AN ABSTRACT OF THE THESIS OF

Wade Alan Semeliss for the degree of Master of Science in Forest Resources presented June 7, 2002. Title: Stand Level Silvicultural Optimization Utilizing a Late-seral Index Constraint.

Abstract approved: Signature redacted for privacy.

J.Douglas Brodie

A dynamic programming algorithm (MS PATH) was used to develop a stand level optimization model, FPS Optimizer. The model incorporates a late-seral constraint on the profit maximizing objective function utilizing a four-descriptor structural index as a proxy. This index, the 'Old-Growth Index', rates the late-seral structural quality of a given stand from a scale of zero to 100.

Preliminary optimization trials were conducted on a generic planted stand representing industrial management practices, and a case-study stand which originated under natural conditions following harvesting. The FPS Optimizer generated silvicutural prescriptions for both stands, over a one hundred year analysis period, at varying potential final harvest ages. Optimization was conducted under two differing objective functions: 1) maximize the land expectation value of the stand; and 2) maximize the late-seral index value.

Trial results indicate the preference for early, heavy thinnings across diameter classes under the late-seral objective, and moderate understory thinnings for LEV objective, followed by moderate overstory thinnings given longer final harvest ages. The industrial planted stand was able to achieve an index value of 50.0 by age 40 when optimizing on that value, but showed an inability to proceed beyond that level for several hundred years. The natural stand demonstrated a smooth and continuous development response of the old-growth index throughout the analysis period, and the FPS Optimizer model found prescriptions which both increased the LEV and index as compared to a no-harvest activity prescription. Limitations in the silvicultural capabilities of the FPS Growth Model and a simple four-variable old-growth index sometimes produced ambiguous and unrealistic results, thus illustrating limitations with the current form of the FPS Optimizer Model.

Stand Level Silvicultural Optimization Utilizing a Late-seral Index Constraint

by
Wade Alan Semeliss

A THESIS

submitted to

Oregon State University

in partial fulfillment
of the requirements for the
degree of

Master of Science

Presented June 7, 2002
Commencement June 2003

Master of Science thesis of Wade Alan Semeliss presented June 7, 2002.

APPROVED:

Signature redacted for privacy.

Major Professor, representing Forest Resources

Signature redacted for privacy.

Chair of Department of Forest Resources

Signature redacted for privacy.

Dean of Graduate School

I understand that my thesis will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my thesis to any reader upon request.

Signature redacted for privacy.

Wade Alan Semeliss, Author

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to the graduate committee members who have provided me with the knowledge, tools, and guidance that have made my educational experience a success: Dr. Douglas Brodie, Dr. John Garland, Dr. Norman Johnson, and Dr. John Tappeiner. In particular, I would like to thank Dr. Brodie for not giving up on me, despite the continual delays on my part, and even in his retirement finding time to send an occasional reminder inquiring about my progress. In addition, I would like to acknowledge my professional mentors for most of the last decade, Dr. John Beuter and Dr. Robert Latham, who taught me how to apply the knowledge I've acquired. They demonstrated remarkable patience with a young and naïve student many years ago, and their insights have made a profound impression on my attitudes and ambitions. Although I question the sensibility of their past timberland purchases, particularly considering their economics background, I could not imagine a more enjoyable pair of foresters to work for. And last but not least, my family and friends, for always reminding me not to be so serious and providing just the right amount of support and motivation when needed the most.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Stand Level Silvicultural Optimization Utilizing a Late-seral Index Constraint

INTRODUCTION

Overview

My purpose is to demonstrate the application of dynamic programming techniques to silvicultural optimization at the stand level, and evaluate the economic dynamics of imposing a late-seral structure constraint, in the form of an index, on that optimization. Industrial and non-industrial forestland owners face ever-increasing public demands on the management of their private timberlands. Growing public desire for control of current management practices continues to focus on the ecological consequences of short-rotation, even-aged management of forests. Concern about the loss of suitable habitat for wildlife species dependent on late-seral stands, has been of increasing concern. The northern spotted owl (*Strix occidentalis)* and the marbled murrelet (*Branchyrampus mamoratus)* are two examples of recently listed threatened species requiring late-seral forest habitat. Given the current knowledge we have regarding the characteristics of this habitat type, and the current availability of robust stand growth and yield models, it is possible to derive management practices that achieve both financial and ecological objectives. This project focuses on the interaction of two objective functions at the stand level: maximizing financial return to the landowner, and achieving minimum target levels of late-seral structure.

