimitan päivityksen yhteydessä tarkistetaan, että harvennuksen yhteydessä keskiläpimitta ei ole muuttunut liikaa (käytännössä rajoitetaan harvennuksesta aiheutuvaa läpimitan kasvua alaharvennuksessa). Lopuksi lasketaan harvennuksen jälkeinen ositteen puuston arvo, puutavaralajijakauma sekä bioenergiapuun määrä.

Chain Update comp_unit after thinning

Käytetään päivitettäessä kuviotason muuttujia harvennuksen jälkeen, kun kyseessä on puustoinen kuvio. Samalla simulointijaksolla harvennuksen kanssa päivitetään kuvion pääpuulaji, pohjapinta-ala, tilavuus, keskiläpimitta, keski- ja valtapituus, runkoluku, odotettu runkoluku, koivun osuus pohjapinta-alasta, kuvauspuiden lukumäärä, valtapuiden keskiläpimitta ja kuoreton pohjapinta-ala. Edelleen nollataan harvennusrajan ylittämisestä kuluneen ajan, aika harvennuksesta ja kaksijaksoisuudesta kertovien muuttujien arvot sekä lasketaan kuvion puuston arvo. Lopuksi asetetaan bioenergiamallia varten kantojen_keruu -muuttujan oletusarvoksi 1. Tämä jälkeen asetetaan edelleen muuttujan arvoksi 0, jos kasvupaikka on kantojen nostolle sopimaton.

Silvicultural operations

Käytetään metsänhoitotoimenpiteiden toteuttamiseen.

Chain Soil preparations after regeneration harvest

Käytetään maanmuokkauksen kustannusten sekä kantojen noston bioenergiatulojen kohdentamiseen uudistushakatuille kuvioille sinä vuonna, kun uudistushakkuu on tehty. Kantojen nosto tehdään 80 % rehevän (MT+) kivennäismaan kuusikkokuvioista. Laikutus tehdään kivennäismaiden reheville (MT+) kuviolle ja yhdelle kolmasosalle turvemaiden keski- ja vähäravinteisista (MT-CT) männyn siemenpuukuvioista. Laikkumätästys tehdään turvemaiden reheville (MT+) avohakkuukuvioille. Äestys tehdään vähäravinteisille (VT-) kivennäismaakuvioille. Toteutetun maanmuokkauksen jälkeen maanmuokkauksesta kertovan muuttujan arvoksi asetetaan 1.

Chain Tending of young stands

Käytetään taimikonhoidon toteuttamiseen tietyssä puuston pituus- ja ikävaiheessa, kun kyseessä on puustoinen kuvio. Malliketju sisältää sekä varhais- että varsinainen taimikonhoidon toteutuksen. Varhaistaimikonhoito (heinäys ja perkaus) toteutetaan varhaisessa pituusvaiheessa yhdelle kolmasosalle kuusen taimikkokohteista, joilta

kannot on nostettu. Samoin 20 % VT-männiköistä tehdään varhaistaimikonhoito ennen varsinaista taimikonhoitoa. Taimikonhoito voidaan uusia tietyn ajan kuluttua edellisestä taimikonhoidosta. Kuitenkin kasvupaikalle on määritelty taimikonhoitokertojen maksimiarvo, jota ei voi ylittää. Taimikonhoidon toteutusvaihe määräytyy kasvupaikan, pääpuulajin ja puuston pituuden mukaan.

Chain Update stratum after tending of young stand

Käytetään päivitettäessä ositetietoja taimikonhoidon jälkeen samalla simulointijaksolla taimikonhoidon kanssa, kun kyseessä on puustoinen kuvio. Päivitys tehdään hieman eri tavalla puulajista riippuen. Jos kyseessä on mäntyosite ja ositteen pohjapinta-ala on suurempi kuin nolla, päivitetään runkoluku. Edelleen, jos runkoluku on taimikonhoidon seurauksena muuttunut päivitetään pohjapinta-ala, tilavuus, keskiläpimitta, keskipituus, kuvauspuiden lukumäärä ja kuoreton pohjapinta-ala. Jos kyseessä on on muu kuin mäntyosite ja ositteen pohjapinta-ala on suurempi kuin nolla, päivitetään runkoluku ja lasketaan runkolukukerroin, joka kertoo kuinka paljon runkoluku muuttui taimikonhoidon seurauksena. Edelleen, jos runkoluku on taimikonhoidon seurauksena muuttunut päivitetään pohjapinta-ala ja tilavuus kertomalla muutujan arvo runkolukukertoimella. Edelleen päivitetään kuvauspuiden lukumäärä ja kuoreton pohjapinta-ala. Lopuksi asetetaan runkoluku_mitattu -muuttujan arvoksi 1, jotta seuraavalla laskentajaksolla runkoluku pysyy samana myös jakaumallien käytön jälkeen.

