The NED Forest Management DSS: The Integration of Growth and Yield Models

by

Astrid Glende

(Under the direction of Donald Nute)

Abstract

NED-2 is a robust, intelligent, goal-driven ecosystem management decision support system that integrates a wide variety of modeling tools. These tools include vegetation growth and yield models, wildlife models, silvicultural models, GIS, and visualization tools.

Integrating growth and yield models is one of the key elements in the NED-2 system. Without its integration goal analysis for projected data is impossible. This paper addresses implementation issues for FVS, the simulation model currently included in NED-2

Index words: Decision support systems, ecosystem management, FVS, knowledge based systems, NED, Prolog, treatment definition

The NED Forest Management DSS: The Integration of Growth and Yield Models

by

Astrid Glende

Vordiplom, The University of Koblenz Germany, 2000

A Thesis Submitted to the Graduate Faculty

of The University of Georgia in Partial Fulfillment

of the

Requirements for the Degree

Master of Science

Athens, Georgia

2004

The NED Forest Management DSS: The Integration of Growth and Yield Models

by

Astrid Glende

Acknowledgments

I thank Dr. Donald Nute, my major professor, and Drs. Walter D. Potter and Allen Partridge for devoting their time to proofreading my thesis.

I thank Drs. H. Michael Rauscher and Mark J. Twery, and Scott Thomasma and Pete Knopp - members of the the Forest Service - for their support during the work on the NED project.

A special thanks to the U.S.D.A. Forest Service who made my studies at the AI Center possible by funding me.

I also like to thank my friends and classmates in the AI center - especially Abhishek, Jason, Chris, Sarah, Mayukh, John, Georgia, Emily, Laura, Manuela, Pascal, Markus, Jonny, and Andi - for helping me to stay calm.

Thanks to my family members; without their support I would have not got this far in my life.

This work was funded by U.S.D.A. Forest Contract 01-CA-11242343-009.

Table of Contents

Page

Acknowledgments iv

List of Figures vii

Chapter

1 An Introduction To NED 1

1.1 Overview 1

1.2 The Decision Process in NED 1

1.3 The NED-1 Architecture 2

1.4 The NED-2 Architecture 4

2 The Integration of Growth and Yield Models 7

2.1 A Basic Description of FVS 7

2.2 An Overview of the Simulation Process 8

2.3 The NED-2 Treatment Development Agent 11

2.4 The NED-2 Treatment Set Selection Agent 20

2.5 The NED-2 Simulation Agent 22

3 Dialog Related Problems and Solutions 41

3.1 Screen Resolution 41

3.2 Disappearing Dialogs 42

4 Debugging 44

5 Conclusion and Future Directions 45

Appendix

A An Index of Predicates 47

A.1 trt_edit.dcm 47

A.2 trtparam_wnd.dut 51

A.3 trtimport_wnd.dut 63

A.4 trt_sel.dcm 65

A.5 simulator.dcm 67

A.6 fvs_bulletproof.dut 78

A.7 mdb2fvs.dut 84

A.8 fvs2mdb.dut 89

Bibliography 94

List of Figures

1.1 NED-1 Architecture 3

1.2 NED-2 Architecture 5

2.1 Develop Treatment Plans Dialog 10

2.2 Create/Edit Treatment Sets 12

2.3 Parameter Settings 14

2.4 Treatment Icon Selection 16

2.5 Treatments Table 21

2.6 Treatment Set Selection 22

2.7 Scenario Designs Table 24

2.8 Program Execution: Baseline Generation 28

2.9 Program Execution: Plan Simulation 30

2.10 The Tree Data File 33

2.11 The Keyword File 35

2.12 The Tree List File 38

Chapter 1

An Introduction To NED

1.1 Overview

NED-2 is a decision support system (DSS) for ecosystem management. It was developed through collaboration between the USDA Forest Service and the Artificial Intelligence Center of the University of Georgia. The idea of the project arose in 1987 with the motive to design a single, easy-to-use software system which would provide an interface for independently developed forest management tools. The objective was to support forest managers to develop goals, to assess current and future conditions, and to produce sustainable management plans for forest properties [12].

A management unit (MU) is a piece of forested land that combines several stands. Each stand is a contiguous section of land representing a single forest type. A management plan for a forest consists of overall goals for the entire management unit and decisions about how each stand in the unit is to be managed. The management plan is used to recommend short-term treatments for individual stands in the forest [7].

1.2 The Decision Process in NED

The *decision process for NED* is described as follows. Usually a forest manager starts with acquiring and entering the data of the management unit. This data include the location and number of stands in the forest. For each stand inventory data (i.e., size, shape, species and size classes of trees in overstory, overstory closure, understory density, presence or absence of water, etc.) are needed. Subsequently, the management objectives for the management unit need to be identified. The objectives supported by NED are timber, wildlife, water, ecological, landscape, and recreational. The next important step in the decision process is goal satisfaction analysis to determine how well the forest currently satisfies the objectives. But goal analysis can also be performed later in the process on any scenario generated for the forest to see how well it will satisfy the goals. Involved with goal satisfaction analysis is goal conflict analysis. This determines whether it is possible to achieve all objectives set in the previous step. Any goal conflicts should be resolved before advancing to the following step. Next, the forest manager should receive recommendations about how to manage the forest in order to reach the desired stand structure. For those stands that have already reached the desired structure, silvicultural methods will be assigned to generate specific treatment recommendations. These recommendations include when and how to regenerate, thin, fertilize, and harvest a stand. These previously described steps do not necessarily require a certain order. The manager always has the option to return to an earlier step in the process and to change the parameters [7].

This describes the original design for NED; not all of these steps have been implemented.

1.3 The NED-1 Architecture

NED-1 is the first release of NED. It supports the functionality of data acquisition and analysis, goal selection, and goal satisfaction analysis for current inventory. The architecture of NED-1 envisions a subdivision of the front-end functionalities, which are programmed in C++, and the inference server Prolog application (see Figure 1.1). The front-end primarily consists of the user interface, the data engine, the report generator, and the model manager.

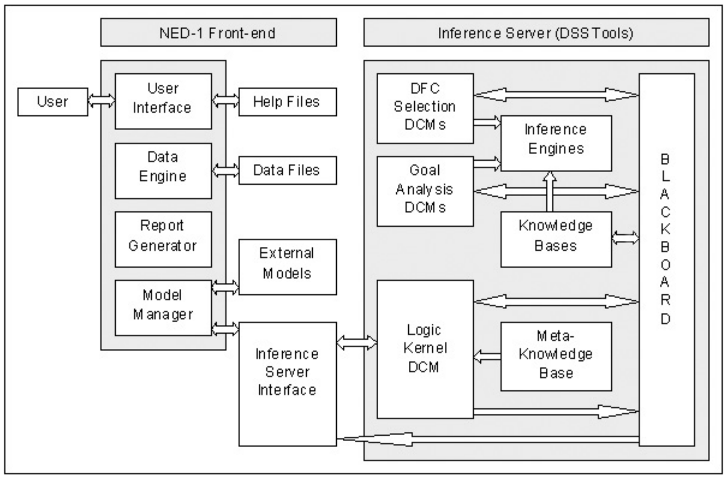


Figure 1.1: NED-1 Architecture

The NED-1 front-end communicates with Prolog through an Inference Server Interface. The Prolog component consists of one or more knowledge bases, several domain control modules (DCMs), inference engines, and a blackboard. The DCMs are semi-autonomous agents which means that they have limited knowledge about the tasks other agents perform and what information they need. They have access to the blackboard and can call any inference engine available in the Prolog component. The blackboard is a global storage area for user input information, inference results, and task requests. DCMs do not communicate directly with each other. Instead, they write their results to the blackboard where every DCM has access to them. At each given time only one DCM is active. After the termination of a DCM, each DCM has the chance to examine the blackboard. The first DCM whose activation conditions are satisfied becomes active and performs its assignment [7].

In the case of a user wanting to perform a task, the NED-1 front-end program sends a request to the Logic Kernel DCM. All management unit data are stored within the NED-1 front-end program. Therefore, the Logic Kernel DCM has to consult its metaknowledge base to determine which knowledge base contains the facts and rules required to complete the request. It loads this knowledge base and places a request on the blackboard. The appropriate DCM sees the request and performs the necessary calculations. Since the inference server does not have access to the forest data it requests this information from the NED-1 front-end program as needed. The NED-1 front-end is informed as soon as the task is complete. It can then call the Inference Server Interface to read the results from the blackboard. As soon as the Logic Kernel DCM receives a new request, it deletes all data from the blackboard to ensure that previous inferences will have no affect [7].

1.4 The NED-2 Architecture

Many of the NED-2 components are still written in C++, but different from NED-1 the Prolog components control the application. Execution of the C++ modules (DLLs) are managed using the *NEDpnp.dll*. User interactions are sent via messages from the PnP interface to Prolog. In response, Prolog can notify the PnP which of the other modules to run [5].

To keep the system flexible and open for future extensions, the NED-2 architecture is distributed. NED-2 is a community of intelligent, semi-autonomous agents each of which know how to use a class of decision support tools. A blackboard architecture is used to coordinate the behaviors and communication of the agents. Unlike object oriented architectures, agents in a blackboard architecture do not need to know what tasks other agents perform or what information other agents need in order to perform those tasks [8] and [9]. The blackboard consists of an MS Access database, and Prolog clauses as depicted in fig. 1.2.

Since a stand is not static, the information about a stand is stored as a snapshot representing a stand at a point in time under a particular treatment regime.

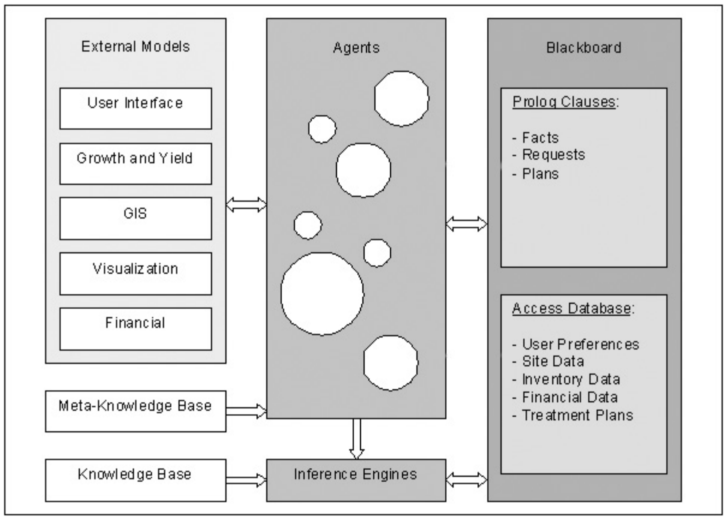


Figure 1.2: NED-2 Architecture

Persistent information is stored either in the database or in Prolog knowledge bases. Temporary information, represented as facts, is stored as a set of Prolog clauses on the blackboard. Facts are represented as Attribute-Object-Value (AOV) triples. For example, the triple:

```
tpa([snapshot(17),species(oak),size(6)],24)
```

indicates that there are 24 six-inch oaks per acre in snapshot 17.

Facts, either stored as a record of the database or as a Prolog clause, can be retrieved from the blackboard by using the Prolog query:

```
known(Attribut(Object,Value)).
```

This predicate first tries to satisfy the request by looking for a corresponding Prolog fact. If this does not exist, it proceeds to determine if the information is stored in the NED-2 database.

Requests are also stored on the blackboard as Prolog clauses. The request

```
request([fvs_baseline])
```

requests that the NED-2 simulation agent initiates the generation of the baseline [9].

To date, the following agents are implemented:

- NED-2 interface agent
- NED-2 treatment development agent
- NED-2 treatment set selection agent
- NED-2 simulation agent
- NED-2 goal analysis agent
- NED-2 GIS agent
- NED-2 report writer agent
- NED-2 planning agent

In the following chapter the implementation of the NED-2 treatment development agent, the NED-2 treatment set selection agent, and the NED-2 simulation agent will be discussed in detail. The description is based on the currently released beta-version of NED-2.

Chapter 2

The Integration of Growth and Yield Models

To assist the decision making in forest ecosystem management a substantial and integral part of the decision process is the integration of growth and yield models. A forest growth and yield model represents the development of tree crops as they increase in age, or as time changes. Therefore, the ability to project inventory data to any future scenario, lets goal satisfaction analysis be performed at any point in time, instead of being restricted to the time of the inventory.

The simulation model currently supported by NED-2 is the Forest Vegetation Simulator (FVS) utilized nationally by the USDA Forest Service. Presently, NED-2 integrates the Northeastern variant, based on NE-TWIGS [11], and the Southern variant. We envision that other growth simulators, including SILVAH [6], will be incorporated in the future.

Previous work on the integration of FVS has been done by Jin Wang and is described in [14].

2.1 A Basic Description of FVS

As described in [2], FVS requires two input files in order to run, a keyword file and a tree data file. The keyword file (*.key) controls the simulation. It describes any treatments to be simulated, controls the printing of output, computes custom variables, and adjusts model estimates. It is also used to enter stand level parameters (e.g., slope, aspect, elevation, sampling design specifications, location such as forest and district, and site productivity). The tree data file (*.fvs) contains tree level information. For each tree record, species and diameter at breast height (dbh) are required. Optionally, tree count, diameter growth, height, height growth, crown ratio, and various other inventory data can be specified.

Up to four output files can be created by FVS. The main output file (*.out) contains information about keyword interpretation and scheduled activities, information about model calibration to the input data, stand composition statistics through time, individual sample trees through time, a stand summary table of the entire simulation, and a summary of the management activities that were simulated. The tree list file (*.trl) gives detailed information about all the individual tree records being projected. The third file (*.sum) outputs the summary table information from the main output file only. Information needed when running the economics model linked to FVS can be processed to a fourth output file (*.chp). The main output file is generated automatically while the last three files are optional and must be requested via keywords in the keyword file.

In FVS the length of time over which simulation results are desired is specified in terms of “cycles”. A cycle is a period of time for which increments of tree characteristics are predicted. All silvicultural treatments for a stand scheduled within a cycle occur at the beginning of the cycle. Growth and mortality are based on post-treatment conditions.

2.2 An Overview of the Simulation Process

The simulation process in NED-2 is usually composed of:

1. Defining treatments
2. Selecting a set of treatment definitions for a management unit
3. Generating a baseline
4. Developing one or more treatment plans

5. Simulating treatment plans

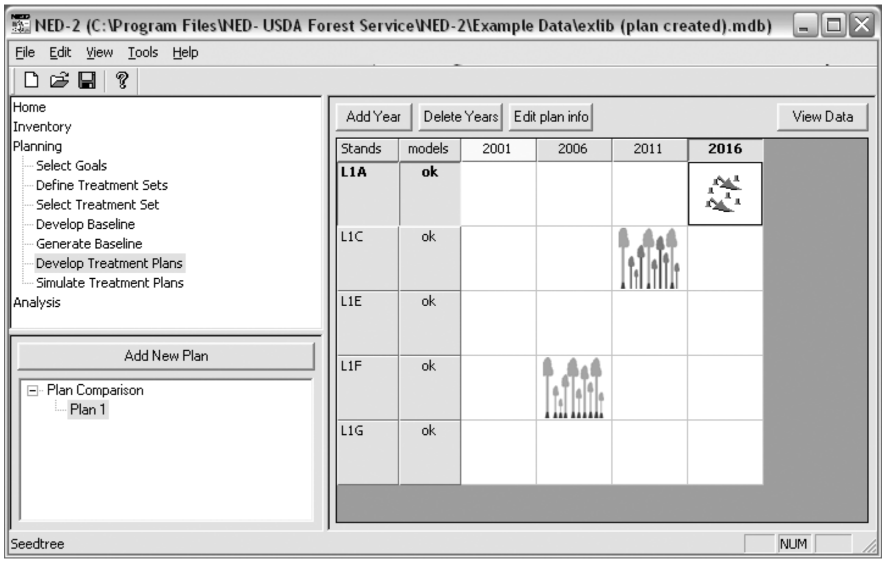
Treatments in FVS are controlled by setting the values of the parameters of a keyword. To achieve the desired outcome of a treatment, the user needs to be able to access and modify the default values of these parameters. This is accomplished by the NED-2 treatment development agent.

Some circumstances require the user to maintain multiple treatment definition sets (e.g., users managing more than one management unit). For this reason it is necessary to offer a comfortable way to associate the appropriate treatment definition set with the current management unit. This is done by the NED-2 treatment set selection agent.

Typically inventory for different stands are taken in different years. But a design decision, made by the NED team, requires that all plans start in a given year. The motive behind this is to simplify the plan development dialog, goal analysis at the stand level, and plan comparison. This makes the generation of a baseline necessary which is achieved by the NED-2 simulation agent.

Before treatment plans can be simulated, they need to be developed within the *Plan Development* dialog (see fig. 2.1). This dialog allows the user to specify the length of each cycle, which is defined by the difference of years between consecutive columns. By double-clicking a cell the user can choose the silvicultural treatments from the set of defined treatments associated with the management unit. The treatment will take effect in the year and for the stand the cell is linked to [13]. There are two cases where adding a treatment is problematic and therefore, a design choice has been made to prevent the user from the following:

1. Adding a clearcut to the **baseline year** in the *Baseline Development* process: If a clearcut is added to the baseline year for *baseline generation*, all trees



Stands	models	2001	2006	2011	2016
L1A	**ok**				🌲
L1C	ok			🌲	
L1E	ok				
L1F	ok		🌲		
L1G	ok				

Figure 2.1: Develop Treatment Plans Dialog

will be removed from a stand. Therefore, there will be no trees to grow when simulating treatment plans.