Criticism of conventional industrial management practices points to the lack of late-seral structure present in traditional even-aged managed stands. That is, the absence of large diameter trees, snags, downed logs, species diversity, and multi-canopy structure to name a few important elements. These features have been found to provide critical structure for the survival of late-seral dependent wildlife species. Although these qualities are strongly correlated with the age of a stand, research has shown that active management can direct and accelerate the progression of late-seral structural development. In order to analyze the interaction and dynamics of a financial return objective with that of an ecological objective, we introduce a structure-based grading scheme in the form of an Old Growth Index (O.G.I.), proposed by Acker, et al. (1997.) The index can be used to evaluate any given stand based on four key characteristics, and provides a score from zero to 100 indicating the relative degree of late-seral structure attained (evaluated as the index increases in magnitude). This index is then incorporated into a growth and yield optimization model, FPS Optimizer (Semeliss, 2002), as a constraint on the financial return objective function. The model is run for varying final harvest ages and minimum acceptable levels of the O.G.I. to develop an array of values representing the optimal financial return for a given stand. We will now discuss, in more detail, the basis for structure-based management, the development of the optimization model, the application of the model to generic stands, and a detailed evaluation of its application to a case study forest.

Late-seral wildlife habitat at the stand level

Characteristics and justification

A stand is defined as a relatively homogenous and continuous area within a larger forested area. Although many wildlife species are known to have large home ranges, it is useful to analyze the specific habitat needs at the stand scale prior to introducing the spatial element. I focus on late-seral habitat in this study, due to the current debate surrounding the future of certain animal species dependent on this habitat type. Furthermore, I introduce a dynamic programming model as a tool to evaluate possibilities, and trade-offs, of silvicultural management as a means to developing late-seral stand structure.

> "Within the framework of natural systems lies a rich array of silvicultural possibilities, including density management..." (Perry, David A., et al., 1989.)

Characterizing an old-growth stand has posed some philosophical questions, due to the fact ecosystems are inherently dynamic in nature. Franklin and Spies (1991) proposed emphasizing structural attributes, which can be measured, rather than conceptual qualities, in order to promote a working solution. Marcot, et al. (1991) reviewed inventories of old-growth forests and interim standards for defining such habitat. The standards reviewed focused on key structural components that are generally well-accepted today and easily measured through traditional procedures (eg. large live trees, snag content, downed logs, etc.).

Franklin and Spies (1991) went on to further define these attributes and point out the possibilities for management that promotes this habitat development:

> "...structure is the key to understanding the ecosytem dynamics of old-growth forests. Also, structure is what the forest manager can manipulate to achieve various objectives; this structural manipulation is what silviculture is about."

Significant work has been done over the last decade to identify the key structural components of old-growth forests and quantify the relative values of these variables. Although Franklin and Spies (1991) identified four classes of structure (overstory trees, stand condition, understory, and debris), I focus my work on the overstory component and its variables due to the ability to readily model these attributes.

Management strategies

McComb, et al. introduced a conceptual basis for silvicultural systems that integrates mature-forest wildlife habitat and timber objectives (1993). Specifically, they discussed structural and compositional features of stands associated with mature-forest wildlife and show which silvicultural methods might be used to imitate patterns of natural disturbance. The concept of density management and possible differential treatment of species in multi-canopied stands was tested using simulation with the ORGANON growth model. Their work concluded that silvicultural methods could promote rapid young stand development towards mature-forest structure found in older natural stands. Cole (1995) took a similar

approach in testing the utility of silviculture as a means for promoting mature-forest characteristics and expanded on McComb's work. She utilized Morisita's community similarity index as a proxy for evaluating the simulated stands to natural stands. She concluded that management strategies can yield both timber and wildlife habitat, in differing ratios, according to management objectives. Tappeiner, et al. (1997) proceeded to determine the historical development of old-growth forests and compare this to modern young-growth forests. They propose:

> "Data on density, growth rates, and ages in old stands could be used to develop guidelines for the intensity and timing of density management and regeneration treatments for young-growth stands so their early development resembles that of old-growth stands."