Chain Update comp_unit after tending of young stand

Käytetään päivitettäessä kuviotietoja taimikonhoidon jälkeen samalla simulointijaksolla taimikonhoidon kanssa, kun kyseessä on puustoinen kuvio. Jos puutaso on olemassa, päivitetään kuvion pääpuulaji, pohjapinta-ala, kuoreton pohjapinta-ala, tilavuus, keskiläpimitta, keskipituus, runkoluku, odotettu runkoluku, ositteiden lukumäärä, koivun osuus pohjapinta-alasta, kuvauspuiden lukumäärä ja valtapuiden keskiläpimitta. Jos puutaso puuttuu, päivitetään keskipituus, runkoluku, odotettu runkoluku ja ositteiden lukumäärä. Tulkitaan ositteiden tietoja käyttäen onko kuvio kaksijaksoinen. Jos kuvio on yksijaksoinen, nollataan ylempään ja alempaan jaksoon kuuluvien ositteiden lukumäärät. Jos kuvio on kaksijaksoinen, lasketaan sekä ylempään että alempaan jaksoon kuuluvien ositteiden lukumäärät. Jos eri ositteisiin kuulumisen tulkinnan kanssa on ollut ongelmia eli ylemmän tai alemman jakson ositteiden lukumäärää ei ole laskettu (esim. kaikki ositteet kuuluvat ylempään jaksoon, siis kyseessä on yksijaksoinen kuvio), asetetaan ylempään ja alempaan jaksoon kuuluvien ositteiden lukumääräksi nolla. Tällöin myös kaksijaksoisuudesta kertovan muuttujan arvoksi asetetaan 0 eli kuvio tulkitaan yksijaksoiseksi.

Forced operations

Käytetään toimenpiteiden pakottamiseen kasvun simuloinnin yhteydessä. Pakotetut toimenpiteet saadaan metsätaloussuunnitelman toimenpide-ehdotuksista. Sisältää kaikki järjestelmällä toteutettavat toimenpiteet, niin hakkut kuin hoitotoimenpiteetkin. Jokaista toimenpidemallia varten on oma malliketjukokonaisuutensa.

Forced thinnings

Chain Calculate thinning limits

Käytetään harvennuksen leimausrajan ja harvennuksen jälkeen jäävän puuston ala- ja ylärajan määrittämiseen.

Chain Forced first thinning

Käytetään pakotetun ensiharvennuksen toteuttamiseen. Harvennusmallin valinta riippuu maantieteellisestä sijainnista (Etelä-, Väli- ja Pohjois-Suomi) ja pääpuulajista. Ennen varsinaista harvennusta asetetaan juuri harvennetusta kuviosta kertovan muuttujan arvoksi 0. Toteutetun harvennuksen jälkeen asetetaan juuri harvennettu -muuttujan arvoksi 1.

Chain Forced low thinning

Käytetään pakotetun harvennuksen toteuttamiseen alaharvennuksena. Harvennusmallin valinta riippuu maantieteellisestä sijainnista (Etelä-, Väli- ja Pohjois-Suomi) ja pääpuulajista. Ennen varsinaista harvennusta asetetaan juuri harvennetusta kuviosta kertovan muuttujan arvoksi 0. Toteutetun harvennuksen jälkeen asetetaan juuri harvennettu -muuttujan arvoksi 1.