2. Adding a treatment to the **baseline year** in the *Plan Development* process: Treatments in FVS are attempted in the first year of a cycle. Then the stand grows for all years in the cycle, including the year in which the treatment is performed. In other words, a stand grows for at least one year. Adding a treatment to the baseline year for *treatment plan simulation* would automatically grow the stand after the treatment is performed. Since the baseline year overlaps *baseline generation* and *treatment plan simulation*, growth has already happened when generating the baseline. So if treatments were allowed in treatment plans during the baseline year, the affected stand would be grown twice for that year.

Additionally, the user is required to select the simulation models for each stand by double-clicking the cells in the *models* column. Without this information the NED-2 simulation agent does not know which simulator to run.

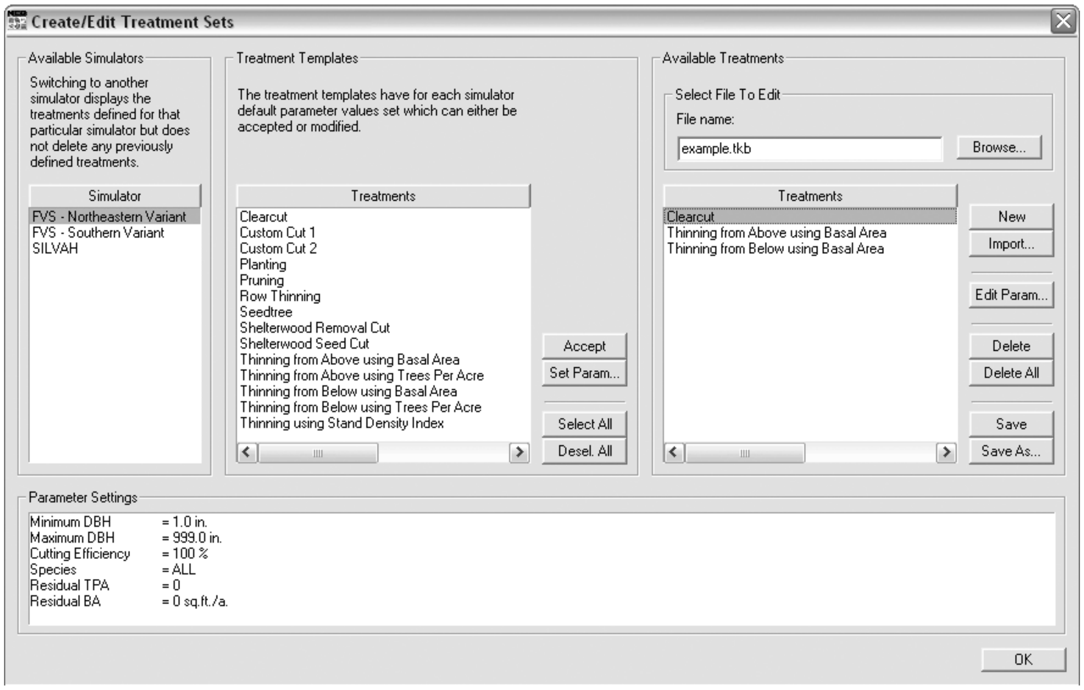
As soon as the user triggers another action in the *Navigation Pane* of the NED-2 interface the plan information is saved in the [Scenario_designs] table of the management unit database.

Subsequently, the simulation process can be started for any chosen plan. When the NED-2 simulation agent is activated, the necessary data to build the two input files are retrieved from the management unit database and the FVS program is launched through a batch file. Finally, the projected tree data output by FVS has to be read back into the database.

2.3 The NED-2 Treatment Development Agent

Silvicultural treatments play an essential part in the development of NED-2 treatment plans. But before treatments can be added to a treatment plan, the user needs to create a treatment knowledge base (stored in a *.tkb file), that includes the definitions of available treatments. The NED-2 treatment development agent assists the user in this process. Consulting a default knowledge base (*trt_def.kb*) the agent offers the treatments specific to each simulator to the user in an easy-to-use interface (see fig. 2.2).

Selecting a simulator in the *Create/Edit Treatment Sets* dialog, displays the treatment templates and if applicable the user defined treatments for that simulator. The parameters of a selected treatment are displayed in the *Parameter Settings* listbox. This allows the user to review the parameter settings before accepting or modifying them. With the *Import...* function, users responsible for managing more than one



Switching to another simulator displays the treatments defined for that particular simulator but does not delete any previously defined treatments.

Simulator
FVS - Northeastern Variant
FVS - Southern Variant
SILVAH

The treatment templates have for each simulator default parameter values set which can either be accepted or modified.

Treatments
Clearcut
Custom Cut 1
Custom Cut 2
Planting
Pruning
Row Thinning
Seedtree
Shelterwood Removal Cut
Shelterwood Seed Cut
Thinning from Above using Basal Area
Thinning from Above using Trees Per Acre
Thinning from Below using Basal Area
Thinning from Below using Trees Per Acre
Thinning using Stand Density Index

File name: example.tkb

Treatments
Clearcut
Thinning from Above using Basal Area
Thinning from Below using Basal Area

Minimum DBH	= 1.0 in.
Maximum DBH	= 999.0 in.
Cutting Efficiency	= 100 %
Species	= ALL
Residual TPA	= 0
Residual BA	= 0 sq.ft./a.

Figure 2.2: Create/Edit Treatment Sets

management unit can import treatment definitions developed for other management units.

The treatment definitions are saved by default in the *User KBs* folder under the root directory for NED-2 in a *.tkb file. The format in which the treatments are stored depends on the simulator the treatments are defined for. Since the FVS simulator encodes treatments in a keyword record consisting of a keyword and its parameters within the *.key file, the format for the treatment (stored in a user defined *.tkb file) is as follows:

```
treatments(fvs(+Variant),+TREATMENTID,+Icon,+Description,+TrtType,
    [[keyword(+KEYWORD_1),
      field(+FieldNr_1, +ParamName, +ParamValue)
      ⋮
```

```
 field(+FieldNr_n, +ParamName, +ParamValue)
],
 ⋮
[keyword(+KEYWORD_n),
 field(+FieldNr_1, +ParamName, +ParamValue)
 ⋮
 field(+FieldNr_n, +ParamName, +ParamValue)]]).
```

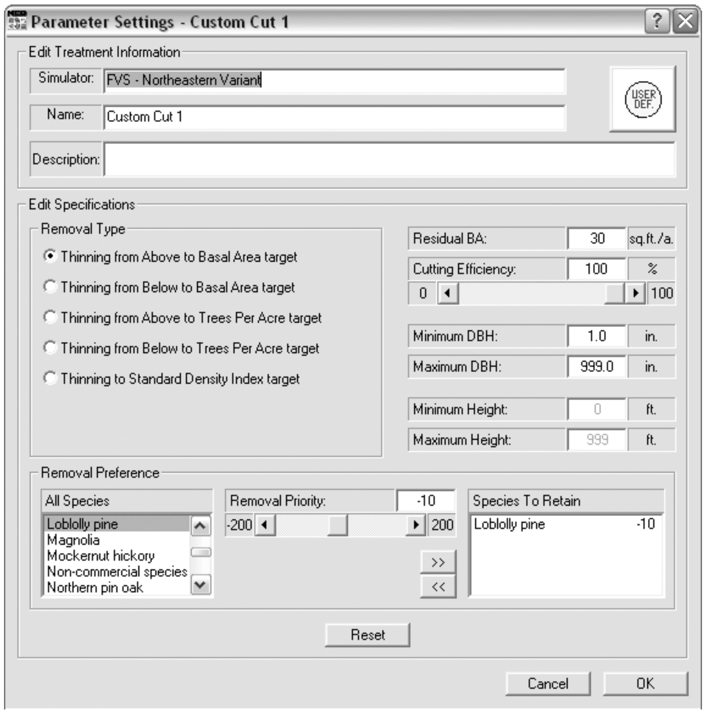
The formats for the default treatments, which are stored in *NED-2/prolog/kbs/ trt_def.kb* are essentially the same. Basically, the only difference is the predicate name which is **def_treatment/6**.

When a *.tkb file is associated with the current management unit, a link to each treatment in the file is stored in the [Treatments] table of the management unit database. This link is used later on in the NED-2 decision process to add treatments to the treatment plans.

2.3.1 Defining Treatments

Creating The Window

To set or edit the parameters of a selected treatment, the NED-2 treatment development agent provides a self-explanatory dialog (see fig. 2.3). This dialog displays the parameters particular to that treatment. But each treatment consists of different parameters. Therefore, the appearance of the *Parameter Settings* dialog is individual for each treatment. To create the window *on the fly* the agent consults a metaknowledge base and selects the layout of the window depending on the treatment type stored in the knowledge entry for the treatment. The format for the window type predicate, which is defined in trt_def.kb, is as follows:


Parameter Settings - Custom Cut 1

Edit Treatment Information

Simulator:	FVS - Northeastern Variant
Name:	Custom Cut 1
Description:	

Edit Specifications

Removal Type
- (●) Thinning from Above to Basal Area target
- ( ) Thinning from Below to Basal Area target
- ( ) Thinning from Above to Trees Per Acre target
- ( ) Thinning from Below to Trees Per Acre target
- ( ) Thinning to Standard Density Index target

Residual BA:	30	sq.ft./a.
Cutting Efficiency:	100	%
Minimum DBH:	1.0	in.
Maximum DBH:	999.0	in.
Minimum Height:	0	ft.
Maximum Height:	999	ft.

Removal Preference

All Species: Loblolly pine, Magnolia, Mockernut hickory, Non-commercial species, Northern pin oak

Removal Priority: -10 (-200 to 200)

Species To Retain	
Loblolly pine	-10

Figure 2.3: Parameter Settings

```
wnd_type(+TrtType, +Width, +Height,
    [[+ParamName, +EditableFlag, +OffsetX, +OffsetY],
     ⋮
     [+ParamName, +EditableFlag, +OffsetX, +OffsetY],
     ['Reset', _, +OffsetX, +OffsetY],
     ['Cancel', _, +OffsetX, +OffsetY],
     ['OK', _, +OffsetX, +OffsetY]]).
```

For each *ParamName* in the **wnd_type/4** predicate, which is equivalent to one parameter in the treatment, the agent consults the parameter template predicate, also defined in trt_def.kb, to create the control windows:

```
param_template(+ParamName,+ValueRange,+Dimension,
[[+WndClass_1,+Text,+OffsetX,+OffsetY,+Width,+Depth,+Style],
  ⋮
 [+WndClass_n,+Text,+OffsetX,+OffsetY,+Width,+Depth,+Style]]).
```

For example, the parameter **Minimum DBH** consists of a static window displaying the parameter name, an edit window taking values in the range specified in *ValueRange*, and another static window denoting the unit.

Once the control windows are created the default values need to be inserted. There is no particular predicate concerned with this task. The predicate responsible for this task depends on the control window type and the parameter name.

Windows Handling

Since the windows are created on the fly, the numbers for the window handlers cannot be predefined. Whenever a control window is created, the number associated with this window is stored in a list. Window handler numbers concerned with one parameter are grouped together and stored within the predicate **hndl/2**.

To handle windows messages, the predicate **hndl_paramdef/4** needs to be asserted when the window is created:

```
assert((hndl_paramdef((param_def,+Hndl),Message,Data,Result) :-
    predicate))
```

In the case of a window handler receiving a windows message, the handler number of this window has been obtained. This can be passed to the predicate handling the window, where the *handler list* for a parameter can be obtained through the **param_get_hndllist/2** predicate for further processing:

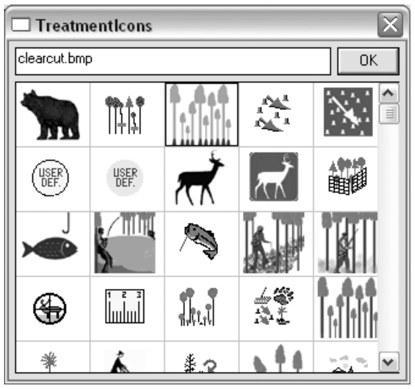


Figure 2.4: Treatment Icon Selection

```
param_get_hndllist(Hndl,HndlListCur) :-
    findall(
          HndlList,
          (
           hndl(_,HndlList),
           member(Hndl,HndlList)
          ),
          List
         ),
    List = [HndlListCur|_].
```

This predicate finds all handler lists stored in memory in which the current handler is a member. The first list found is then returned.

Treatment Icon Selection

An appropriate icon for a treatment can be selected from the icon library shown in fig. 2.4. This icon is displayed in a matrix during plan development, making it easy for the user to understand a plan at a glance. The icon selection dialog can be accessed by clicking the image button in the *Parameter Definition* dialog. The call to the icon library is performed by a dynamic link library (DLL) written in C++.

Prolog provides the command **winapi/3** to execute external WIN32 API or 32-bit DLL functions.

Before a DLL-function can be called the particular DLL has to be loaded. This is done by the 'Load LibraryA' function provided by the *kernel.dll*. This function takes the path to the DLL as an input parameter and returns a handle to the loaded DLL. Then the functions of the loaded DLL can be executed. The *PickTreatmentIcon.dll* has one function, named 'PickTreatmentIcon'. This function takes a buffer as its input parameter in which it returns the icon file name. For this reason, a read/write buffer needs to be created with the **fcreate/5** predicate. If the call to the PickTreatmentIcon.dll is successful, the file name can be read out of the buffer with the **wintext/4** command. Afterwards the DLL needs to be freed, which is done by the 'FreeLibrary' function from the *kernel.dll*. This function takes the handle of the DLL to be freed as input. Following are excerpts from the **param_set_icon/0** predicate, which is responsible for the execution:

```
param_set_icon :-
    ⋮
    fcreate(iconfile,[],-2,256,0),
    winapi((kernel32, 'LoadLibraryA'), [DLLPATH], Handle),
    winapi(('PickTreatmentIcon','PickTreatmentIcon'),[iconfile],R),
    winapi((kernel32,'FreeLibrary'),[Handle],_),
    (
     R = 1,
     wintxt(iconfile,0,0,File),
     ⋮
    )
    fclose(iconfile).
```

The Help System

The *Parameter Definition* dialog incorporates a help system. This is achieved by including the WS_ES_CONTEXTHELP style to the style list for that dialog. This creates the **?-button** in the upper right corner of the window. When pushed, clicking on any of the control windows within the dialog displays a context help specific to the parameter the control window is associated with.

For each control window a handler handling the **msg_help** message is asserted which calls a predicate distinct to each parameter:

```
assert((hndl_paramdef((param_def,Hndl),msg_help,(HX,HY),_) :-
    param_help(ParamName,HX,HY))),
```

This way, the proper context help text for the parameter is selected using the **help_txt/3** predicate.

The Creation of Metaknowledge

As mentioned above, a silvicultural treatment in FVS consists of a keyword and its parameters. To translate standard and customized treatments used in forestry to the FVS keyword system, NED-2 has developed so-called treatment templates. For example, a **Clearcut** can be achieved in FVS with the THINDBH keyword. This keyword has the following parameters:

```
    field 1:    Year or cycle that thinning is scheduled
    field 2:    Smallest dbh to be considered for removal
    field 3:    Largest dbh to be considered for removal
    field 4:    Cutting efficiency parameter specific to
                this thinning request
    field 5:    Species code for trees to be removed
    field 6:    Target trees per acre for the segment of
                the dbh distribution of the species
    field 7:    Target basal area for the segment of the
                dbh distribution of the species
```

Metaknowledge for a treatment template has to consider all knowledge essential to a treatment. The simulator model for which the treatment is defined, the treatment name, the icon associated with the treatment, a treatment description, and the treatment type[1] are necessary to generate the *Parameter Settings* dialog. A keyword list includes knowledge about each parameter: the field number in FVS and its default value. The information contained in the keyword list is used by the NED-2 simulation agent. The metaknowledge for a treatment template is incorporated in the **def_treatment/6** predicate.

Each treatment template is matched to a different set of FVS keywords. Keywords can have up to seven parameters. For this reason, the window size of the *Parameter Definition* dialog varies depending on the treatment. To deal with the unknown, information about `Width` and `Height` have to be maintained. As mentioned, the treatment parameters differ for each treatment. Therefore, information about which parameters and where to place each of them within the dialog has to be stored. The **wnd_type/4** predicate is used to store this metaknowledge.

When displayed in the *Parameter Definition* dialog, each parameter is represented by several control windows. The metaknowledge for the control window information for each parameter (i.e., class, label, left, top, width, depth, style) is maintained in the **param_def/4** predicate.

In more complex treatment templates, like **Custom Cut 1** and **Shelterwood Seed Cut**, radio buttons control which thinning type is being defined. Since the thinning types have different parameters, the window's layout changes when switching to another thinning type. The **custom/2** predicate (defined in the trt_def.kb file) deals with this by retaining the layout of each parameter in a list. The thinning types for which this predicate is defined are thinning from above using basal area

[1]A treatment is identified by its type instead of its name because treatments can be renamed by the user.