In summary, they concluded that density of old stands was historically low and variable. Furthermore, their study of young unthinned stands showed much higher densities than those that the old-growth stands had at similar ages. Thus, the large acreage of existing young, dense stands are developing on a different trajectory than those stands identified as old-growth likely did. They conclude that, when the management object is to develop old-growth structure, density management will likely be required, possibly several thinnings at early ages.

Hicks, et al. (1999) recommended five strategies for developing spotted owl habitat in managed forests after studying owl telemetry data. Two of their methods point directly to silviculture as a means for producing late-seral structure: 1. Use stocking control practices that accelerate the development of large-diameter stands and understory shrub diversity. 2. Enhance spotted owl hunting habitat by

thinning dense stands to facilitate movement through the site. All of the work points towards the use of silviculture as a means for more quickly achieving late-seral structure objectives, and yields insight into the applicable stand dynamics.

The Old-Growth Index

Building on the interest in the management of and for old-growth structure, Acker, et al. (1997) produced a study investigating the utility of an index-based approach for evaluating the overall late-seral structure of a stand. They attempted to address the following issue:

> "Among the fundamental questions concerning the ecology of forests are how long old-growth structure takes to develop and whether some aspects of this structure develop more quickly than others. By understanding the processes underlying old-growth development in natural forests, it may be possible to devise practices that will hasten such development in managed forests."

They explored the trends in development of old-growth characteristics and timber volume growth-rates using permanent plot data from 20 stands in Oregon and Washington. These stands had been measured periodically from four to eight decades and represented a spectrum of maturity.

In order to document the development of old-growth characteristics in these stands, they incorporated a multivariate index, I_{og}. This old-growth index focuses on four key structural variables that successfully discriminate among age classes of forest (Spies and Franklin, 1991):

1. standard deviation (SD) of tree DBH;
2. density of large(>100 cm DBH) Douglas-fir trees (trees ha^{-1});
3. mean tree DBH (cm); and
4. density of all trees > 5 cm DBH (trees ha^{-1}).

Spies and Franklin (1991) had originally identified 22 descriptors of live forest structure, however, these four variables were isolated for the author's preliminary work.

The index is noted as a special case of the dissimilarity measure known as the Gower metric (Greig-Smith, 1983). That is, I_{og} measures the dissimilarity of young stands from old stands based on the composite value of the variables in the function. The young and old forests are opposite endpoints on a one-dimensional continuum. The index is formulated as shown:

$$I_{og} = 25\sum_i \left| \frac{x_{i,\,obs} - x_{i,\,young}}{x_{i,\,old} - x_{i,\,young}} \right|$$

where i = 1 to 4 (each of the structural variables) and $x_{i,obs}$ is the observed value for that variable. $x_{i,\,young}$ and $x_{i,old}$ are the mean values of the young and old forests of the ith variable. As expected, the equation applied to any given stand will yield a value between 0 and 100, with the largest number suggesting a stand has similar characteristics to that of the old forests used to calibrate the equation, and a zero score representing the equivalent of a young forest. Any number in between is interpreted as a stand's relative position between the young and old forest condition gradient.

I_{og} is appropriately constrained to only return a value between zero and 100. If the value of a particular structural variable is more extreme than either the mean of the 'young' or 'old' values, then the observed value is constrained by setting it equal to that value. The constraints on the observed values are described as follows:

$$X_{i,\,obs} = \begin{cases} X_{i,\,young}, & \text{if } X_{i,\,obs} < X_{i,\,young} \text{ for } i = 1 \text{ to } 3 \text{ and} \\ & \text{if } X_{i,\,obs} > X_{i,\,young} \text{ for } i = 4; \\ X_{i,\,old}, & \text{if } X_{i,\,obs} > X_{i,\,old} \text{ for } i = 1 \text{ to } 3 \text{ and} \\ & \text{if } X_{i,\,obs} < X_{i,\,old} \text{ for } i = 4; \end{cases}$$