Chain Update stratum after forced thinning

Käytetään päivitettäessä ositetason muuttujia harvennuksen jälkeen. Samalla simulointijaksolla harvennuksen kanssa päivitetään ositteen pohjapinta-ala, keskiläpimitta, keskipituus, valtapituuden kasvu, runkoluku, kuvauspuiden lukumäärä sekä tilavuus. Lopuksi lasketaan harvennuksen jälkeinen puuston arvo, puutavaralajijakauma sekä bioenergiapuun määrä.

Chain Update comp_unit after forced thinning

Käytetään päivitettäessä kuviotason muuttujia harvennuksen jälkeen. Samalla simulointijaksolla harvennuksen kanssa päivitetään kuvion pääpuulaji, pohjapinta-ala, tilavuus, keskiläpimitta, keski- ja valtapituus, valtapituus, runkoluku, odotettu runkoluku, koivun osuus pohjapinta-alasta, kuvauspuiden lukumäärä sekä valtapuiden keskiläpimitta ja latvussuhde. Lopuksi nollataan harvennusrajan ylittämisestä kuluneen ajan, aika harvennuksesta ja kaksijaksoisuudesta kertovien muuttujien arvot sekä lasketaan kuvion puuston arvo.

Chain Calculate productive value

Käytetään kuvion maan ja puuston tuottoarvon laskentaan harvennuksen jälkeen. Lasketaan puulajeittaisilla malleilla sekä paljaan maan että puuston tuottoarvo. Mineraali ja turvemaalle laskenta tehdään samalla mallilla, mutta turvemaan tuottoarvoa on laskettu kertoimella.

Forced clearcut and strip cut

Chain Forced clearcut

Käytetään pakotetun avohakkuun toteuttamiseen.

Chain Forced stripcut

Käytetään pakotetun kaistalehakkuun toteuttamiseen. Kaistalehakkuun toteuttamisen jälkeen asetetaan kaistalehakkuukuviosta ja taimettomasta luontaisen uudistamisen kuviosta kertovien muuttujien arvoksi 1.

Chain Update comp_unit after forced clearcut or strip cut

Käytetään kuviotietojen päivittämiseen pakotetun avo- tai kaistalehakkuun jälkeen. Samalla simulointijaksolla avohakkuun kanssa nollataan kuviontason muuttujien arvoja. Poistetaan vielä kuviolle mahdollisesti jääneet jättöpuut. Edelleen asetetaan aukea-muuttujan arvoksi 1, kehitysluokaksi avoin_uudistusala ja kuvion puuston arvoksi nolla.

Chain Calculate productive value

Käytetään kuvion maan ja puuston tuottoarvon laskentaan avo- tai kaistalehakkuun jälkeen. Lasketaan puulajeittaisilla malleilla sekä paljaan maan että puuston tuottoarvo. Mineraali ja turvemaalle laskenta tehdään samalla mallilla, mutta turvemaan tuottoarvoa on laskettu kertoimella.

Forced seedtree position, sheltertree, covertree position and seed/sheltertree removal

Chain Forced seedtree position

Käytetään pakotetun siemenpuuhakkuun toteuttamiseen. Ennen siemenpuuhakkuun totetuttamista lasketaan valtapuiden keskiläpimitta. Siemenpuuhakkuun jälkeen asetetaan kuvion kehitysluokaksi siemenpuumetsikkö.

Chain Forced sheltertree position

Käytetään pakotetun suojuspuuhakkuun toteuttamiseen. Ennen suojuspuuhakkuun totetuttamista lasketaan valtapuiden keskiläpimitta. Suojuspuuhakkuun jälkeen asetetaan kuvion kehitysluokaksi suojuspuumetsikkö.

Chain Forced covertree position

Käytetään pakotetun verhopuuhakkuun toteuttamiseen. Ennen verhopuuhakkuun totetuttamista lasketaan valtapuiden keskiläpimitta. Verhopuuhakkuun jälkeen asetetaan kuvion kehitysluokaksi suojuspuumetsikkö.

Chain Update stratum after forced seed, shelter or covertree cut

Käytetään ositetietojen päivittämiseen siemen-, suojus- tai verhopuuhakkuun jälkeen, kun kyseessä on puustoinen kuvio. Jos siemen-, suojus- tai verhopuuhakkuu on tehty samalla simulointijaksolla, päivitetään ositteen pohjapinta-ala, keskiläpimitta, keskipituus, runkoluku, tilavuus, kuvauspuiden lukumäärä ja ositteen puuston arvo, puutavaralajijakauma sekä bioenergiapuun määrä. Lopuksi asetetaan siemenpuuositteesta kertovan muuttujan arvoksi 1 sekä nollataan ositteen puuston arvo ja asetetaan ylemmästä jaksosta kertovan muuttujan arvoksi 0.