(thinaba), thinning from above using trees per acre (thinata), thinning from below using basal area (thinbba), thinning from below using trees per acre (thinbta), and thinning using stand density index (thinsdi):

```
custom(+Type,[[ParamName,EditFlag,OffsetX,OffsetY],...]
```

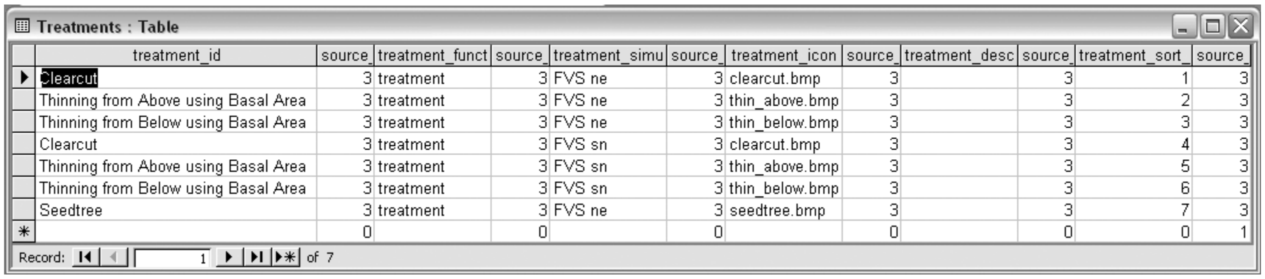
A **def_treatment/6** or **treatment/6** predicate maintains the parameter values for only one thinning type. The parameter default values for the remaining thinning types can be retrieved through the **defaults/4** predicate, also defined in the trt_def.kb file:

```
defaults(+Type,+Icon,+Description,
    [[keyword(+KEYWORD_1),
      field(+FieldNr_1, +ParamName, +ParamValue)
      ⋮
      field(+FieldNr_n, +ParamName, +ParamValue)
     ],
      ⋮
     [keyword(+KEYWORD_n),
      field(+FieldNr_1, +ParamName, +ParamValue)
      ⋮
      field(+FieldNr_n, +ParamName, +ParamValue)]]).
```

2.4 The NED-2 Treatment Set Selection Agent

Selecting a treatment set (*.tkb file) for a management unit means:

- associating the treatment set to the management unit by inserting the path of the treatment set in the mu_treatment_knowledge_base column of the [MU_header] table of the management unit database, and


Treatments : Table

	treatment_id	source_	treatment_funct	source_	treatment_simu	source_	treatment_icon	source_	treatment_desc	source	treatment_sort_	source_
▶	Clearcut	3	treatment	3	FVS ne	3	clearcut.bmp	3		3	1	3
	Thinning from Above using Basal Area	3	treatment	3	FVS ne	3	thin_above.bmp	3		3	2	3
	Thinning from Below using Basal Area	3	treatment	3	FVS ne	3	thin_below.bmp	3		3	3	3
	Clearcut	3	treatment	3	FVS sn	3	clearcut.bmp	3		3	4	3
	Thinning from Above using Basal Area	3	treatment	3	FVS sn	3	thin_above.bmp	3		3	5	3
	Thinning from Below using Basal Area	3	treatment	3	FVS sn	3	thin_below.bmp	3		3	6	3
	Seedtree	3	treatment	3	FVS ne	3	seedtree.bmp	3		3	7	3
*		0		0		0		0		0	0	1

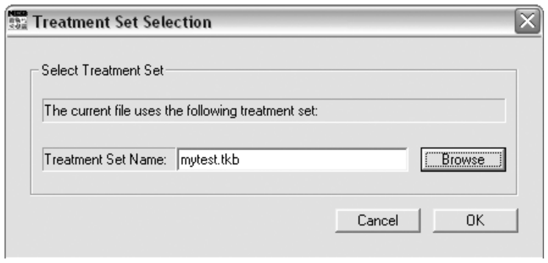
Record: 1 of 7

Figure 2.5: Treatments Table

- creating a link to the treatments defined in the treatment set in the [Treatments] table of the management unit database. A link is created by populating the table (see fig. 2.5). Treatment id, treatment simulator, and treatment icon are acquired from the **treatment/6** predicate. For silvicultural treatments, the treatment function is always equal to 'treatment'. Other treatment function names are used for non silvicultural treatments such as "construct road", and these are ignored by the simulation agent. A counter calculates the sort order for the treatment. To keep track where the particular data was obtained from a source code[2] is entered into the source columns.

Only treatments linked to the [Treatments] table in the management unit database are available to the user to add to a treatment plan. The NED-2 treatment development agent creates treatment sets, but does not populate the management unit database (unless saved to the treatment set already associated with the management unit). The system is designed this way because users might want to create several treatment sets at a time. If saving the treatment set would associate the set with the current management unit in any case, users were required to create and save the treatment set to be associated last. This might not be intuitive to the user.

[2]0=empty, 1=default, 2=default other, 3=default user, 4=calculated, 5=calculated outside NED, 6=imported, 7=user, 8=model, 9=program, 10=absolute



The current file uses the following treatment set:

Treatment Set Name: mytest.tkb

Figure 2.6: Treatment Set Selection

To eliminate this source of error, establishing the association is carried out by the NED-2 treatment set selection agent in a separate process. A simple dialog (see fig. 2.6) allows the user to select a pre-defined treatment set. The association process is initiated by clicking the OK-button. This launches the **associate_file/1** predicate defined in trt_sel.dcm which is responsible for populating the database.

The association between the management unit database and a treatment set can be broken when the *.tkb file is moved to another directory, deleted from the hard drive, or overwritten by a new treatment set. In the case of the latter, this could result in a state where the treatment plans and the treatment set are out of sync. A bulletproofing process, initiated before the simulation attempt, will catch these errors and report them to the user. In this case, this dialog might help to reestablish the association.

2.5 The NED-2 Simulation Agent

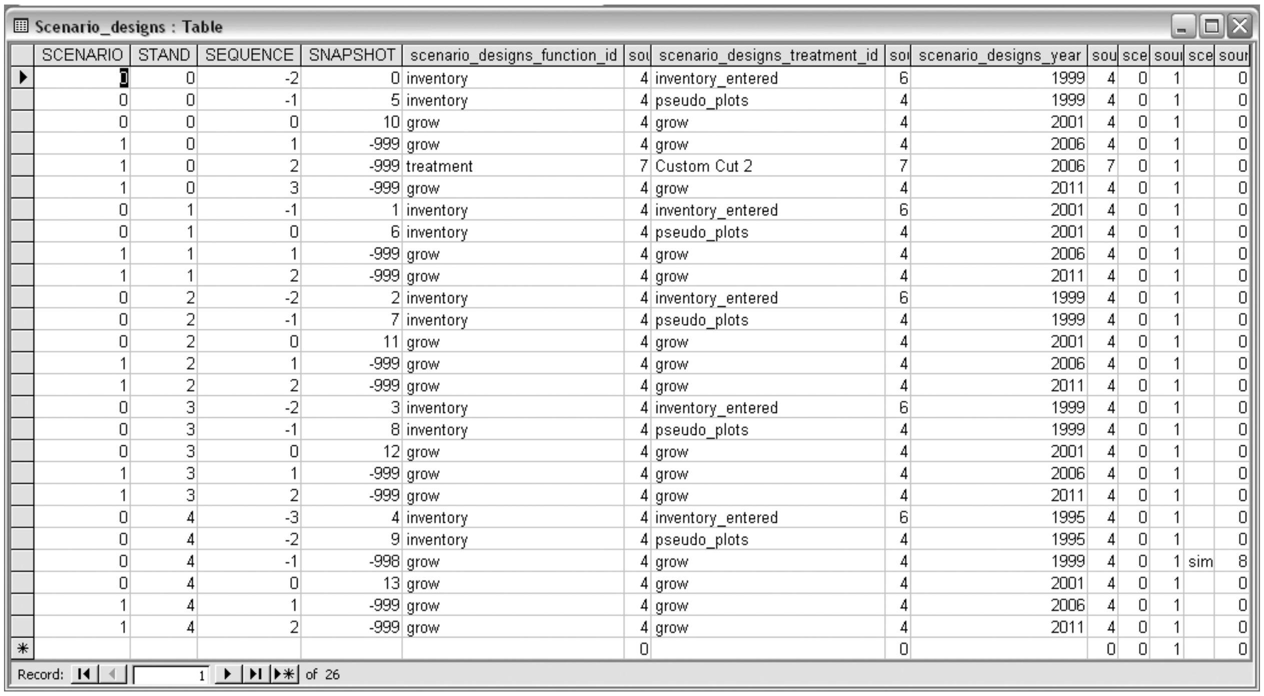
As mentioned above, a design choice was made that requires all stands in a treatment plan to begin in the same year. But the year the last inventory was taken for each stand might differ. Especially with large management units, users like to spread the expenses that are involved in taking inventory over several years. Therefore, projected data have to be simulated for those stands that have no inventory data available for a common baseline year. This process is called *Baseline Generation*.

In NED-2, a management unit is divided into several STANDS. Each stand is in turn subdivided into several CLUSTERS, which again are subdivided into several PLOTS. Each cluster contains exactly one overstory plot, but it may have any number of understory and ground plots. According to the tree's dbh, the tree data are stored in the [Overstory_obs] table or [Understory_obs] table of the management unit database.

Trees in FVS are identified by their stand and plot. To simplify the process of converting the tree data stored in the management unit database to a format readable by FVS, all understory plots in a cluster are *averaged* into one understory pseudo-plot and all ground plots in a cluster are *averaged* into one ground pseudo-plot. The function 'CreatePseudoSnapshot' defined in the dynamic link library *NEDcalc.dll* performs this calculation. This ensures that each cluster contains exactly one overstory plot, at most one understory plot, and at most one ground plot which is necessary to prepare the tree data input for FVS.

The call to *NEDcalc.dll* to execute the 'CreatePseudoSnapshot' function is initiated in the *baseline generation* process only. It is necessary to generate a baseline before simulating treatment plans in any case.

The information for baseline and treatment plans is stored in the [Scenario_designs] table of the management unit database (see fig. 2.7). Plans are identified by their SCENARIO number. The scenario for baseline plan is zero. Treatment plans are assigned increasing integer values. Within a stand the sequence of treatment events are controlled by the SEQUENCE number. Since the sequence number of the baseline year is zero, the sequence numbers are negative for baseline and positive for treatment plans. As mentioned above, a SNAPSHOT represents a stand at a point in time under a particular treatment regime. Therefore, with the aid of the snapshot



Scenario_designs : Table

SCENARIO	STAND	SEQUENCE	SNAPSHOT	scenario_designs_function_id	sou	scenario_designs_treatment_id	so	scenario_designs_year	sou	sce	sou	sce	sou
0	0	-2	0	inventory	4	inventory_entered	6	1999	4	0	1		0
0	0	-1	5	inventory	4	pseudo_plots	4	1999	4	0	1		0
0	0	0	10	grow	4	grow	4	2001	4	0	1		0
1	0	1	-999	grow	4	grow	4	2006	4	0	1		0
1	0	2	-999	treatment	7	Custom Cut 2	7	2006	7	0	1		0
1	0	3	-999	grow	4	grow	4	2011	4	0	1		0
0	1	-1	1	inventory	4	inventory_entered	6	2001	4	0	1		0
0	1	0	6	inventory	4	pseudo_plots	4	2001	4	0	1		0
1	1	1	-999	grow	4	grow	4	2006	4	0	1		0
1	1	2	-999	grow	4	grow	4	2011	4	0	1		0
0	2	-2	2	inventory	4	inventory_entered	6	1999	4	0	1		0
0	2	-1	7	inventory	4	pseudo_plots	4	1999	4	0	1		0
0	2	0	11	grow	4	grow	4	2001	4	0	1		0
1	2	1	-999	grow	4	grow	4	2006	4	0	1		0
1	2	2	-999	grow	4	grow	4	2011	4	0	1		0
0	3	-2	3	inventory	4	inventory_entered	6	1999	4	0	1		0
0	3	-1	8	inventory	4	pseudo_plots	4	1999	4	0	1		0
0	3	0	12	grow	4	grow	4	2001	4	0	1		0
1	3	1	-999	grow	4	grow	4	2006	4	0	1		0
1	3	2	-999	grow	4	grow	4	2011	4	0	1		0
0	4	-3	4	inventory	4	inventory_entered	6	1995	4	0	1		0
0	4	-2	9	inventory	4	pseudo_plots	4	1995	4	0	1		0
0	4	-1	-998	grow	4	grow	4	1999	4	0	1	sim	8
0	4	0	13	grow	4	grow	4	2001	4	0	1		0
1	4	1	-999	grow	4	grow	4	2006	4	0	1		0
1	4	2	-999	grow	4	grow	4	2011	4	0	1		0
					0		0		0	0	1		0

Record: 1 of 26

Figure 2.7: Scenario Designs Table

number pre and post treatment information can be retrieved. Initially, all snapshot numbers are set to -999, which denotes that no calculations have been performed. The NED-2 simulation agent is responsible for assigning unique snapshot values. A value of -998 denotes that no simulation is required.

2.5.1 Baseline Generation

Before the NED-2 simulation agent starts simulating tree data for each stand in the management unit, it first checks if all data necessary for FVS to function are available (see section 2.5.3 for further details). If there is any information missing the agent terminates and opens a *Missing Data Error Report* file in the default Web browser. Otherwise, for every forested[3] stand in the management unit the agent calls *NEDcalc.dll* to create the pseudo-plots for understory and ground plots. The input parameters for the function 'CreatePseudoSnapshot' are the source snapshot, the destination snapshot, and a flag controlling a progress bar. Since the NED-2 simulation agent incorporates its own progress bar, the flag is set to zero to hide the one provided by *NEDcalc.dll*. This is accomplished by the following **winapi/4** command:

```
winapi(('NEDcalc','CreatePseudoSnapshot'),[Source,Destination,0],_)
```

Once the pseudo-plots are created the agent simulates the baseline plan for every forested stand where the inventory year differs from the baseline year. To simulate a baseline plan, the agent needs to consult the management unit database to retrieve information required to set up the FVS controls:

- The baseline year is obtained by parsing the scenario_desc column in the [Scenarios] table where SCENARIO = 0, which is achieved by the **fvs_getBaselineYear/1** predicate.
- To retrieve the forest type, the **is_forested/2** predicate consults the stand_cover_type column in the [Stand_snapshots_measures] table.
- The **fvs_getGModel/4** returns the growth simulator model and its variant stored in the scenario_models_simulator column in the [Scenario_models] table for each stand.
- The treatments applied in the simulation period for a stand are returned in a list of the form `[[TreatmentId,TreatmentYear,TreatmentFunction],...]`

[3]forest type ∈ {Forest, Broadleaf forest, Coniferous forest, Mixed Coniferous/Broadleaf forest, Forested wetland}

by the **fvs_getTreatments/4** predicate. The treatment information is retrieved from the [Scenario_designs] table. For baseline generation, consecutive growth “treatments” are removed from the list. This treatment list is used to determine the thinning keyword to be used and to calculate the cycle lengths later on in the simulation process.

Next, the agent creates the FVS input files (i.e., the tree record file and the keyword file) by executing the predicate **mdb2fvs/6**, which will be explained in detail in section 2.5.4. Afterwards the agent runs the FVS module. The predicate **fvs_run/2** is responsible for the execution.

As mentioned above, the variants currently integrated in NED-2 are the Northeastern variant and the Southern variant. The executable programs for these variants are located in the *prolog/fvs* folder, and are named *NE1.exe* and *SN1.exe*. To launch these programs batch files for both of these variants are created. In the batch files the command line is constructed, the program is executed, and temporary files concerned with the simualation are deleted.

Since FVS is called as an external program, the NED-2 simulation agent does not know when the simulator has finished processing. For this reason, the agent creates a flag file before running the batch file and waits for it to be renamed from *starting.flg* to *finished.flg* by the batch file. This gives the agent adequate control. The content of the batch file *fvs_ne.bat*, which launches the simulator for the Northeastern variant without saving the output files for the user to review is as follows:

```
rem stdFVS run on DOS.
echo data.key >data.rsp
echo data.tre >> data.rsp
echo data.out >> data.rsp
echo data.trl >> data.rsp
echo data.sum >> data.rsp
echo data.chp >> data.rsp
```

```
NE1.exe < data.rsp
del data.rsp
del data.key
del *.fvs
ren starting.flg finished.flg
```

To execute an external Win32 API program, the *kernel.dll* provides the function 'WinExec'. This takes the command line and a flag controlling the 'ShowWindow' function as its parameter. A flag value equal to zero denotes that the window is hidden. The **winapi/4** command is accordingly:

```
winapi((kernel32,'WinExec'),[Command,0],0,_)
```

To retrieve the appropriate command line, **fvs_command/4** provides the necessary metaknowledge. Currently, the FVS output files are saved, depending on the mode the application is run in. If the application is run in stand-alone mode, the batch files *fvs_ne.bat* and *fvs_sn.bat* are called. In development mode the batch files *fvs_ne_copy.bat* and *fvs_sn_copy.bat* are executed in order to save the output to the *prolog/fvs/fvsfiles* folder. Optimally, whether the output files are saved or not should be controlled by the user via a user dialog.

Once FVS has generated the tree list file, including detailed information about all individual tree records, the NED-2 simulation agent can proceed to read the information into the management unit database. This is performed by the **fvs2mdb/14** predicate and will be discussed in more detail in section 2.5.4.

Finally, after all forested stands have been simulated, the NED-2 simulation agent calls the functions 'ShuffleObservations' and 'CalcSnapshot' defined in *NEDcalc.dll*. The function 'ShuffleObservations' is responsible for moving overstory observations whose dbh values are below the overstory/understory threshold to the [Understory_obs] table. It will also move understory observations whose dbh values are

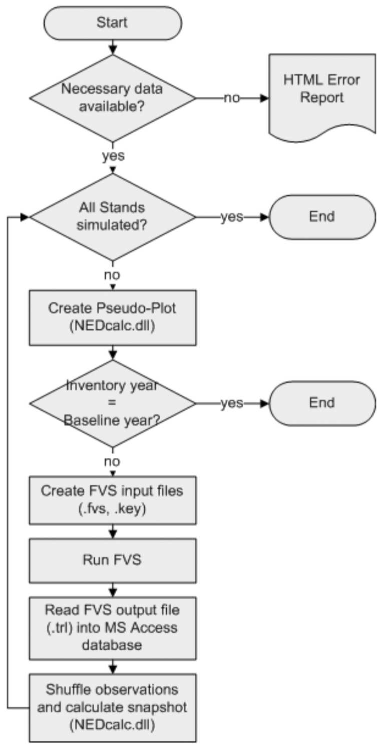

Start
Necessary data available?
no
HTML Error Report
yes
All Stands simulated?
yes
End
no
Create Pseudo-Plot (NEDcalc.dll)
Inventory year = Baseline year?
yes
End
no
Create FVS input files (.fvs, .key)
Run FVS
Read FVS output file (.trl) into MS Access database
Shuffle observations and calculate snapshot (NEDcalc.dll)

Figure 2.8: Program Execution: Baseline Generation

greater than or equal to the overstory/understory threshold to the [Overstory_obs] table. The overstory/understory threshold is defined by the user and stored in the pref_over_under column in the [Preferences] table. The 'CalcSnapshot' function calculates basal area, volumes, and other values. Both functions have as their parameters the snapshot the function should be performed on and the progress-bar controlling flag. Following is a listing of the **winapi/4** commands:

```
winapi((‘NEDcalc’,‘ShuffleObservations’),[Snapshot,0],_)
winapi((‘NEDcalc’,‘CalcSnapshot’),[Snapshot,0],_)
```

The previously described process is depicted in fig. 2.8.