The authors applied the index to their case study stands and mapped the progression and dynamics of the index over the time period in which data was observed. They concluded that the overall trend among all stands was a rapid increase in I_{og} up to about 60 to 80 years of age, with a more gradual increase beyond that. At measurements closest to 100 years of age, the I_{og} ranged from 43 to 67. That is, stands generally achieved about half the transition to old-growth by age 100. Their work suggested that limiting tree density in the first several decades of growth may hasten development of old-growth characteristics. Furthermore, activities that promote the growth of large individual trees and accentuate heterogeneity of tree sizes were likely important steps for speeding development of old-growth structure.

Stand level silvicultural optimization

Much work has been done previously regarding goal-based programming of stand development. The earliest methods involved the use of simulation as a means for identifying management scenarios for achieving a given objective. Simulation, however, does not ensure that one has achieved the 'best' solution. Given a complex objective function with competing variables, one is not even assured of obtaining a 'good' solution. Dynamic Programming (DP) is the most prevalent technique in the literature for determining optimal management regimes given an objective function.

Brodie, Adams, and Kao (1978) applied a methodology for determining optimal thinning intensity and rotation age interval for an even-aged stand of Douglas-fir based on maximization of present net worth as the objective function. They introduced the dynamic programming technique as a forward recursive search algorithm. Later (1979), they improved the approach using three-descriptors which accounted for accelerated diameter growth as provided by the DFIT model (Bruce, et al. 1977).

Due to early computational limitations, subsequent effort was focused on neighborhood approaches to solving the 'curse of dimensionality'. That is, as more variables are included in the model, the number of nodes in the network increases exponentially. Although increases in the number of state variables leads to a more robust production surface, the computational efficiency degrades rapidly. The

neighborhood approach reduced the number of nodes in the network by grouping similar solutions into 'neighborhoods'. This approach proceeded into the mid-1980's and culminated with the four-descriptor LPOPT (Lodgepole Pine Optimization) model (Haight, Brodie, and Dahms, 1985).

In 1987, Paredes and Brodie broke new ground and introduced the PATH algorithm to address the computational inefficiencies of previous work. The PATH algorithm was derived through the use of LaGrange multipliers and is applied silviculturally by prescribing treatments in a stand by considering both the direct return (or cost) from a particular decision and the return from the future productivity of the residual stand. Thus, the algorithm analyzes all possible decisions completely at a single stage and selects the single path leading to the next stage of analysis. This approach is based on the assumption of a monotonically increasing production function, which is usually satisfied considering the concave yield surface of stand volume growth over time. Yoshimoto, Haight, and Brodie (1990) improved upon the PATH algorithm by introducing MSPATH. As previously mentioned, the calculus underlying the PATH algorithm relies on the assumption of a concave production surface. When a model is expanded to include variables that deform the surface and may result in a quasi-concave shape, the single stage look-ahead approach of the PATH algorithm may miss the optimal network route. For example, the inclusion of variable log prices for different diameters introduces a 'stair-step' shape to the objective value function which may extend beyond a single stage analyzed. The MSPATH algorithm solves this

dilemma by evaluating each decision based on current returns (and costs) plus the value of the residual stand *several* stages ahead.

Most recently, Cousar (1992) expanded on the MSPATH algorithm by incorporating a Region Limiting Search (RLS) feature. The RLS method allows problems to be handled with numerous state variables. RLS is applied by introducing a second phase in addition to the traditional DP search - a strategy phase. Initially, the algorithm determines which combination of controls are optimal, and then an adjustment phase is entered which iterates the thinning levels. The algorithm alternates between the two phases as better solutions are found until convergence to a local optimum is found. It is important to note that the RLS method does not guarantee a global optimum, thus only providing a 'good' solution.

FPS OPTIMIZER MODEL

The FPS Optimizer model is a desktop software application that integrates the MSPATH algorithm, the FPS growth and yield software package, and a late-seral index mechanism to generate optimal silvicultural prescriptions. I will now describe the critical elements of the MS PATH algorithm (with modifications), the FPS (Arney, 1995) growth and yield package, and the structure of the FPS Optimizer application.