Chain Update comp_unit after forced seed, shelter or covertree cut

Käytetään kuviotietojen päivittämiseen siemen-, suojus- tai verhopuuhakkuun jälkeen, kun kyseessä on puustoinen kuvio. Jos siemen-, suojus- tai verhopuuhakkuu on tehty samalla simulointijaksolla, päivitetään kuvion pääpuulaji, pohjapinta-ala, tilavuus, keskiläpimitta, keskipituus, runkoluku, koivun osuus pohjapinta-alasta, valtapuiden keskiläpimitta ja latvussuhde. Edelleen asetetaan siemenpuukuviomuuttujan arvoksi 1, nollataan kuviotason muuttujien arvoja sekä asetetaan taimettomasta siemenpuukuviosta kertovan muuttujan arvoksi 1.

Edelleen päivitetään kuvion kehitysluokka. Sitä varten asetetaan ennen toimenpiteen toteutusta ollut kehitysluokka vanha_kehitysluokka -muuttujan arvoksi, eri jaksojen ikäeroksi 0 ja lasketaan uudistamisrajat. Lopuksi päivitetään kuvion puuston arvo.

Chain Assign seedtree_stratum to 1 for Upper_st stratum

Käytetään kaksijaksoisen kuvion ylemmän jakson määrittämiseen siemenpuuositteeksi.

Chain Forced remove seed trees

Käytetään pakotetun siemenpuiden poiston toteuttamiseen, kun kyseessä on kaksijaksoinen kuvio.

Chain Update comp_unit after forced removing seed trees

Käytetään kuviotietojen päivittämiseen pakotetun siemenpuiden poiston jälkeen. Samalla simulointijaksolla siemenpuiden poiston kanssa asetetaan ensin puhtaasta taimikkokuviosta kertovan muuttujan oletusarvoksi 1. Jos kuitenkin kuviolla on muitakin kuin taimikko-ositteita, asetetaan puhdas taimikko -muuttujan arvoksi 0. Siemenpuukuvio -muuttujan arvoksi asetetaan 0. Päivitetään kuvion runkoluku, odotettavissa oleva runkoluku, ikä, ositteiden lukumäärä ja aritmeettinen keskipituus. Edelleen nollataan kuviotason muutujien arvoja ja päivitetään taimikkokuviosta kertovan muuttujan arvo. Jos siemenpuiden poiston jälkeen kuviolla on puhdas taimikko, päivitetään pääpuulaji, keski- ja valtapituus sekä nollataan kuviotason muuttujien arvoja. Jos jäljelle jäävä puusto ei ole puhdas taimikko, päivitetään pohjapinta-ala, pääpuulaji, tilavuus, keskiläpimitta, keski- ja valtapituus, koivun osuus pohjapinta-alasta, kuvauspuiden lukumäärä, suhteellinen tiheys sekä valtapuiden keskiläpimitta ja latvussuhde. Edelleen päivitetään kuvion kehitysluokka. Sitä varten asetetaan ennen toimenpiteen toteutusta ollut kehitysluokka vanha_kehitysluokka -muuttujan arvoksi, eri jaksojen ikäeroksi 0 ja lasketaan uudistamisrajat. Lopuksi asetetaan kuvion puuston arvoksi nolla.

Chain Calculate productive value

Käytetään kuvion maan ja puuston tuottoarvon laskentaan siemen-, suojus- tai verhopuuhakkuun tai siemenpuiden poiston jälkeen. Lasketaan puulajeittaisilla malleilla sekä paljaan maan että puuston tuottoarvo. Mineraali ja turvemaalle laskenta tehdään samalla mallilla, mutta turvemaan tuottoarvoa on laskettu kertoimella.