2.5.2 Treatment Plan Simulation

As for *Baseline Generation*, the NED-2 simulation agent starts by checking if all necessary data for FVS are available before simulating any treatment plans. Again, if any information is missing the agent terminates and opens a *Missing Data Error Report* file in the default Web browser. If all data exist, a simple dialog lets the user pick the plans and the stands for simulation. This dialog displays only forested stands, which have not been simulated in a previous simulation process. When the user clicks the OK-button the predicate **scenarios/1** is asserted with a list of plan-stand associations of the form `[[Scenario,[Stand_1,...,Stand_n]],...]` as parameter.

For each scenario in the *plan-stand associations* list, the agent simulates the tree data for each stand associated with this scenario. The simulation process is only initiated if tree records exist either in the [Understory_obs] table or in the [Overstory_obs] table.

Identically to the *Baseline Generation* process the NED-2 simulation agent retrieves the information about the baseline year, the simulator model, and the treatments applied in the simulation period from the management unit database. Furthermore, the process of creating the input files for FVS, running the simulator, reading the output file data to the management unit database, shuffling the observations, and performing the calculations on a snapshot corresponds to the process described in section 2.5.1.

The complete process is illustrated in fig. 2.9

2.5.3 Bulletproofing

For the NED-2 simulation agent to function properly certain conditions have to be satisfied. Therefore, a checkup is performed by the **fvs_bulletproof/2** predicate, before the simulation process is triggered. If any information is missing either a

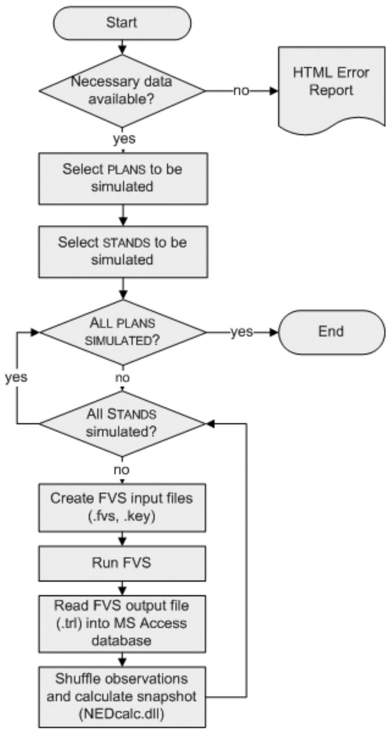

Start
Necessary data available?
no
HTML Error Report
yes
Select PLANS to be simulated
Select STANDS to be simulated
ALL PLANS SIMULATED?
yes
End
yes
no
All STANDS simulated?
no
Create FVS input files (.fvs, .key)
Run FVS
Read FVS output file (.trl) into MS Access database
Shuffle observations and calculate snapshot (NEDcalc.dll)

Figure 2.9: Program Execution: Plan Simulation

messagebox pops up letting the user know what has to be done, or a *Missing Data Error Report* file is generated and opened in the default Web browser.

When generating a baseline it is possible that the user attempts to simulate data that do not exist yet. For this reason, the utility function checks the [Stand_header] table for stand records. This checkup is skipped when simulating treatment plans. Instead, the NED-2 simulation agent checks if treatment plans are developed. If this is the case then there exist records with SCENARIO > 0 in the [Scenarios] table. Also, for a particular scenario to pass the test, there have to be treatments (including growth) defined in the treatment plan. This is determined by the **fvsbp_hasTreatments/1** predicate, which examines the [Scenario_designs] table for records where SEQUENCE > 0 and SNAPSHOT < 0.

To determine if a baseline has been generated, two conditions have to be met. First, there has to be a record in the [Scenarios] table with SCENARIO = 0. If this is not the case, the user has not developed a baseline plan. Second, for each record in the [Scenario_designs] table where SEQUENCE = 0, an integer value greater than zero has to be entered in the snapshot column. If the values in the appropriate snapshot columns equal -999, a baseline has not been simulated.

If the user attempts to generate the baseline, but the baseline already exists, the agent advises the user to advance to develop treatment plans and terminates. If the user tries to simulate treatment plans and the baseline does not exist, the agent warns the user to develop and/or generate the baseline first and terminates. In any other case, the predicate **fvsbp_getMissingData/3** is called to check if the data required by FVS are available.

To run FVS, characteristics of the site where the stand is located have to be specified. The characteristics cover the location (nearest National Forest, and in some cases Ranger District), habitat type or plant community code (ecological unit code for the Southern variant), stand age, aspect, slope, elevation, species code, and site index. The values for these parameters are retrieved from the management unit, are calculated from known variables, or are hard-coded.

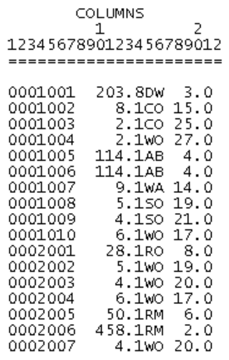
Values retrieved from the management unit database include the aspect, slope, elevation, and site index. These values come from the stand_aspect, stand_slope, stand_elevation, and stand_site_index columns of the [Stand_header] table. The stand age is a calculated variable. It is computed by subtracting the contents of the stand_year_origin column in the [Stand_snapshots_measures] table from the simulation start year. The species code is also calculated. This variable is acquired by converting the value contained in the stand_site_spp column of the [Stand_header] table to the alpha code used by FVS.

To ensure that FVS does not run into any errors, it is checked that the previously mentioned columns are filled with data. For each variable a list is created, containing the stands where data are missing. Location and habitat are hard-coded and therefore do not need checking.

The NED-2 simulation agent determines which *growth* simulator model to use for each stand in a scenario from the scenario_models_simulator column in the [Scenario_models] table where SCENARIO_MODELS_FUNCTION = 'grow'. Hence, the **fvsbp_getMissingGrowthModels/3** predicate has to check if this information is provided. It returns a list of the stands which have no growth model defined.

When silvicultural treatments are defined in the [Scenarios_designs] table, it has to be guaranteed that the necessary treatment information are available to the NED-2 simulation agent. Therefore, in the case that the treatment list returned by the **fvsbp_getTrtId/3** predicate is *not* empty, the **fvsbp_getMissingTrtModels/3** predicate checks if a *treatment* simulation model is defined. Again the scenario_models_simulator column in the [Scenario_models] table is consulted. But this time the content is matched to 'treatment'. If stands exist for which no treatment simulation model is defined, a list of these stands is returned by the predicate. At present, FVS is also used to simulate treatments. This feature has been included in case a later version of NED should allow users to specify different models for growth and for treatment simulations.

As mentioned in section 2.4, the association between a *.tkb file and a management unit can be broken. For this reason, it has to be confirmed that the treatment set defined in the mu_treatment_knowledge_base column in the [MU_header] table exists in the specified location. Furthermore, the **fvsbp_getMissingTrts/3** predicate checks that there exists a definition for each treatment applied in the [Scenario_designs] table to discover any the states where treatment plans are out of sync with the treatment set.



```
         COLUMNS
            1         2
1234567890123456789012
======================

0001001  203.8DW  3.0
0001002    8.1CO 15.0
0001003    2.1CO 25.0
0001004    2.1WO 27.0
0001005  114.1AB  4.0
0001006  114.1AB  4.0
0001007    9.1WA 14.0
0001008    5.1SO 19.0
0001009    4.1SO 21.0
0001010    6.1WO 17.0
0002001   28.1RO  8.0
0002002    5.1WO 19.0
0002003    4.1WO 20.0
0002004    6.1WO 17.0
0002005   50.1RM  6.0
0002006  458.1RM  2.0
0002007    4.1WO 20.0
```

Figure 2.10: The Tree Data File

All *missing data* lists previously created in the subtasks are appended to one list which is used to construct the output in HTML. The **fvsbp_writeErrorReport/3** predicate and its sub-predicates **fvsbp_writeHead/1**, **fvsbp_writeTitle/2**, and **fvsbp_writeData/3** maintain the metaknowledge for the output format.

2.5.4 The Metaknowledge for FVS

Creating the Tree Record File

As stated in [2], the tree record file (*.fvs) inputs the tree information, from which future tree heights and diameters are predicted.

Similar to the keyword file, a certain format has to be provided (see fig. 2.10). The metaknowledge for this format is contained in the **mdb2fvs_fvsFileFormat/6** predicate. For each record in the [Overstory_obs] table of the management unit database the predicate generates a tree data input record to the file. Records from the [Understory_obs] table are only taken into account if the dhb is ≥ 0.05.

Each tree data input record, created by the NED-2 Simulation agent, consists of the plot id (columns 1-4), the tree id (columns 1-7), the tree count (columns 8-13), the tree history (column 14), the species alpha code (columns 15-17), and the dbh (columns 18-21).

The plot id is equivalent to the cluster number. This value comes from the cluster column of the appropriate observation tables. For overstory trees the tree id is retrieved from the over_obs field of the [Overstory_obs] table. To determine the tree id for understory trees the value 200 is added to the value obtained from the under_obs column of the [Understory_obs] table.

The tree history denotes whether the tree is alive or not. The codes 6, 7, 8, and 9 indicate types of tree records that are *not* projected. All other codes assume a live tree that is to be projected. For overstory trees this value is read from the tree_alive column of the [Overstory_obs] table. This variable is set to 1 for understory trees.

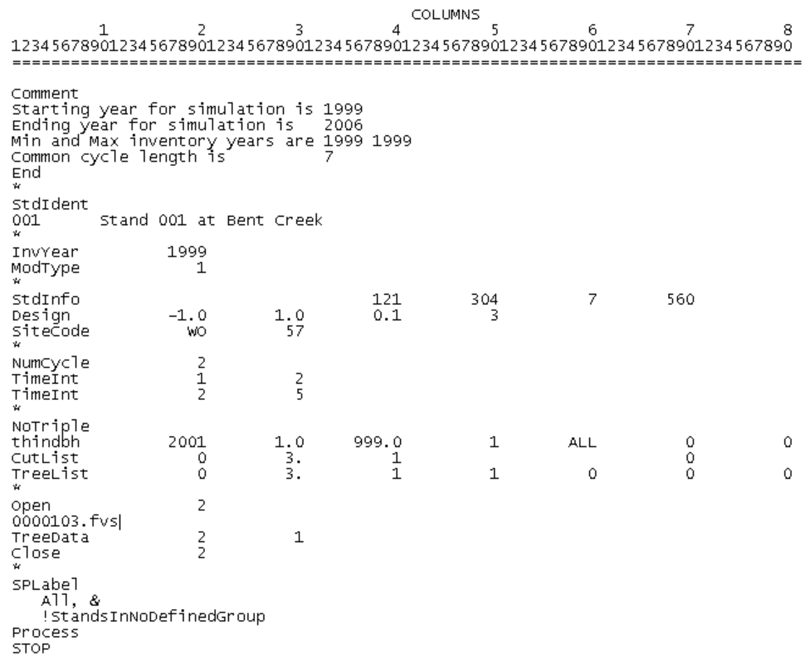
The species code retrieved from the overstory tables have to be converted into the FVS alpha code to be recognized. This is performed by the **ned_alpha_species_code/3** predicate. The tree count is retrieved from the *_stems_per column and the dbh from the *_dbh column of the appropriate observation tables.

To actually construct the tree record in the appropriate format, the predicate **mdb2fvs_writeRecord_xx/8** is consulted.

Creating the Keyword File

In FVS, the simulation of treatment plans is controlled by a logical set of keyword records in the keyword file (*.key). These keyword records have a specific format which needs to be followed precisely for FVS to recognize and run flawlessly. The predicate **mdb2fvs_keyFileFormat/8** maintains the metaknowledge to create such a file (refer to [2] and [3] for more detail on the keyword system). The keyword file controls the simulation of one stand in a scenario.

As shown in fig. 2.11, the keywords specified for a simulation activated by NED-2 are StdIdent, InvYear, ModType, StdInfo, Design, SiteCode, NumCycle



```
                                         COLUMNS
         1         2         3         4         5         6         7         8
12345678901234567890123456789012345678901234567890123456789012345678901234567890
================================================================================

Comment
Starting year for simulation is 1999
Ending year for simulation is   2006
Min and Max inventory years are 1999 1999
Common cycle length is          7
End
*
StdIdent
001      Stand 001 at Bent Creek
*
InvYear         1999
ModType            1
*
StdInfo                              121       304         7       560
Design          -1.0       1.0       0.1         3
SiteCode          WO        57
*
NumCycle           2
TimeInt            1         2
TimeInt            2         5
*
NoTriple
thindbh         2001       1.0     999.0         1       ALL         0         0
CutList            0        3.         1                             0
TreeList           0        3.         1         1         0         0         0
*
Open               2
0000103.fvs|
TreeData           2         1
Close              2
*
SPLabel
   All, &
   !StandsInNoDefinedGroup
Process
STOP
```

Figure 2.11: The Keyword File

and TimeInt, NoTriple, TreeList, CutList, plus the keywords for the silvicultural treatments specified in the scenario of a stand.

Usually, the parameter values for a keyword are retrieved as described in section 2.5.3. The construction of the critical keyword records is described below.

To be able to analyze the tree data generated by FVS in a later step of the *NED-2 decision process*, tree record information has to be output for each year specified in a treatment plan. This is achieved by defining the TreeList keyword. Setting the first parameter of the TreeList keyword to zero tells FVS to output a treelist for each year in the plan. The plan development interface does not restrict the user to a constant cycle length. Therefore, the keyword file has to be set up to account for varying cycle lengths by defining the keywords NumCycle and TimeInt.

The NumCycle keyword denotes the number of projection cycles. This number is determined by the **mdb2fvs_getCycles/2** predicate. For each cycle, the TimeInt keyword indicates how long that cycle is (in years). The **mdb2fvs_writeKeyFile_B/4** predicate calculates this value and creates a keyword record for each cycle. Both predicates use a set of <TreatmentId,TreatmentYear,TreatmentFunction>-triplets returned by the predicate **fvs_getTreatments/4** to perform their calculations.

For baseline generation, it is not necessary to output tree data information for every year specified in a treatment plan. Usually, the baseline tree data are not analyzed. But treatments are performed in the first year of a cycle. To guarantee accuracy it is necessary to start a new cycle in every year in which a treatment is applied. Otherwise, every treatment would be applied in the beginning of the baseline plan.

To have FVS attempt the treatments specified in the scenario for a stand, the appropriate silvicultural treatment keywords (including their parameters) have to be listed. This is acquired by consulting the treatment set (*.tkb) linked to the management unit. The **mdb2fvs_writeKeyFile_C/3** predicate builds the keyword record from the information available in the **treatment/6** predicate defined for the particular treatment.

To retrieve the harvested tree records triggered by a silvicultural treatment keyword, the CutList keyword is specified. This appends the list of harvested tree records to the treelist output file. This information can be used to compute post-treatment tree data.

To read the tree input data records from the tree record file into FVS the keyword Open has to be used in conjunction with the TreeData keyword. The Open keyword tells FVS to open the declared file and associate it with the file reference number specified in its first parameter. The TreeData keyword tells FVS to read the tree records from the file referenced by the reference number.

Interpreting the Tree and Cut List File

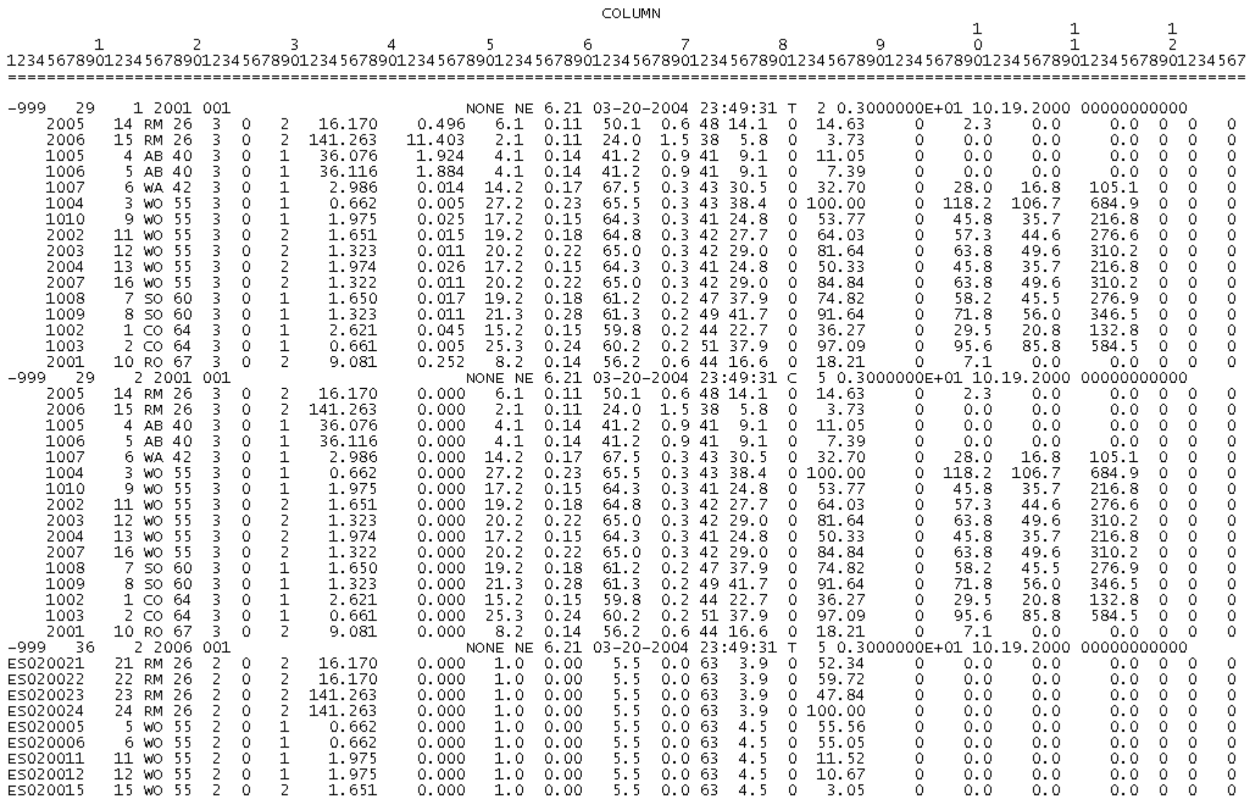
The tree list file generated by FVS has to be read back to the management unit database for NED-2 to handle the output. Hence, the tree records of the tree list file have to be interpreted.