The MS PATH Algorithm

The PATH algorithm (Brodie and Paredes, 1987, Yoshimoto et al. 1990) is a forward recursive dynamic programming construct based on the following concept:

> "maximize (or minimize) the sum of the return from the activity at the current stage and the return from the residual stand at a future stage. Thus, at every stage, where the activity (thinning) is implemented, the optimal strategy (thinning level) is determined at the same time." (Paredes and Brodie).

The mathematical formulation is expressed in terms of marginal return (M) in order to solve the objective function J^*:

$$J^* = \text{maximize} \sum_{n-1}^{N-1} \int_{n}^{n+1} M(t)dt$$

where t_n is time at stage n, N is the last stage of analysis, and the integral is the total return within the interval (t_n, t_{n+1}). If we specify that $R(X_n)$ as the return from the

residual stand (at stage n) X_n after the thinning, $R(Y_n)$ as the return from the residual stand Y_n before the thinning, and $R(T_n)$ as the return from the activity T_n, we can link all the states using a motion equation, which implies:

$$R(Y_n) - R(T_n) = R(X_n)$$

That is, the return from the residual stand after the activity, minus the return from the activity, equals the return from the residual stand after thinning. If we further specify the relationship as a continuous function:

$$J^* = \text{maximize} \sum_{n-1}^{N-1} J_n = \sum_{n-1}^{N-1} R(X_{n+1}) + R(T_n) - R(Y_n)$$

We now have the objective function in terms of a maximization problem based on the summation of the return from the future activity plus the marginal return in the current period (yielding J_n, the return at stage n), summed over all stages (N - 1). In terms of the PATH algorithm, this is applied by maximizing J_n for all stages (n = 1,2,,N).

Cousar (1992) describes the global objective mathematically as:

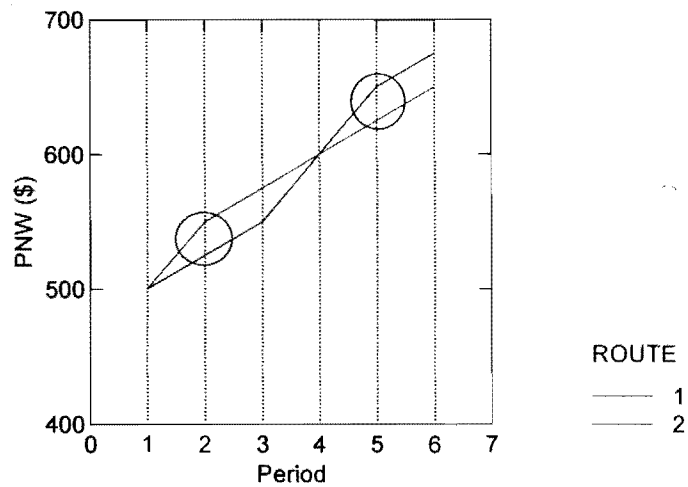
$$J^* = \text{maximize} \sum_{n-1}^{N-1} J_n = (Y_{n+1}) + (T_n)$$

for the domain ($0 < T_n < Y_n$). In other words, the optimal path is determined in a step-wise fashion, starting at (n = 1), and solving for the objective function J_n at each period. Thus, the computational storage is limited to all the numerical iterations associated with thinning possibilities for a single period. The 'winning' activity is selected based on the maximization of the marginal return from the

activity (at n), evaluated at (n + 1), simultaneously considering the value of the residual stand (at n + 1). The algorithm then increments forward one period and repeats the single-stage maximization function, etc.

MS PATH introduces the concept of a multi-period look ahead for evaluating the objective function, J_n. Given the underlying assumption of a concave monotonically increasing production function that supports the optimality assumption of the PATH algorithm, what happens when the production surface assumption is violated? Figure 2.1 demonstrates a scenario where this can occur and the repercussions of violating the optimality assumption: when the projected revenues and costs are based on Scribner board foot volume and a price differential is given for logs of increasing diameter.