Forced regeneration

Chain Forced planting

Käytetään pakotetun istutuksen toteuttamiseen. Istutuksen jälkeen asetetaan kuviotason muuttujille alkuarvoja. Edelleen asetetaan istutuksesta kertovan muuttujan arvoksi 1. Lasketaan runkoluku, ikä, pohjapinta-ala, tilavuus, kuvion odotettavissa oleva runkoluku ja pääpuulaji ositetietojen avulla.

Chain Forced seeding

Käytetään pakotetun kylvön toteuttamiseen. Kylvön jälkeen asetetaan kuviotason muuttujille alkuarvoja. Edelleen asetetaan istutuksesta kertovan muuttujan arvoksi 0. Lasketaan runkoluku, ikä, pohjapinta-ala, tilavuus, kuvion odotettavissa oleva runkoluku ja pääpuulaji ositetietojen avulla.

Chain Forced natural regeneration

Käytetään pakotetun luontaisen uudistamisen toteuttamiseen.

Chain Calculate attributes to stratum after planting or seeding

Käytetään ositetietojen täydentämiseen pakotetun istutuksen tai kylvön jälkeen. Lasketaan rinnankorkeuden saavuttamiseen kuluva aika. Arvo määräytyy sijainnin (kunta), kasvupaikan ja puulajin mukaan. Lisäksi metsämaalle sekä kitu- ja joutomaalle on omat mallinsa. Edelleen kasvatetaan istutetun ositteen keskipituutta ja ikää. Tätä varten lasketaan taimikon vuotuinen pituuskasvu.

Forced regeneration check

Chain Forced supplemental_planting

Käytetään pakotetun täydennysistutuksen toteuttamiseen. Täydennysistutuksen jälkeen asetetaan istutuksesta kertovan muuttujan arvoksi 1. Lasketaan runkoluku, ikä, pohjapinta-ala, tilavuus, kuvion odotettavissa oleva runkoluku ja pääpuulaji ositetietojen avulla.

Chain Calculate attributes to stratum after supplemental planting

Käytetään ositetietojen täydentämiseen pakotetun täydennysistutuksen jälkeen. Lasketaan rinnankorkeuden saavuttamiseen kuluva aika. Arvo määräytyy sijainnin (kunta), kasvupaikan ja puulajin mukaan. Lisäksi metsämaalle sekä kitu- ja joutomaalle on omat mallinsa. Edelleen kasvatetaan istutetun ositteen keskipituutta ja ikää. Tätä varten lasketaan taimikon vuotuinen pituuskasvu.

Chain Forced grass suppression

Käytetään pakotetun mekaanisen heinäntorjunnan toteuttamiseen.

Chain Forced chemical grass suppression

Käytetään pakotetun mekaanisen heinäntorjunnan toteuttamiseen.

Forced soil preparation

Chain Forced scarification

Käytetään pakotetun maanmuokkauksen toteuttamiseen laikuttamalla. Maanmuokkauksen toteuttamisen jälkeen asetetaan maanmuokkauksesta kertovan muuttujan arvoksi 1.

Chain Forced mounding

Käytetään pakotetun maanmuokkauksen toteuttamiseen laikkumätästämällä. Maanmuokkauksen toteuttamisen jälkeen asetetaan maanmuokkauksesta kertovan muuttujan arvoksi 1.

Chain Forced harrowing

Käytetään pakotetun maanmuokkauksen toteuttamiseen äestämällä. Maanmuokkauksen toteuttamisen jälkeen asetetaan maanmuokkauksesta kertovan muuttujan arvoksi 1.

Forced tending and management of seedling and young stands

Chain Forced tending of young stands

Käytetään pakotetun taimikonhoidon toteuttamiseen. Jos kuviolla on olemassa puita, taimikonhoito tehdään aidosti puita poistamalla, muuten kyseessä on taimikon perkaus.

Chain Forced early tending of young stands

Käytetään pakotetun varhaistaimikonhoidon toteuttamiseen. Jos kuviolla on olemassa puita, varhaistaimikonhoito tehdään aidosti puita poistamalla, muuten kyseessä on taimikon perkaus.

Chain Forced cleaning

Käytetään pakotetun perkauksen toteuttamiseen.

Chain Forced improvement_of_young_stand

Käytetään pakotetun nuoren metsän hoidon toteuttamiseen.