In the tree list file (see fig. 2.12), cycles are separated by a header line identifiable by a preceding -999 (columns 1-4). Harvested tree data for a cycle are output in the same file in a cut list, which has a similar header file. To distinguish between a tree list and a cut list, column 81 in the header line denotes the list type: T for tree list and C for cut list. To identify a cycle, columns 16-19 in the header line define the starting year.

The set of tree data for a cycle follows the header line, one record for each tree. The information relevant to NED-2 are the tree type (columns 1-2), the tree number (column 6-8), the species alpha code (columns 15-16), the plot id (columns 27-29), the trees per acre (tpa) (columns 31-38), and the dbh (columns 49-53).

To translate this knowledge into the NED-2 variables the predicate **fvs2mdb_readFile/19** reads this file line by line. For each line it is checked if in column 1-4 the number -999 is encountered. If this is the case, the agent knows that tree data from a new cycle or from a cut list is about to be read. Therefore, it has to prepare the tables in the management unit database for a new snapshot. Otherwise, the agent consults the metaknowledge for the tree data, which is maintained in the **fvs2mdb_readData/9** predicate.

The tables in the management unit database have dependencies. This means some tables have to be updated first before other tables can be modified. As a rule, first a new record for the new snapshot has to be added to the tables [Stand_snapshots_treatment], [Stand_snapshots_volumes], [Stand_snapshots_measures], and [Stand_snapshots_nontimber]. Then, records can be added to the [Plot_clus-



```
                                                                COLUMN
                                                                                                   1         1         1
         1         2         3         4         5         6         7         8         9         0         1         2
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567
===============================================================================================================================

-999   29    1 2001 001                        NONE NE 6.21 03-20-2004 23:49:31 T  2 0.3000000E+01 10.19.2000 00000000000
    2005   14 RM 26  3  0   2   16.170    0.496   6.1  0.11  50.1  0.6 48 14.1  0  14.63     0    2.3    0.0     0.0  0  0   0
    2006   15 RM 26  3  0   2  141.263   11.403   2.1  0.11  24.0  1.5 38  5.8  0   3.73     0    0.0    0.0     0.0  0  0   0
    1005    4 AB 40  3  0   1   36.076    1.924   4.1  0.14  41.2  0.9 41  9.1  0  11.05     0    0.0    0.0     0.0  0  0   0
    1006    5 AB 40  3  0   1   36.116    1.884   4.1  0.14  41.2  0.9 41  9.1  0   7.39     0    0.0    0.0     0.0  0  0   0
    1007    6 WA 42  3  0   1    2.986    0.014  14.2  0.17  67.5  0.3 43 30.5  0  32.70     0   28.0   16.8   105.1  0  0   0
    1004    3 WO 55  3  0   1    0.662    0.005  27.2  0.23  65.5  0.3 43 38.4  0 100.00     0  118.2  106.7   684.9  0  0   0
    1010    9 WO 55  3  0   1    1.975    0.025  17.2  0.15  64.3  0.3 41 24.8  0  53.77     0   45.8   35.7   216.8  0  0   0
    2002   11 WO 55  3  0   2    1.651    0.015  19.2  0.18  64.8  0.3 42 27.7  0  64.03     0   57.3   44.6   276.6  0  0   0
    2003   12 WO 55  3  0   2    1.323    0.011  20.2  0.22  65.0  0.3 42 29.0  0  81.64     0   63.8   49.6   310.2  0  0   0
    2004   13 WO 55  3  0   2    1.974    0.026  17.2  0.15  64.3  0.3 41 24.8  0  50.33     0   45.8   35.7   216.8  0  0   0
    2007   16 WO 55  3  0   2    1.322    0.011  20.2  0.22  65.0  0.3 42 29.0  0  84.84     0   63.8   49.6   310.2  0  0   0
    1008    7 SO 60  3  0   1    1.650    0.017  19.2  0.18  61.2  0.2 47 37.9  0  74.82     0   58.2   45.5   276.9  0  0   0
    1009    8 SO 60  3  0   1    1.323    0.011  21.3  0.28  61.3  0.2 49 41.7  0  91.64     0   71.8   56.0   346.5  0  0   0
    1002    1 CO 64  3  0   1    2.621    0.045  15.2  0.15  59.8  0.2 44 22.7  0  36.27     0   29.5   20.8   132.8  0  0   0
    1003    2 CO 64  3  0   1    0.661    0.005  25.3  0.24  60.2  0.2 51 37.9  0  97.09     0   95.6   85.8   584.5  0  0   0
    2001   10 RO 67  3  0   2    9.081    0.252   8.2  0.14  56.2  0.6 44 16.6  0  18.21     0    7.1    0.0     0.0  0  0   0
-999   29    2 2001 001                        NONE NE 6.21 03-20-2004 23:49:31 C  5 0.3000000E+01 10.19.2000 00000000000
    2005   14 RM 26  3  0   2   16.170    0.000   6.1  0.11  50.1  0.6 48 14.1  0  14.63     0    2.3    0.0     0.0  0  0   0
    2006   15 RM 26  3  0   2  141.263    0.000   2.1  0.11  24.0  1.5 38  5.8  0   3.73     0    0.0    0.0     0.0  0  0   0
    1005    4 AB 40  3  0   1   36.076    0.000   4.1  0.14  41.2  0.9 41  9.1  0  11.05     0    0.0    0.0     0.0  0  0   0
    1006    5 AB 40  3  0   1   36.116    0.000   4.1  0.14  41.2  0.9 41  9.1  0   7.39     0    0.0    0.0     0.0  0  0   0
    1007    6 WA 42  3  0   1    2.986    0.000  14.2  0.17  67.5  0.3 43 30.5  0  32.70     0   28.0   16.8   105.1  0  0   0
    1004    3 WO 55  3  0   1    0.662    0.000  27.2  0.23  65.5  0.3 43 38.4  0 100.00     0  118.2  106.7   684.9  0  0   0
    1010    9 WO 55  3  0   1    1.975    0.000  17.2  0.15  64.3  0.3 41 24.8  0  53.77     0   45.8   35.7   216.8  0  0   0
    2002   11 WO 55  3  0   2    1.651    0.000  19.2  0.18  64.8  0.3 42 27.7  0  64.03     0   57.3   44.6   276.6  0  0   0
    2003   12 WO 55  3  0   2    1.323    0.000  20.2  0.22  65.0  0.3 42 29.0  0  81.64     0   63.8   49.6   310.2  0  0   0
    2004   13 WO 55  3  0   2    1.974    0.000  17.2  0.15  64.3  0.3 41 24.8  0  50.33     0   45.8   35.7   216.8  0  0   0
    2007   16 WO 55  3  0   2    1.322    0.000  20.2  0.22  65.0  0.3 42 29.0  0  84.84     0   63.8   49.6   310.2  0  0   0
    1008    7 SO 60  3  0   1    1.650    0.000  19.2  0.18  61.2  0.2 47 37.9  0  74.82     0   58.2   45.5   276.9  0  0   0
    1009    8 SO 60  3  0   1    1.323    0.000  21.3  0.28  61.3  0.2 49 41.7  0  91.64     0   71.8   56.0   346.5  0  0   0
    1002    1 CO 64  3  0   1    2.621    0.000  15.2  0.15  59.8  0.2 44 22.7  0  36.27     0   29.5   20.8   132.8  0  0   0
    1003    2 CO 64  3  0   1    0.661    0.000  25.3  0.24  60.2  0.2 51 37.9  0  97.09     0   95.6   85.8   584.5  0  0   0
    2001   10 RO 67  3  0   2    9.081    0.000   8.2  0.14  56.2  0.6 44 16.6  0  18.21     0    7.1    0.0     0.0  0  0   0
-999   36    2 2006 001                        NONE NE 6.21 03-20-2004 23:49:31 T  5 0.3000000E+01 10.19.2000 00000000000
ES020021   21 RM 26  2  0   2   16.170    0.000   1.0  0.00   5.5  0.0 63  3.9  0  52.34     0    0.0    0.0     0.0  0  0   0
ES020022   22 RM 26  2  0   2   16.170    0.000   1.0  0.00   5.5  0.0 63  3.9  0  59.72     0    0.0    0.0     0.0  0  0   0
ES020023   23 RM 26  2  0   2  141.263    0.000   1.0  0.00   5.5  0.0 63  3.9  0  47.84     0    0.0    0.0     0.0  0  0   0
ES020024   24 RM 26  2  0   2  141.263    0.000   1.0  0.00   5.5  0.0 63  3.9  0 100.00     0    0.0    0.0     0.0  0  0   0
ES020005    5 WO 55  2  0   1    0.662    0.000   1.0  0.00   5.5  0.0 63  4.5  0  55.56     0    0.0    0.0     0.0  0  0   0
ES020006    6 WO 55  2  0   1    0.662    0.000   1.0  0.00   5.5  0.0 63  4.5  0  55.05     0    0.0    0.0     0.0  0  0   0
ES020011   11 WO 55  2  0   1    1.975    0.000   1.0  0.00   5.5  0.0 63  4.5  0  11.52     0    0.0    0.0     0.0  0  0   0
ES020012   12 WO 55  2  0   1    1.975    0.000   1.0  0.00   5.5  0.0 63  4.5  0  10.67     0    0.0    0.0     0.0  0  0   0
ES020015   15 WO 55  2  0   2    1.651    0.000   1.0  0.00   5.5  0.0 63  4.5  0   3.05     0    0.0    0.0     0.0  0  0   0
```

Figure 2.12: The Tree List File

ters] table. Next, the tables [Overstory_plots], [Understory_plots], [Ground_plots], and [Transects] can be updated. Finally, records can be added to the tables [Ground_obs], [Transect_obs], [Overstory_obs], and [Understory_obs].

Since the NED-2 project is still expanding, the tables in the database might change to accommodate new variables. To eliminate sources of errors and to stabilize the agent's access to the database, a design decision was made by the NED-2 team to add new columns to a table only at the end of a record. Therefore, to add a record to a table, the length of the record has to be determined first. This is achieved by a conjunction of the predicates **db_get_schema/2** and **length/2**. This knowledge can then be used to read and add records to the table. This is demonstrated in the following code excerpt:

```
db_get_schema(columns('TableName'),Col),
length(Col,ColN),
length([AVar_1,...,AVar_n|LRest],ColN),
db_tuple('TableName',[PVar_1,...,PVar_n|LRest]),
db_add_record('TableName',[Val_1,...,Val_n|LRest]).
```

The tree data records are interpreted by the **fvs2mdb_readData/9** predicate. This predicate extracts the values of interest and translates these into values used by the NED-2 system.

In FVS, trees are identified by their stand and their plot. But in NED-2 trees are distinguished by stand, cluster, and an observation number (OBS). Therefore, the tree number and the plot id retrieved from the tree record have to be converted into the NED-2 format. The cluster of a tree is calculated by subtracting 1 from the plot id. The observation number results from subtracting 1 from the tree number. The tree id is composed of the plot id, a colon, and the tree number (e.g., 1:17).

A tree type equal to ES denotes that this is a new tree created by FVS through regeneration. To ensure that this tree receives a unique tree id, 500 is added to its tree number.

The tpa determines whether the tree is a live tree or not. Therefore, the variable TREEALIVE is set dependent on the tpa as follows:

```
if tpa == 0
then TREEALIVE = 0
else TREEALIVE = -1
```

The species alpha code has to be converted back into the NED-2 species code by the **ned_alpha_species_code/3** predicate.

When the tree data record read from a list of type T, the values for tpa and dbh are as read from the tree data record. Otherwise, the new tpa is calculated by subtracting the tpa read from the tree data record from the old tpa stored in the previous snapshot. The dbh remains the same.

Depending on the value of the observation number (OBS), a record is added to the [Overstory_obs] or to the [Understory_obs] table. If a record with this OBS value already exists in the [Overstory_obs] table or if OBS ≥ 500, then the record is added to the [Overstory_obs]. Otherwise a record with this OBS value must already exist in the [Understory_obs] table, and therefore it is added to the [Understory_obs] table. The original design was to write *all* trees to the [Overstory_obs] table and then the 'ShuffleObservations' function in *NEDcalc.dll* moves them to the appropriate table. But the design for the [Overstory_obs] and the [Understory_obs] table is not the same. For this reason, trees have to be written to the table in which a record with the according OBS value already exists. The function 'ShuffleObservation' will then take care of the conversion.

Chapter 3

Dialog Related Problems and Solutions

3.1 Screen Resolution

When working with dialogs, it has to be considered that users run their computers with different screen resolutions. Especially when users have set up their computers with a low resolution, some of the dialog windows might not fit on the screen.

To prevent this problem, the **rwdcreate/9** and **rwccreate/10** predicates have been defined in *prolog/Utilitis/wnd_scale.dut*. Both predicates calculate the new dimensions of the dialog or control window by considering the screen resolution of the user's computer, the original screen resolution the dialog was built in, and the original width and height of the window to be created. The new dimensions are calculated as follows:

```
NewWidth is int(ScreenWidth * (OrigWidth/OrigScreenWidth))
NewLeft is ScreenWidth/2 - NewWidth/2
NewTop is ScreenHeight/2 - Depth/2
```

Then the window can be created, as usual, with the new dimensions using the predicates **wdcreate/7** and **wccreate/8** provided by Prolog.

To retrieve the current screen resolution the predicate **rcreate_init/0** has to be called before creating the windows. This predicate asserts the screen resolution, which has to be retracted after the windows are created by calling the **rcreate_cleanup/0**.

3.2 Disappearing Dialogs

Another problem related to dialogs is caused by using a combination of C++ and Prolog modules. The problem occurs when a dialog, controlled by the Prolog component, is opened within NED-2. When the user switches to another application and returns to NED-2 again, the dialog did not reappear. But the underlying NED-2 interface (written in Visual C++) is locked since the dialog is still active. The only way to access the Prolog dialog was by using ALT-TAB to go through the list of open tasks.

The main NED-2 interface has no way to determine when Prolog opens a dialog. So it must be told. When Prolog opens a dialog, it passes the handle of the window to C++ by using the 'SetActiveDialog' function in *NEDpnp.dll*. If Prolog passes zero to C++ as the window handle, the C++ module is reset to **no** existing children. Then, when the C++ NED-2 interface has been suspended and receives an "in focus" message, it sends Windows a message telling it to show the current Prolog dialog. The predicate **neddcreate/7**, which is responsible for this action is defined below:

```
neddcreate(Window,Caption,Left,Top,Width,Depth,Style) :-
   wdcreate(Window,Caption,Left,Top,Width,Depth,Style),
   winapi((user32, 'FindWindowA'),[0,Caption],Hndl),
   winapi(('NEDpnp','SetActiveDialog'),[Hndl],Result),
   Result = 1.
```

This solves the problem of coordinating the C++ and the top-level Prolog dialogs. But what happens when the top-level Prolog dialog has opened its own child dialog? In this case, the parent window has to keep in mind which child window it has created. Whenever a window gains focus it has to check if any child windows exist and switches focus to it if it does. This process is repeated until no more children are found.

The window that has received focus, calls the **showChild/1** predicate, which checks if a child window exists. If this is the case, the **wfocus/1** predicate sets the focus to it.

```
showChild(ChildWindow) :-
   catch(ERROR,whandle(ChildWindow,_)),
   (
    ERROR = 0,
    wfocus(ChildWindow)
   ;
    !
   ).
```

The predicate **whandle/2** is responsible for detecting whether a window exists or not. It retrieves the caption of this child window using the **wtext/2** predicate. Using the **winapi/3** predicate, the ‘FindWindowA’ function of the Windows system *user32.dll* is able to obtain the handle for a window with the specified caption.

```
whandle(Window,Hndl) :-
   catch(ERROR,wtext(Window,Caption)),
   ERROR = 0,
   winapi((user32, 'FindWindowA'),[0,Caption],Hndl).
```

Chapter 4

Debugging

Running NED-2 as a stand-alone application sometimes causes errors that have not been detected by the compiler. These errors are extremely hard to trace, since they do not occur when debugging the program.

The only way to catch these errors is by displaying a message box in order to confine the error. The predicate developed for this purposes is the **myspy/1** predicate defined in *prolog/Utilites/nedutil.dut*. It has either a string or a list as input parameter. If the parameter is a string, this string will be displayed in a message box. If the argument is a list, a string will be generated from the elements of the list beforehand. The elements do not necessarily have to be strings, they can be of any type. Most importantly, they can be variables that are instantiated during execution. Thus, a message can be displayed that shows the developer the values of variables at some point in program execution.

One of these errors, still inexplicable, occurs when accessing the management unit database using a call to **db_sql/1**. In some cases, inserting the **myspy/1** predicate or a counter in the form of `forall(integer_bound(1,I,50),(write(I),nl)).` before the offending database access call helps the code to work properly. There seems to be some sort of timing problem that can be eliminated by disconnecting from and reconnecting to the database every time we run into a database read error. But since the NED-2 application is data intensive, this clearly is only a work-around to some deeper problem not yet resolved.

Chapter 5

Conclusion and Future Directions

NED-2 is a robust, intelligent, goal-driven decision support system, that includes a wide variety of software tools available to support decision making in forest ecosystem management. It allows the user to employ the integrated tools without knowledge of the complex interfaces and the data formats inherent to each of these [9].

The preceding chapters have demonstrated what is involved in integrating one of the tools available to the NED-2 system: the Forest Vegetation Simulator. This description can serve as a template for integrating further simulators like the SILVAH model. SILVAH is expected to be the next simulation model to be integrated.

There still exist some unresolved issues concerning the FVS integration. When creating the keyword file (input file for FVS), there are several variables that are hard-coded. To create a more flexible system, efforts should probably be made to calculate those values from known information stored in the management unit database.

At this point, the user picks a simulator. But there does not yet exist a checkup that confirms that the chosen simulation model is able to grow the species included in the management unit. Therefore, a checkup should be added to the bulletproofing process that compares the species defined for a simulation model (a variant of FVS) with the species stored in the stand_site_spp column of the [Stand_header] table of the management unit database.

It is anticipated to extend the metaknowledge of the NED-2 simulation agent to handle additional variants for FVS. This will increase the geographical area the simulation covers. Thereby, the NED-2 application can server a wider range of users.

As mentioned in [10] the regeneration model REGEN described in [1], which is based on [4], is to be integrated into the NED-2 system. Since regeneration is triggered by a treatment the NED-2 simulation agent must interleave the growth model it is using with REGEN. In the case of regeneration being triggered, the agent passes the snapshot number for the tree data identifying the treatment, the REGEN model to be used in the simulation, and a snapshot number associated with the results of the simulated data to the REGEN inference engine. After REGEN completes, the NED-2 simulation agent proceeds to simulate the growth and the silvicultural treatments of the stand. The remaining issue to be solved is how to trigger the regeneration process.

Appendix A

An Index of Predicates

The following is a list of all predicates defined to integrate growth and yield model integration. For each file the predicates are sorted in alphabetical order.

A.1 trt_edit.dcm

dcm/1

dcm(treatments_edit)

+AgentName	<atom>

Comments: This initiates the NED-2 treatment development agent.

fill_default_list/1

fill_default_list(Window)

+Window	<window_handle>

Comments: This fills the *Treatment Templates* listbox with all default treatments defined for the selected simulator.

fill_sim_list/1

fill_sim_list(Window)

+Window	<window_handle>

Comments: This fills the *Simulator* listbox with the names of all defined simulators.

FILL_USER_LIST/1

fill_user_list(Window)

+Window	<window_handle>

Comments: This fills the *Available Treatments* listbox with all user defined treatments for the selected simulator.

GET_SIM/2

get_sim(Window,Simulator)

+Window	<window_handle>
-Simulator	<atom>

Comments: This returns the simulator currently selected.

HNDL_TRTDEF/4

hndl_trtdef(Windows,Message,Data,Return)

+Window	<window_handle>
+Message	<atom>
+Data	<atom>, <integer> or <conjunction>
?Return	<variable> or <atom>

Comments: This provides the processing of window messages for the *Create/Edit Treatment Sets* dialog, specified by the given *Window* handle, *Message* name and *Data* item. In certain cases, it attempts to unify *Result* with an atom that is used to indicate completion of a dialog.

IF_EXIST/2

if_exist(Simulator,TreatmentId)

+Simulator	<atom>
+TreatmentId	<atom>

Comments: This predicate fails when no user defined treatments exist.

TRT_DEF/0

trt_def

Comments: This creates the *Create/Edit Treatment Sets* dialog.

TRTDEF_BROWSE/0

trtdef_browse

Comments: This lets the user browse for a *.tkb file to open.

TRTDEF_CLEANUP/0

trtdef_cleanup

Comments: This retracts all treatment/6 predicates and safely closes the dialog.

TRTDEF_CPY_TREATMENT/0

trtdef_cpy_treatment

Comments: This copies all selected treatments in the *Treatment Templates* listbox to the *Available Treatments* listbox.

TRTDEF_CREATE/0

trtdef_create

Comments: This clears the *Available Treatments* listbox and retracts all user defined treatments to start new.

TRTDEF_DEFINE_PARAMETER/1

trtdef_define_parameter(Window)

+Window	<window_handle>

Comments: This initiates the generation of the *Parameter Settings* dialog.

TRTDEF_DEL_TREATMENT/0

trtdef_del_treatment

Comments: This deletes all treatments selected in the *Available Treatments* listbox.

TRTDEF_DELALL_TREATMENTS/0

trtdef_delall_treatments

Comments: This deletes all treatments displayed in the *Available Treatments* listbox.

TRTDEF_IMPORT/0

trtdef_import

Comments: This opens the *Import* dialog.

TRTDEF_INIT/0

trtdef_init

Comments: This initializes all variables necessary for the *Create/Edit Treatment Sets* dialog.

TRTDEF_SAVE/0

trtdef_save

Comments: This saves all treatments displayed in the *Available Treatments* listbox back to the open *.tkb file.

TRTDEF_SAVEAS/1

trtdef_saveas(Return)

-Return	<atom>

Comments: This save the treatments displayed in the *Available Treatments* listbox to a new file and closes the dialog.

TRTDEF_WRITE_PARAM/2

trtdef_write_param(Window,Treatment)

+Window	<window_handle>
+Treatment	<atom>

Comments: This writes the parameters of the selected treatment to the *Parameter Settings* listbox.

TRUNCATE/2

truncate(String,NewString)

+String	<string>
-NewString	<string>

Comments: This deletes all spaces at the end of a string.

A.2 TRTPARAM_WND.DUT

HEADER/8

header(Simulator,TreatmentId,Icon,Description,Width,Height, WindowType,Key)

+Simulator	<atom>
+TreatmentId	<atom>
+Icon	<atom>
+Description	<atom>
+Width	<integer>
+Height	<integer>
+WindowType	<list>
+Key	<list>

Comments: This creates the header for the *Parameter Settings* dialog.

PARAM_ADD_SPECIES/1

param_add_species(Handle)

+Handle	<integer>

Comments: This adds a species with its associated priority to the *To Retain Species* listbox.

PARAM_ALL_SPECIES/1

param_all_species(Handle)

+Handle	<integer>

Comments: This deselects all species displayed in the *To Retain Species* listbox.

PARAM_CLEANUP/0

param_cleanup

Comments: This retracts all dynamic predicates necessary for handling the *Parameter Settings* dialog and safely closes the dialog.

PARAM_CLOSE_HELP/0

param_close_help

Comments: This closes the *Help* window.

PARAM_CREATE_DEFBUTTON/4

param_create_defbutton(Handle,ParamName,OffsetX,OffsetY)

+Handle	<integer>
+ParamName	<atom>
+OffsetX	<integer>
+OffsetY	<integer>

Comments: This creates the default buttons.

PARAM_CREATE_SUBWND/11

param_create_subwnd(Handle,ParameterName,ValueRange,EditFlag, OffsetX,OffsetY,ParamTemplate,KeywordList,TmpHndlList,HndlList, HndlNew)

+Handle	<number>
+ParameterName	<atom>
+ValueRange	<atom>
+EditFlag	<boolean>
+OffsetX	<integer>
+OffsetY	<integer>
+ParamTemplate	<list>
+KeywordList	<list>
+TmpHndlList	<list>
-HndlList	<list>
-HndlNew	<list>

Comments: This creates the control windows for the parameter base on the *ParamTemplate* template.

PARAM_CREATE_WND/4

param_create_wnd(Handle,WindowTemplate,KeywordList,HandleLast)

+Handle	<integer>
+WindowTemplate	<list>
+KeywordList	<list>
+HandleLast	<integer>

Comments: This creates the windows for the treatment parameters based on the *WindowTemplate* template.

PARAM_CUTCONT_COMBO/1

param_cutcont_combo(Handle)

+Handle	<integer>

Comments: This fills the richtext control window with the pointer to the selected item in the *Cutting Control Flag* combobox.

PARAM_CUTCONTFLAG_CHANGED/1

param_cutcontflag_changed(Handle)

+Handle	<integer>

Comments: This changes the selection in the *Cutting Control Flag* combobox.

PARAM_CUTEFF_RADIOBUTTON/2

param_cuteff_radiobutton(Handle,OldHandel)

+Handle	<integer>
+OldHandle	<integer>

Comments: This handles the changes of the radiobuttons controlling the *Cutting Efficiency* for the treatment *Row Thinning*.

PARAM_CUTEFF0_CHANGED/1 AND PARAM_CUTEFF1_CHANGED/1

param_cuteff0_changed(Handle)

+Handle	<integer>

Comments: This changes the value of the richtext control window to the value associated with the *Cutting Efficiency* scrollbar.

param_cuteff1_changed(Handle)

+Handle	<integer>

Comments: This changes the radiobutton selection depending on the value displayed in the richtext control window controlling the *Cutting Efficiency* for the treatment *Row Thinning*.

PARAM_CUTEFFEDIT/1

param_cuteffedit(Handle)

+Handle	<integer>

Comments: This changes the richtext control window depending on the radiobutton selection controlling the *Cutting Efficiency* for the treatment *Row Thinning*.

PARAM_DEF/6

param_def(Simulator,TreatmentId,Icon,Description,WindowType,Key)

+Simulator	<atom>
+TreatmentId	<atom>
+Icon	<atom>
+Description	<atom>
+WindowType	<list>
+Key	<list>

Comments: This initiates the generation of the *Parameter Settings* dialog.

PARAM_DEL_SPECIES/1

param_del_species(Handle)

+Handle	<integer>

Comments: This deletes all selected species from the *To Retain Species* listbox.

PARAM_EDIT/1

param_edit(Handle)

+Handle	<integer>

Comments: This ensures that only numbers are entered in an edit/richtext control window.

PARAM_EXPECTEDSURV_CHANGED/1

param_expectedsurv_changed(Handle)

+Handle	<integer>

Comments: This changes the value of the richtext control window to the value associated with the *Expected Survival* scrollbar.

PARAM_FILL_COMBOBOX/3

param_fill_combobox(Handle,Key,ParamName)

+Handle	<integer>
+Key	<list>
+ParamName	<atom>

Comments: This fills a combobox, selects the default value, and asserts the hndl_paramdef/4 predicate for this window.

PARAM_FILL_EDIT/4

param_fill_edit(Handle,EditFlag,Key,ParamName)

+Handle	<integer>
+EditFlag	<boolean>
+Key	<list>
+ParamName	<atom>

Comments: This fills an edit control window with its default value and asserts the hndl_paramdef/4 predicate for this window.

PARAM_FILL_LISTBOX/4

param_fill_listbox(Handle,Key,ParamName,Style)

+Handle	<integer>
+Key	<list>
+ParamName	<atom>
+Style	<list>

Comments: This fills a listbox, selects the default value, and asserts the hndl_paramdef/4 predicate for this window .

PARAM_FILL_RICH/4

param_fill_rich(Handle,EditFlag,Key,ParamName)

+Handle	<integer>
+EditFlag	<boolean>
+Key	<list>
+ParamName	<atom>

Comments: This fills the richtext control window with its default value and asserts the hndl_paramdef/4 predicate for this window.

PARAM_FILL_STATIC/3

param_fill_static(Handle,ParamName,Text)

+Handle	<integer>
+ParamName	<atom>
+Text	<string>

Comments: This fills the static control window with its default text.

PARAM_GET_DEFAULT/3

param_get_default(Key,ParamName,DefaultValue)

+Key	<list>
+ParamName	<atom>
-DefaultValue	<atom> or <integer>

Comments: This returns the default value for *ParamName* in the *Key* list.

PARAM_GET_DEFAULT_KEY/2

param_get_default_key(Key,DefaultKeyword)

+Key	<list>
-DefaultKeyword	<atom>

Comments: This returns the default tinning keyword contained in the *Key* list.

PARAM_GET_HNDLLIST/2

param_get_hndllist(Handle,HandleList)

+Handle	<integer>
-HandleList	<list>

Comments: This returns the handle list associated with the current handler (denoting a particular parameter).

PARAM_HELP/3

param_help(ParamName,X,Y)

+ParamName	<atom>
+X	<integer>
+Y	<integer>

Comments: This generates and popups a help context window for *ParamName*.

PARAM_INIT/4

param_init(Simulator,TreatmentId,Icon,Description)

+Simulator	<atom>
+TreatmentId	<atom>
+Icon	<atom>
+Description	<atom>

Comments: This initializes all variables necessary to handle the *Parameter Settings* dialog.

PARAM_METHOD_CHANGED/1

param_method_changed(Handle)

+Handle	<integer>

Comments: This changes the selection of the radiobuttons depending on the value displayed in the richtext control window controlling the *Method of Pruning*.

PARAM_METHOD_RADIOBUTTON/1

param_method_radiobutton(Handle)

+Handle	<integer>

Comments: This changes the value of the richtext control window depending on the selection of the radiobuttons controlling the *Method of Pruning*.

PARAM_OK/1

param_ok(Response)

-Response	<atom>

Comments: This predicate ensures that the treatment name is not a NED-2 protected name, and that there are no empty control windows. It then builds the keyword list and either asserts a new treatment/6 predicate or overwrites the old one. If no errors occur *ok* is returned to close the dialog.

PARAM_PAINT/0

param_paint

Comments: This repaints the graphics control window whenever necessary.

PARAM_PRIORITY_CHANGED/1

param_priority_changed(Handle)

+Handle	<integer>

Comments: This changes the setting of the scrollbar depending on the value entered in the richtext control window controlling the *Priority*.

PARAM_RESET/0

param_reset

Comments: This resets all control windows to their default values.

PARAM_RESIDUALEDIT/1

param_residualedit(Handle)

+Handle	<integer>

Comments: This enables/disables the control windows associated with *Cutting Efficiency* depending on the value in *Residual BA* or *Residual TPA*. If either one of the values is greater zero, the *Cutting Efficiency* windows are disabled.

PARAM_RETAIN_SPECIES/1

param_retain_species(Handle)

+Handle	<integer>

Comments: This deselects all species displayed in the *All Species* listbox.

PARAM_RMTYPE0_RADIOBUTTON/1 AND PARAM_RMTYPE1_RADIOBUTTON/1

param_rmtype0_radiobutton(Handle)

param_rmtype1_radiobutton(Handle)

+Handle	<integer>

Comments: This changes the layout of the dialog depending on the selection of the radiobuttons controlling the *Removal Type* for the treatments *Custom Cut 1* or *Shelterwood Seed Cut*.

PARAM_SBMCUTEFF_CHANGED/2

param_sbmcuteff_changed(Handle,Position)

+Handle	<integer>
+Position	<integer>

Comments: This changes the value of the richtext control window to the value associated with the scrollbar controlling the *Cutting Efficiency*.

PARAM_SBMPRIORITY_CHANGED/2

param_sbmpriority_changed(Handle,Position)

+Handle	<integer>
+Position	<integer>

Comments: This changes the value of the richtext control window to the value associated with the scrollbar controlling the *Priority*.

PARAM_SBMPROPORTION_CHANGED/2

param_sbmproportion_changed(Handle,Position)

+Handle	<integer>
+Position	<integer>

Comments: This changes the value of the richtext control window to the value associated with the scrollbar controlling the *Maximum Proportion.*

PARAM_SET_BUTTON/3

param_set_button(Handle,ParamName,Text)

+Handle	<integer>
+ParamName	<atom>
+Text	<string>

Comments: This asserts the hndl_paramdef/4 predicate for this window.

PARAM_SET_ICON/0

param_set_icon

Comments: This launches the *PickTreatmentIcon.dll* to let the user pick a treatment icon.

PARAM_SET_RADIODEF/5

param_set_radiodef(ParamName,Key, HndlList,Hndl,HndlNew)

+ParamName	<atom>
+Key	<list>
+HndlList	<list>
+Hndl	<integer>
-HndlNew	<integer>

Comments: This sets a radiobutton to its default value, and asserts a hndl_paramdef/4 predicate for this window.

PARAM_SET_SBMCONTROL/4

param_set_sbmcontrol(Handle,Key,ParamName,ValueRange)

+Handle	<integer>
+Key	<list>
+ParamName	<atom>
+ValueRange	<atom>

Comments: This sets a scrollbar to its default value, and asserts a hndl_paramdef/4 predicate for this window.

PARAM_SHADECODE_CHANGED/1

param_shadecode_changed(Handle)

+Handle	<integer>

Comments: This changes the selection of the combobox to the value displayed in the richtext control window controlling the *Shade Code*.

PARAM_SPECIES_CHANGED/1

param_species_changed(Handle)

+Handle	<integer>

Comments: This changes the selection of the combobox to the value displayed in the richtext control window controlling the *Species*.

PARAM_SPECIES_COMBO/1

param_species_combo(Handle)

+Handle	<integer>

Comments: This changes the value of the richtext control window to the *Alpha Code* associated with the selected item in the combobox controlling the *Species*.

A.3 TRTIMPORT_WND.DUT

FILL_FROM_LIST/1

fill_from_list(Window)

+Window	<window_handle>

Comments: This fills the *From File* listbox with all treatments for the selected simulator defined in the loaded *.tkb file.