Chain Update stratum after forced tending or improvement of young stand

Käytetään päivitettäessä ositetietoja taimikonhoidon tai nuoren metsän kunnostuksen jälkeen samalla simulointijaksolla. Jos ositteen pohjapinta-ala on suurempi kuin nolla, päivitetään pohjapinta-ala, keskiläpimitta, keskipituus, runkoluku, tilavuus ja kuvauspuiden lukumäärä.

Chain Update comp_unit after forced tending or improvement of young stand

Käytetään päivitettäessä kuviotietoja taimikonhoidon tai nuoren metsän kunnostuksen jälkeen samalla simulointijaksolla. Jos puutaso on olemassa, päivitetään kuvion pääpuulaji, pohjapinta-ala, tilavuus, keskiläpimitta, keskipituus, runkoluku, odotettu runkoluku, ositteiden lukumäärä, koivun osuus pohjapinta-alasta, kuvauspuiden lukumäärä, valtapuiden keskiläpimitta ja latvussuhde. Jos puutaso puuttuu, päivitetään keskipituus, runkoluku, odotettu runkoluku ja ositteiden lukumäärä. Tulkitaan ositteiden tietoja käyttäen onko kuvio kaksijaksoinen. Jos kuvio on yksijaksoinen, nollataan ylempään ja alempaan jaksoon kuuluvien ositteiden lukumäärät. Jos kuvio on kaksijaksoinen, lasketaan sekä ylempään että alempaan jaksoon kuuluvien ositteiden lukumäärät. Jos eri osittеisiin kuulumisen tulkinnan kanssa on ollut ongelmia eli ylemmän tai alemman jakson ositteiden lukumäärää ei ole laskettu (esim. kaikki ositteet kuuluvat ylempään jaksoon, siis kyseessä on yksijaksoinen kuvio), asetetaan ylempään ja alempaan jaksoon kuuluvien ositteiden lukumääräksi nolla. Tällöin myös kaksijaksoisuudesta kertovan muuttujan arvoksi asetetaan 0 eli kuvio tulkitaan yksijaksoiseksi.

Forced clearing

Chain Forced clearing

Käytetään pakotetun raivauksen toteuttamiseen.

Chain Forced clearing_before_harvest

Käytetään pakotetun ennakkoraivauksen toteuttamiseen.

Forced pruning

Chain Forced pruning

Käytetään pakotetun pystykarsinnan toteuttamiseen.

Forced ditching

Chain Forced ditch cleaning

Käytetään pakotetun kunnostusojituksen toteuttamiseen.

Forced fertilization

Chain Forced fertilization

Käytetään pakotetun kasvatuslannoituksen toteuttamiseen.

Chain Forced vitality fertilization

Käytetään pakotetun terveyslannoituksen toteuttamiseen.

CHAPTER

TEN

KNOWN ISSUES WITH SIMO INSTALLATION

10.1 On windows, missing dlls (msvcr*.dll) or incorrect Side-by-side configuration error

It's possible that you get an error complaining about missing dlls (namely, one that starts with msvcr) or "Side-by-side configuration is incorrect" ("Rinnakkaismääritys on virheellinen" if you're Finnish). This should only happen in situations where the Visual C++ 2008 Redistributable package has not been installed on your system.

10.1.1 Solution:

(Re)Install the Visual C++ 2008 Redistributable package.

10.2 Problem with Twisted installation:

It may happen that your system doesn't have libraries needed to build the bzip2 support into SIMO's Python. In that case the buildout installation of SIMO will terminate at Twisted installation with the message:

```
Error: Couldn't install Twisted 8.2.0
```

(or whatever the Twisted version might be when you're installing). A fex lines above you'll see something like this:

```
error: Not a recognized archive type: /tmp/tmpV4NoSrget_dist/Twisted-8.2.0.tar.bz2
```

10.2.1 Solution:

You need to install development version of bzip2 and rebuild the Python used by SIMO. On Ubuntu:

```
sudo apt-get install libbz2-dev
rm -rf parts/python-2.6
python build_env.py
```

and everything should finish cleanly.

On systems using the rpm based package management, try:

```
sudo yum install bzip2-devel
```

for the first step.