FILL_IMPORT_LIST/1

fill_import_list(Window) +Window <window_handle>

Comments: This fills the *To Import* listbox with all treatments for a selected simulator chosen for import.

HNDL_IMPORT/4

hndl_import(Windows,Message,Data,Return)

+Window	<window_handle>
+Message	<atom>
+Data	<atom>, <integer> or <conjunction>
?Return	<variable> or <atom>

Comments: This provides the processing of window messages for the *Import* dialog, specified by the given *Window* handle, *Message* name and *Data* item. In certain cases, it attempts to unify *Result* with an atom that is used to indicate completion of a dialog.

IF_EXIST_IMP/2

if_exist_imp(Simulator,TreatmentId)

+Simulator	<atom>
+TreatmentId	<atom>

Comments: This predicate fails when no treatments for the selected *Simulator* exist.

IMPORT/0

import

Comments: This creates the *Import* dialog.

IMPORT_BROWSE/0

import_browse

Comments: This lets the user browse for a *.tkb file to import.

IMPORT_COPY_TREATMENT/0

import_copy_treatment

Comments: This copies all selected treatments in the *From File* listbox to the *To Import* listbox.

IMPORT_CPY_ALL_TREATMENTS/1

import_cpy_all_treatments(Count)

+Count	<integer>

Comments: This copies all treatments displayed in the *From File* listbox to the *To Import* listbox.

IMPORT_DATA/0

import_data

Comments: This copies all chosen treatments to the *Available Treatments* listbox in the *Create/Edit Treatment Sets* dialog. A treatment/6 predicate is asserted for each treatment.

IMPORT_DEL_ALL_TREATMENT/1

import_del_all_treatment(Count)

+Count	<integer>

Comments: This deletes all treatments from the *To Import* listbox.

IMPORT_DEL_TREATMENT/0

import_del_treatment

Comments: This deletes all selected treatments from the *To Import* listbox.

IMPORT_INIT/0

import_init

Comments: This initializes all variables necessary to handle the *Import* dialog.

IMPORT_RESTORE_DATA/0

import_restore_data

Comments: This retracts all temporary created treatment/6 predicates and loads the old treatment/6 predicates previously developed in the *Create/Edit Treatment Sets* dialog.

IMPORT_START/0

import_start

Comments: This initiates the creation of the *Import* dialog.

A.4 TRT_SEL.DCM

ASSOCIATE_FILE/1

associate_file(Name)

+Name	<atom>

Comments: This associates the *.tkb file with the management unit database by inserting the path of the *.tkb file in the mu_treatment_knowledge_base column of the [MU_header] table. Initiates the population of the [Treatments] table.

DCM/1

dcm(treatments_select)

Comments: This initiates the NED-2 treatment set selection agent.

HNDL_TRTSETSEL/4

hndl_trtsetsel(Windows,Message,Data,Return)

+Window	<window_handle>
+Message	<atom>
+Data	<atom>, <integer> or <conjunction>
?Return	<variable> or <atom>

Comments: This provides the processing of window messages for the *Treatment Set Selection* dialog, specified by the given *Window* handle, *Message* name and *Data* item. In certain cases, it attempts to unify *Result* with an atom that is used to indicate completion of a dialog.

POPULATE_TREATMENTS/1

populate_treatments(Counter)

+Counter	<integer>

Comments: This adds a record for every user defined treatment to the [Treatments] table of the management unit database.

TRTSEL_INIT/0

trtsel_init

Comments: This initializes all variables necessary to handle the *Treatment Set Selection* dialog.

TRTSET_SEL/0

trtset_sel

Comments: This creates the *Treatment Set Selection* dialog.

A.5 SIMULATOR.DCM

COPY_DATA/6

copy_data([TreatmentId,TreatmentYear],Scenario,Stand, PreSnapshot,Snapshot,StartPos)

+[TreatmentId,TreatmentYear]	<[atom,integer]>
+Scenario	<integer>
+Stand	<integer>
+PreSnapshot	<integer>
+Snapshot	<integer>
+StartPos	<integer>

Comments: This copies the tree data from a previous snapshot instead of simulating it through FVS.

DCM/1

dcm(Agent)

+Agent	<atom>

Comments: This initiates the NED-2 simulation agent.

FORESTED_STAND/2

forested_stand(Scenario,Stand)

+Scenario	<integer>
+Stand	<integer>

Comments: This checks if a *Stand* in *Scenario* is forested.

FVS_ADD/0

fvs_add

Comments: This adds the selected plans with the selected stands to the *Selected* listbox of the *Plan Selection* dialog.

FVS_BULLETPROOFDLG/0

fvs_bulletproofDlg

Comments: This creates the *Bulletproof* dialog.

FVS_CLEANUP/0

fvs_cleanup

Comments: This deletes all FVS files after the data are copied to the management unit database.

FVS_CLOSENEDCALC/1

fvs_closeNEDcalc(Handle)

+Handle	<integer>

Comments: This closes *NEDcalc.dll*.

FVS_COMMAND/4

fvs_command(Mode,Variant,Stand,Command)

+Mode	<atom>
+Variant	<atom>
+Stand	<integer>
-Command	<string>

Comments: This formulates the command line to run FVS depending on the *Variant* and the *Mode*

FVS_CREATEPSEUDOPLOTS/5

fvs_createPseudoPlots(Stand,Count,Ratio,Snapshot,NextSnapshot)

+Stand	<integer>
+Count	<integer>
+Ratio	<float>

+Snapshot	<integer>
-NextSnapshot	<integer>

Comments: This runs *NEDcalc.dll* for all stands.

FVS_DELETE/2

fvs_delete(Window,IdxList)

+Window	<window_handle>
+IdxList	<list>

Comments: This deletes all selected plans from the *Selected* listbox in the *Plan Selection* dialog.

FVS_DESELECT/2

fvs_deselect(Window,IdxList)

+Window	<window_handle>
+IdxList	<list>

Comments: This deselects all stands in the *Available* listbox in the *Plan Selection* dialog.

FVS_ERRORLISTDLG/1

fvs_errorListDlg(ErrorList)

-ErrorList	<list>

Comments: This creates the *Failed Trees Warning* dialog.

FVS_ESTIMATETIME/2

fvs_estimateTime(ScenarioList,(Hour,Min,Sec))

+ScenarioList	<list>
-(Hour,Min,Sec)	<(integer,integer,integer)>

Comments: This estimates the time the simulation will take.

FVS_FILLSCENARIOS/2

fvs_fillScenarios(Window,ScenarioList)

+Window	<window_handle>
+ScenarioList	<list>

Comments: This fills all scenarios in the listbox of the specified *Window* in the *Plan Selection* dialog.

FVS_FILLSTANDS/2

fvs_fillStands(Window,StandList)

+Window	<window_handle>
+StandList	<list>

Comments: This fills all stands in the listbox of the specified *Window* in the *Plan Selection* dialog.

FVS_FILLSTANDS2/1

fvs_fillStands2(StandList)

+StandList	<list>

Comments: This fills all stands in the *Selected* listbox of the *Plan Selection* dialog.

FVS_FILLTREES/1

fvs_fillTrees(ScenarioList)

+ScenarioList	<list>

Comments: This fills the listbox in the *Failed Trees* dialog with the trees that have been failed to add to either the [Overstory_obs] or the [Understory_obs] table.

FVS_GENERATEBASELINE/8

fvs_generateBaseline(BaselineYear,StandList,NumberOfStands,Count, NumberOfStandsToBeSimulated,Snaphot,StartPos,LastSnapshot)

+BaselineYear	<integer>
+StandList	<list>
+NumberOfStands	<integer>
+Count	<integer>
+NumberOfStandsToBeSimulated	<integer>
+Snapshot	<integer>
+StartPos	<integer>
-LastSnsphot	<integer>

Comments: This generates the baseline for every stand in the management unit. If a stand is not forested or if a stand does not have any treatments, the stand is skipped.

FVS_GETBASELINEYEAR/1

fvs_getBaselineYear(BaselineYear)

-BaselineYear	<integer>

Comments: This returns the baseline year stored in the [Senarios] table.

FVS_GETGMODEL/4

fvs_getGModel(Scenario,Stand,Model,Variant)

+Scenario	<integer>
+Stand	<integer>

-Model	<atom>
-Variant	<atom>

Comments: This returns the simulation model defined for *Stand* and *Scenario*.

fvs_getProgressRatios/6

fvs_getProgressRatios(TrtYear,EndYear,NumStands,StartPosX, StandRatio,DataRatio)

+TrtYear	<integer>
+EndYear	<integer>
+NumStands	<integer>
+StartPos	<integer>
-StandRatio	<integer>
-DataRatio	<integer>

Comments: This returns the ratios for the progress bars.

fvs_getStandList/2

fvs_getStandList(Scenarios,StandList)

+Scenarios	<list>
-StandList	<list>

Comments: This returns a list with all scenario-stands which have not been simulated.

fvs_getTreatments/4

fvs_getTreatments(Scenario,Stand,BasisYear,Treatments)

+Scenario	<integer>
+Stand	<integer>
+BasisYear	<integer>
-Treatments	<list>

Comments: This returns a list of all treatments scheduled for *Scenario* and *Stand*.

fvs_hasTreatments/4

fvs_hasTreatments(Scenario,Stand,StartYear,EndYear)

+Scenario	<integer>
+Stand	<integer>
?StartYear	<integer>
?EndYear	<integer>

Comments: This checks if treatments are defined for *Scenario* and *Stand*.

fvs_hasTrees/1

fvs_hasTrees(Snapshot)

+Snapshot	<integer>

Comments: This checks if there are any trees stored in the tables [Overstory_obs] or [Understory_obs] for a *Snapshot* .

fvs_init/0

fvs_init

Comments: This initialize the simulation process.

fvs_insertScenarios/0

fvs_insertScenarios

Comments: This asserts a list of all scenarios with the selected stands in the *Plan Selection* dialog.

fvs_insertStands/2

fvs_insertStands(Window,Stands)

+Window	<window_handle>
+Stands	<list>

Comments: This adds all *Stands* to *Window*.

FVS_LARGESTNUM/2

fvs_largestNum(List,Largest)

+List	<list>
+Largest	<integer>

Comments: This find the largest number in *List*.

FVS_LOADKBS/0

fvs_loadKBs

Comments: This loads all necessary *.kb files.

FVS_OPENNEDCALC/1

fvs_openNEDcalc(Handle)

-Handle	<integer>

Comments: This initiates *NEDcalc.dll* and returns the *Handle* for this DLL.

FVS_PAINT/1

fvs_paint(Window)

+Window	<window_handle>

Comments: This repaints the graphics *Window* whenever necessary.

FVS_PLANSIMDLG/2

fvs_planSimDlg(Plans,Txt)

+Scenarios	<list>
+Text	<string>

Comments: This creates the *Plan Simulation* dialog.

FVS_RECTANGLE/3

fvs_rectangle(Window,Brush,Width)

+Window	<window_handle>
+Brush	<atom>
+Width	<integer>

Comments: This draws the progress bar.

FVS_RUN/2

fvs_run(Variant,Stand)

+Variant	<atom>
+Stand	<integer>

Comments: This executes FVS.

FVS_RUNNEDCALC/3

fvs_runNEDcalc(Snapshot,LastSnapshot,Scenarios)

+Snapshot	<integer>
+LastSnapshot	<integer>
+Scenarios	<list>

Comments: This runs *NEDcalc.dll* for all snapshots.

FVS_SELECTSCENARIOSDLG/1

fvs_selectScenariosDlg(StandList)

+StandList	<list>

Comments: This creates the *Plan Selection* dialog.

FVS_SETSNAPSHOTS/4

fvs_setSnapshots(Scenarios,Year,Stand,Snapshot)

+Scenarios	<integer>
+Year	<integer>
+Stand	<integer>
+Snapshto	<integer>

Comments: This sets the snapshots in the [Senarios_designs] table for a *Stand* and *Snapshot*.

FVS_SIMULATE/2

fvs_simulate(SimType,LScenario)

+SimType	<atom>
+LScenario	<list>

Comments: This prepares for the simulation process.

FVS_SIMULATEPLANS/5

fvs_simulatePlans(LScenario,Year,LStand,CurSnapshot,LastSnapshot)

+LScenario	<list>
+Year	<integer>
+LStand	<list>
+CurSnapshot	<integer>
-LastSnapshot	<integer>

Comments: This simulates the selected plans. If a stand is not forested or does not have any treatments, this stand is skipped.

FVS_SIMULATINGDLG/2

fvs_simulatingDlg(Scenario,TotPlans)

+Scenario	<integer>
+TotPlans	<list>

Comments: This creates the *Simulating* dialog.

HNDL_FAILEDTREES/4

hndl_failedtrees(Window,Message,Data,Result)

+Window	<window_handle>
+Message	<atom>
+Data	<integer> or <conjunction>
?Result	<atom>

Comments: This provides the processing of window messages for the *Failed Trees* dialog, specified by the given *Window* handle, *Message* name and *Data* item. In certain cases, it attempts to unify *Result* with an atom that is used to indicate completion of a dialog.

HNDL_PLANSIM/4

hndl_plansim(Window,Message,Data,Result)

+Window	<window_handle>
+Message	<atom>
+Data	<integer> or <conjunction>
?Result	<atom>

Comments: This provides the processing of window messages for the *Plan Simulation* dialog, specified by the given *Window* handle, *Message* name and *Data* item. In certain cases, it attempts to unify *Result* with an atom that is used to indicate completion of a dialog.

HNDL_SCENARIOSEL/4

hndl_scenariosel(Window,Message,Data,Result)

+Window	<window_handle>
+Message	<atom>
+Data	<integer> or <conjunction>
?Result	<atom>

Comments: This provides the processing of window messages for the *Plan Selection* dialog, specified by the given *Window* handle, *Message* name and *Data* item. In certain cases, it attempts to unify *Result* with an atom that is used to indicate completion of a dialog.

HNDL_SIMULATING/4

hndl_simulating(Window,Message,Data,Result)

+Window	<window_handle>
+Message	<atom>
+Data	<integer> or <conjunction>
?Result	<atom>

Comments: This provides the processing of window messages for the *Simulating* dialog, specified by the given *Window* handle, *Message* name and *Data* item. In certain cases, it attempts to unify *Result* with an atom that is used to indicate completion of a dialog.

A.6 FVS_BULLETPROOF.DUT

FVS_BULLETPROOF/2

fvs_bulletproof(Scenario,Request)

+Scenario	<integer>
-Request	<atom>

Comments: Before the simulation can run, it has to be ensured that all data necessary for FVS are available. Also it has to be checked if the management unit plans are out of sync with the treatment sets. This initiates the checkup.

fvsbp_getGModel/3

fvsbp_getGModel(Scenario,Stand,Model)

+Scenario	<integer>
+Stand	<atom>
-Model	<atom>

Comments: This returns the growth *Model* defined for the *Stand* in the *Scenario.*

fvsbp_getMissingData/3

fvsbp_getMissingData(Scenario,StandList,MissingData)

+Scenario	<integer>
+StandList	<list>
-MissingData	<list>

Comments: This returns a list of all data missing. This predicate first checks if the year of origin, the site index, the site species, and the growth model are defined for each Stand in the Scenario. If treatments are applied in any year of the Scenario, the predicate further checks if a treatment simulation model and a treatment knowledge base are defined. It also checks if the treatment knowledge base exists in the specified location and if all treatments applied in the Scenario are defined in the treatment knowledge base.

fvsbp_getMissingGrowthModels/3

fvsbp_getMissingGrowthModels(Scenario,StandList,MissingList)

+Scenario	<integer>
+StandList	<list>
-MissingList	<list>

Comments: This returns a list of all stands in *Scenario* that do not have a growth model defined.

fvsbp_getMissingSiSSpp/3

fvsbp_getMissingSiSSpp(Scenario,StandList,MissingList)

+Scenario	<integer>
+StandList	<list>
-MissingList	<list>

Comments: This returns a list of all stands that do not have a site index or a site species defined.

fvsbp_getMissingTrtModels/3

fvsbp_getMissingTrtModels(Scenario,StandList,MissingList)

+Scenario	<integer>
+StandList	<list>
-MissingList	<list>

Comments: This returns a list of all stands that do not have a treatment simulation model defined.

fvsbp_getMissingTrts/4

fvsbp_getMissingTrts(Scenario,TrtIdList,TrtKB,MissingList)

+Scenario	<integer>
+TrtIdList	<list>
+TrtKB	<atom>
-MissingList	<list>

Comments: This returns a list of all treatments displayed in the [Scenario_designs] table that are not defined in the treatment knowledge base.

fvsbp_getMissingYearOfOrigin/3

fvsbp_getMissingYearOfOrigin(Scenario,StandList,MissingList)

+Scenario	<integer>
+StandList	<list>
-MissingList	<list>

Comments: This returns a list of all stands that do not have a year of origin defined.

fvsbp_getNonTrt/2

fvsbp_getNonTrt(Simulator,NoTrtList)

+Simulator	<integer>
-NoTrtList	<list>

Comments: This returns a list of all the events displayed in the [Scenario_designs] table that are not treatments.

fvsbp_getTModel/3

fvsbp_getTModel(Scenario,Stand,Model)

+Scenario	<integer>
+Stand	<atom>
-Model	<atom>

Comments: This returns the treatment simulation *Model* defined for the *Stand* in the *Scenario*.

fvsbp_getTrtId/3

fvsbp_getTrtId(Scenario,StandList,TrtIdList)

+Scenario	<integer>
+StandList	<list>
-TrtIdList	<list>

Comments: This returns a list of treatments displayed in the [Scenario_designs] table.

fvsbp_getTrtKB/1

fvsbp_getTrtKB(TrtKB)

-TrtKBList	<list>

Comments: This returns the treatment knowledge base defined in the [MU_header] table.

FVSBP_HASBASELINE/1

fvsbp_hasBaseline(StandList)

+StandList	<list>

Comments: This predicate checks if the baseline is generated for all stands. It fails, if the baseline does not exist.

FVSBP_HASPLAN/0

fvsbp_hasPlan

Comments: This predicate checks if any plans exist. It fails, if there are no plans developed.

FVSBP_HASSTANDS/1

fvsbp_hasStands(StandList)

+StandList	<list>

Comments: This predicate checks if there exist any stands in the management unit. It fails, if there are no stands in the management unit.

FVSBP_HASTREATMENTS/1

fvsbp_hasTreatments(StandList)

+StandList	<list>

Comments: This predicate checks if the stands in the list have any treatments. It fails, if no treatments are defined in the [Scenario_designs] table for any stand.

FVSBP_LBTEXT/3

fvsbp_lbText(List,TmpString,LbTxt)

+List	<list>
+TmpString	<string>
-LbTxt	<string>

Comments: This returns the text in the format required for the listbox.

FVSBP_PADDING/3

fvsbp_padding(OrigString,PadNum,NewString)

+OrigString	<string>
+PadNum	<integer>
-NewString	<string>

Comments: This fills the text with blanks so that a full line in the listbox is occupied.

FVSBP_REMDOUBLES/2

fvsbp_remDoubles(OrigList,NewList)

+OrigList	<list>
-NewList	<list>

Comments: This ensures that each treatment occurs only ones in the MissingTreatmentList.

FVSBP_WRITEDATA/3

fvsbp_writeData(MissingDataList,TmpString,MissingDataString)

+MissingDataList	<list>
+TmpString	<string>
-MissingDataString	<string>

Comments: This writes the information of the missing data to the error report.

FVSBP_WRITEERRORREPORT/3

fvsbp_writeErrorReport(Scenario,MissingData,File)

+Scenario	<integer>
+MissingData	<list>
-File	<atom>

Comments: This writes the error report to an HTML File.

FVSBP_WRITEHEAD/1

fvsbp_writeHead(Head)

-head	<string>

Comments: This writes the header of the error report.

FVSBP_WRITEHEADERPAGE/1

fvsbp_writeHeaderPage(Scenario,File)

+Scenario	<integer>
-File	<atom>

Comments: This writes the index *File* of the error report for plan simulation.

FVSBP_WRITETITLE/2

fvsbp_writeTitle(Scenario,Title)

+Scenario	<integer>
-Title	<string>

Comments: This writes the *Title* of the error report.

A.7 MDB2FVS.DUT

CONSTRUCT_SUBATOM_AUX/3

construct_subatom_aux(Value,FormatList,Formatted)

+Value	<integer>
+FormatList	<list>
-Formatted	<string>

Comments: This constructs the proper formatting for the variables inserted into the files.

MAKE_ATOM/2

make_atom(Item,Atom)

+Item	<integer>, <string> or <atom>
-Atom	<atom>

Comments: This converts an *Item* of any type into an *Atom*.

MDB2FVS/6

mdb2fvs(Variant,Stand,StartYear,EndYear,PreSnapshot,Treatments)

+Variant	<atom>
+Stand	<integer>
+StandYear	<integer>
+EndYear	<integer>
+PreSnapshot	<integer>
+Treatment	<list>

Comments: This initiates the generation of the input files for FVS (*.key, *.fvs).

MDB2FVS_CHECKNULL/2

mdb2fvs_checkNull(Orig,New)

+Orig	<atom>
-New	<atom>

Comments: This checks if the column in a table of the management unit is empty ($null$).

MDB2FVS_CREATEFVSFILENAME/3

mdb2fvs_createFvsFileName(Stand,Cluster,FileName)

+Stand	<integer>
+Cluster	<integer>
+FileName	<atom>

Comments: This generates the *FileName* for the *.fvs file according to *Stand* and *Cluster*.

mdb2fvs_fvsFileFormat/4

mdb2fvs_fvsFileFormat(Variant,Snapshot,ClusterList,FvsFileTxt)

+Variant	<atom>
+Snapshot	<integer>
+ClusterList	<list>
+FVSFileTxt	<string>

Comments: This creates the content of the *.fvs file according to its format definition.

mdb2fvs_getBafSize/6

mdb2fvs_getBafSize(Variant,OverType,OverBaf,OverSize,Baf,PlotSize)

+Variant	<atom>
+OverType	<integer>
+OverBaf	<integer>
+OverSize	<integer>
-Baf	<integer>
-Size	<integer>

Comments: This returns the values for *Baf* and *PlotSize*.

mdb2fvs_getCycles/2

mdb2fvs_getCycles(Treatments,Cycles)

+Treatments	<atlistom>
-Cycles	<list>

Comments: This returns the number of *Cycles* according to the elements in *Treatments*.

mdb2fvs_getHabitat/3

mdb2fvs_getHabitat(Stand,Variant,Habitat)

+Stand	<integer>
+Variant	<atom>
-Habitat	<atom>

Comments: This returns the value for *Habitat*.

mdb2fvs_keyFileFormat/8

mdb2fvs_keyFileFormat((fvs,Variant),Snapshot,Stand,StartYear,EndYear, Treatments,NCluster,KeyFileTxt)

+Variant	<atom>
+Snapshot	<integer>
+Stand	<integer>
+StartYear	<integer>
+EndYear	<integer>
+Treatments	<list>
+NCluster	<integer>
-KeyFileTxt	<string>

Comments: This initiates the generation of the *.key file according to its format definition.

mdb2fvs_writeKeyFile_A/19

mdb2fvs_writeKeyFile_A(StartYear,EndYear,ComCycleLen,Stand,MuName, LocCode,Habitat,YearsSinceOrig,Aspect,Slope,Elevation,Baf,Size, BreakPointDBH,AlphaCode,Index,NCluster,NumCycle,HeadTxt)

+StartYear	<integer>
+EndYear	<integer>
+ComCycleLen	<integer>
+Stand	<integer>
+MuName	<atom>
+LocCode	<integer>
+Habitat	<atom>
+YearsSinceOrig	<integer>

+Aspect	<integer>
+Slope	<integer>
+Elevation	<integer>
+Baf	<integer>
+Size	<integer>
+BreakPointDBH	<integer>
+AlphaCode	<atom>
+Index	<integer>
+NCluster	<integer>
+NumCycle	<integer>
-HeadTxt	<string>

Comments: This creates the content of the header of the *.key file.

MDB2FVS_WRITEKEYFILE_B/1

mdb2fvs_writeKeyFile_B(TimeIntTxt)

-TimeIntTxt	<string>

Comments: This creates the content of the time interval keyword of the *.key file.

MDB2FVS_WRITEKEYFILE_C/3

mdb2fvs_writeKeyFile_C(Treatments,Variant,TrtTxt)

+Treatments	<list>
+Variant	<atom>
-TrtText	<string>

Comments: This writes the keywords for the *Treatments* in the *.key file.

MDB2FVS_WRITEKEYFILE_D/3

mdb2fvs_writeKeyFile_D(Stand,NCluster,BodyTxt)

+Stand	<integer>
+NCluster	<integer>
-BodyTxt	<string>

Comments: This creates the content of the body of the *.key file.

MDB2FVS_WRITERECORD/6

mdb2fvs_writeRecord(Table,Variant,Snapshot,Cluster,ObsList,Txt)

+Table	<list>
+Variant	<atom>
+Snapshot	<integer>
+Cluster	<list>
+ObsList	<atom>
-Txt	<integer>

Comments: This creates the tree data records of the *.fvs file.

PAD/5

pad(End,Atom,Padding,Length, -NewAtom)

+End	<atom>
+Atom	<atom>
+Padding	<atom>
+Length	<integer>
-NewAtom	<atom>

Comments: This adds the proper padding for the formatting.

A.8 FVS2MDB.DUT

FVS2MDB/14

fvs2mdb(Scenario,Variant,Stand,StartYear,EndYear,PreSnapshot,Snapshot, StandRatio,DataStartPosX,StartPosX,DataRatio,NextSnapshot, NextDataStartPosX,NextStartPosX)

+Scenario	<integer>
+Variant	<atom>
+Stand	<integer>
+StartYear	<integer>
+EndYear	<integer>
+PreSnapshot	<integer>
+Snapshot	<integer>
+StandRatio	<float>

+DataStartPosX	<integer>
+StartPosX	<integer>
+DataRatio	<float>
-NextSnapshot	<integer>
-NextDataStartPosX	<integer>
-NextStartPosX	<integer>

Comments: This initiates the conversion of the FVS output to the database format.

FVS2MDB_ADDRECORD/3

fvs2mdb_addRecord(TableName,PreSnapshot,Snapshot)

+TableName	<atom>
+PreSnapshot	<integer>
+Snapshot	<integer>

Comments: This adds a record to one of the following tables: [Stand_snapshots_treatment], [Stand_snapshots_volumes], [Stand_snapshots_measures], [Stand_snapshots_nontimber], [Plot_clusters], [Overstrory_plots], [Understory_plots], [Ground_plots], [Transects].

FVS2MDB_COPYRECORDS/4

fvs2mdb_copyRecords(TableName,ListType,PreSnapshot,Snapshot)

+TableName	<atom>
+ListType	<atom>
+PreSnapshot	<integer>
+Snapshot	<integer>

Comments: This copies a record to one of the following tables: [Ground_obs], [Transet_obs], [Understory_obs].

FVS2MDB_DATE/1

fvs2mdb_date(Date)

-Date	<string>

Comments: This returns a string of the form: *'Month Day, Year Hour:Minute:Second'*.

fvs2mdb_equal/2

fvs2mdb_equal(FloatNumA,FloatNumB)

+FloatNumA	<float>
+FloatNumB	<float>

Comments: This checks if two floating numbers are equal within 0.000001 accuracy.

fvs2mdb_modifyTables/12

fvs2mdb_modifyTables(ListType,Scenario,Variant,Stand,PreSnapshot, Snapshot,TrtYear,StandRatio,StartPosX,DataRatio,Count,NCount

+ListType	<atom>
+Scenario	<integer>
+Variant	<atom>
+Stand	<integer>
+PreSnapshot	<integer>
+Snapshot	<integer>
+TrtYear	<integer>
+StandRatio	<float>
+StartPosX	<integer>
+DataRatio	<float>
+Count	<integer>
-NCount	<integer>

Comments: This initiates the update of the [Scenario_designs] table, adding records to the tables [Stand_snapshots_treatment], [Stand_snapshots_volumes], [Stand_snapshots_measures], [Stand_snapshots_nontimber], [Plot_clusters], [Overstory_plots], [Understory_plots], [Ground_plots], [Transects]; and coping records to the tables [Ground_obs], [Transet_obs], [Understory_obs].

fvs2mdb_readData/9

fvs2mdb_readData(DataList,Variant,Cluster,Obs,TreeId,Spp,Dbh,TreeAlive, Tpa)

+DataList	<list>
+Variant	<atom>
-Cluster	<integer>

-Obs	<integer>
-TreeId	<integer>
-Spp	<atom>
-Dbh	<integer>
-TreeAlive	<integer>
-Tpa	<float>

Comments: This extracts the relevant data from a record in the *.trl file.

FVS2MDB_READFILE/17

fvs2mdb_readFile(Scenario,Variant,Stand,TrtYear,EndYear,OldPreSnapshot, OldSnapshot,TreeList+ListType,PreSnapshot,Snapshot,StandRatio, StartPosX,DataRatio,Count,NextSnapshot,NextStartPosX

+Scenario	<integer>
+Variant	<atom>
+Stand	<integer>
+TrtYear	<integer>
+EndYear	<integer>
+OldPreSnapshot	<integer>
+OldSnapshot	<integer>
+TreeList	<list>
+ListType	<atom>
+PreSnapshot	<integer>
+Snapshot	<integer>
+StandRatio	<float>
+StartPosX	<integer>
+DataRatio	<float>
+Count	<integer>
-NextSnapshot	<integer>
-NextStartPosX	<integer>

Comments: This reads a line from the *.trl file.

FVS2MDB_UPDATEGRAPHIX/5

fvs2mdb_updateGraphix(StandRatio,DataRatio,StartPosX,Count,NextCount)

+StandRatio	<float>
+DataRatio	<float>

+StartPosX	<integer>
+Count	<integer>
-NextCount	<integer>

Comments: This updates the progress bars.

FVS2MDB_WRITEDATA/12

fvs2mdb_writeData(ListType,PreSnapshot,Snapshot,Cluster,Obs,TreeId, Spp,Dbh,TreeAlive,Tpa,TmpTreeList,TreeList)

+ListType	<atom>
+PreSnapshot	<integer>
+Snapshot	<integer>
+Cluster	<integer>
+Obs	<integer>
+TreeId	<atom>
+Spp	<integer>
+Dbh	<integer>
+TreeAlive	<integer>
+Tpa	<integer>
+TmpTreeList	<list>
-TreeList	<list>

Comments: This writes the tree data to the [Overstory_obs] and [Understory_obs] tables.

UPDATE_SCENARIO_DESIGNS/6

update_scenario_designs(Scenario,Stand, Snapshot,PreSnapshot, TreatmentYear,ListId)

+Scenario	<integer>
+Stand	<integer>
+Snapshot	<integer>
+PreSnapshot	<integer>
+TreatmentYear	<integer>
+ListId	<atom>

Comments: This updates the [Scenario_designs] table by inserting the *Snapshot* number to the appropriate record.

Bibliography

[1] Boucugnani, D.A.; Nute, D., and Loftis, D.L. (2003) *REGEN agent for Excel: a modular and generalized forest regeneration agent.* In Proceedings: Decision Support for Multiple Purpose Forestry, April 23-25, 2003, Vienna, Austria, International Union of Forestry Research Organizations.

[2] Dixon, G.E. (2002) *Essential FVS: A User's Guide to the Forest Vegetation Simulator.* U.S. Department of Agriculture, Forest Service, Fort Collins, CO.

[3] Dyck, M.G. (2002) *Keyword Reference Guide for the Forest Vegetation Simulator.* U.S. Department of Agriculture, Forest Service, Fort Collins, CO.

[4] Loftis, D. L. (1990) *Regenration of southern hardwoods: some ecological concepts.* In: Proceedings of the National Silvicultureal Workshop, July 10-13, 1989, Petersburg, AL. Washington, D.C.: U. S. Department of Agriculture, Forest Service, pp. 139-143.

[5] Maier, M. (2002) *Notes on a Blackboard: Recent work on NED2.* Master Thesis.

[6] Marquis, D.A. and Ernst, R.L. (1992) *User's guide to SILVAH: stand analysis, prescription, and management simulator program for hardwood stands of the Alleghenies.* Gen. Tech. Rep. NE-162. U.S. Department of Agriculture, Forest Service, Northeastern Forest Experiment Station, Radnor, PA. 124 p.

[7] Nute, D.; Kim, G.; Potter, W.D.; Twery, M.J.; Rauscher, H.M.; Thomasma, S.; Bennett, D.J.; and Kollasch, P. (1999) *A Multi-criteria Decision Support*

System for Forest Management. Environmental Decision Support Systems and Artificial Intelligence. AAAI-99, Technical Report WS-99-07 (pp. 68-73). AAAI Press, Menlo Park, California.

[8] Nute, D.; Potter, W.D.; Maier, F.; Wang, J.; Twery, M.; Rauscher, H.M.; Knopp, P.; Thomasma, S.; Dass, M.; and Uchiyama, H.(2002) *Intelligent model management in a forest ecosystem management decision support system.* In A. E. Rizzoli and A. J. Jakeman (eds.), Integrated Assessment and Decision Support: Proceedings of the First Biennial Meeting on the International Environment Modeling and Software Society, Vol. 3: 396-401, International Environmental Modeling and Software Society, Lugano, Switzerland, 2002.

[9] Nute, D.; Potter, W.D.; Maier, F.; Wang, J.; Twery, M.; Rauscher, H.M.; Knopp, P.D.; Thomasma, S.A.; Dass, M.; Uchiyama, H. and Glende, A. (2003) *An Agent Architecture for an Integrated Forest Ecosystem Management Decision Support System.* In Proceedings: Decision Support for Multiple Purpose Forestry, April 23-25, 2003, Vienna, Austria, International Union of Forestry Research Organizations, CD-Rom Proceedings, p. 1-12.

[10] Nute, D.; Potter, W.D.; Cheng, Z.; Dass, M.; Glende, A.; Maier, F.; Routh, C.; Uchiyama, H.; Wang, J.; Witzig, S.; Twery, M.; Knopp, P.D.; Thomasma, S.A.; and Rauscher, H.M. (2004) *Adding New Agents and Models to the NED-2 Forest Management System.* submitted to COMPAG-2004.

[11] Teck, R.M. (1990) *NE-TWIGS 3.0: An individual-tree growth and yield projection system for the northeastern United States.* The Compiler. 8(1):25-27.

[12] Twery, M.J.; Rauscher, H.M.; Bennett, D.J.; Thomasma, S.; Stout, S.; Palmer, J.; Hoffmann, R.; DeCalesta, D.; Gustafson, E.; Cleveland, H.; Grove, J.M.; Nute,

D.; Kim, G.; and Kollasch, R.P. (2000) *NED-1: Integrated Analysis for Forest Stewardship Decisions.* Computers and Electronics in Acriculture, 27, 167-193.

[13] Twery, M.J.; Rauscher, H.M.; Knopp, P.D.; Thomasma, S.A.; Nute, D.; Potter, W.D.; Maier, F.; Wang, J.; Dass, M.; Uchiyama, H.; and Glende, A. (2003) *NED-2: An Integrated Forest Ecosystem Management Decision Support System.* In Proceedings: Decision Support for Multiple Purpose Forestry, April 23-25, 2003, Vienna, Austria, International Union of Forestry Research Organizations, CD-Rom Proceedings, p. 1-17.

[14] Wang J. (2002) *External Heterogeneous Information Source Management Agents.* Master Thesis.