FIGURE 2.1. Two Route Network Illustrating Impact of Quasi-concave Production Function[1]



Observe how the PATH algorithm with the single-stage look-ahead period would select the incorrect node at the evaluation period and proceed forward on a sub-optimal route of the network (i.e. the price premium of large diameter sawlogs is not realized for several periods after the low density thinning occurs.) The MS PATH algorithm essentially solves this problem by allowing for quasi-concave production surfaces - a multiple stage look-ahead approach would observe the downstream benefit of the activity at period 1. This ability is of particular importance when we introduce the old-growth index (I_{og}) as a constraint on the objective function, as some index variables have a significant time frame for response to thinning activities under certain conditions.

---

[1] Assume two decisions available at period 1: A no-thinning activity (Route 1) and a heavy understory thinning (Route 2). The effect of diameter growth acceleration is not realized in terms of value-added effect to PNW from the larger sawlog diameter price premium until period 5.

The FPS Optimizer model modifies the PATH algorithm, using the underlying principle of multi-stage look-ahead from the MS PATH algorithm. By evaluating the decision function for all periods (J_n) with a constant stage for evaluating the residual stand: $X_n = X_N$, the look-ahead stage is always the final harvest stage. This modification increases the computational requirements of the model, as well as prohibits the direct solution of the optimal rotation age question, but avoids the peril of entrapment in local optima.

Dual-optimization, the search for an optimal activity route with two discrete objective functions, is not directly feasible. We integrate the second objective function, I_{og}, as a constraint on the economic objective (PNW). The FPS Optimizer model seeks to maximize PNW for a stated final harvest period (N), based on achieving a minimum Iog (at stage N). Thus, the FPS Optimizer objective function is illustrated by:

$$J^* = \text{maximize} \sum_{n-1}^{N-1} J_n = \sum_{n-1}^{N-1} R(X_N) + R(T_n) - R(Y_n)$$

$$s.t. \Rightarrow I_{og,N} \geq I_{og,min.,N}$$

The Forest Projection System (FPS)

The Forest Projection & Planning System (FPS) was created by Dr. James Arney of Forest Biometrics, Inc. The software utilities represent modern forest inventory database design, area and volume computations, growth projection, and harvest planning. The utilities are designed to read and write to any Microsoft ODBC compliant database system, with GIS functionality included. The core

utilities are the FPS compiler (for compiling field plot data and computing stand statistics), the FPS growth model (for projecting compiled stands growth under a variety of conditions and future activities with a library of species and regional calibrations), and the FPS harvest scheduler (a binary search harvest scheduling model).

The FPS Optimizer model utilizes the FPS compiler and growth model to calculate stand statistics from plot data and projected stand growth. The FPS compiler can compile stands that were inventoried using fixed plot or variable radius sampling techniques, or a combination of both sampling systems. The FPS growth model is a single-tree, distance-dependent growth model capable of modeling mixed-species and mixed-age stands. In addition, the user can specify parameters regarding site-preparation, brush control, animal control, planting, fertilization (nitrogen only), and a mix of alternative intermediate harvesting activities. Intermediate harvesting activities can be specified for any report year and can be identified in terms of basal area or stems per acre (hectare) removed. Harvesting can be accomplished from 'below', 'above', or at a fixed 'd/D ratio' (diameter of trees cut to diameter of residual stand). Multiple harvest entries are allowed.

Initial plot data is read from the ODBC database in the form of plot data. Output is written back to the database in the form of a treelist containing detailed records for each species and diameter class. Harvested trees and mortality are

represented separately by a group flag. The output table specifies cubic and board foot volumes, total heights, crown ratios, ages, and defect.

FPS Optimizer Model Structure

I have named the central software developed in this project The FPS Optimizer. It was programmed using the JAVA (JDK 1.3.1) language (SUN Microsystems, Inc.). A third-party database server, JDataConnect (NetDirect, Inc.), is used to provide a linkage between the ODBC database and the program. The model requires an ASCII formatted text file 'Opti_in.txt' to specify the input parameters required by the model, an FPS compatible ODBC database, and the FPS software package to run. Output is sent to an ASCII text file 'Opti_out.txt' after a single optimization run is completed. Table 2.1 lists the input variables required by the model. Figure 2.2 depicts the relationship of the three entities mentioned.

Figure 2.2. FPS Optimizer Model Process Schematic

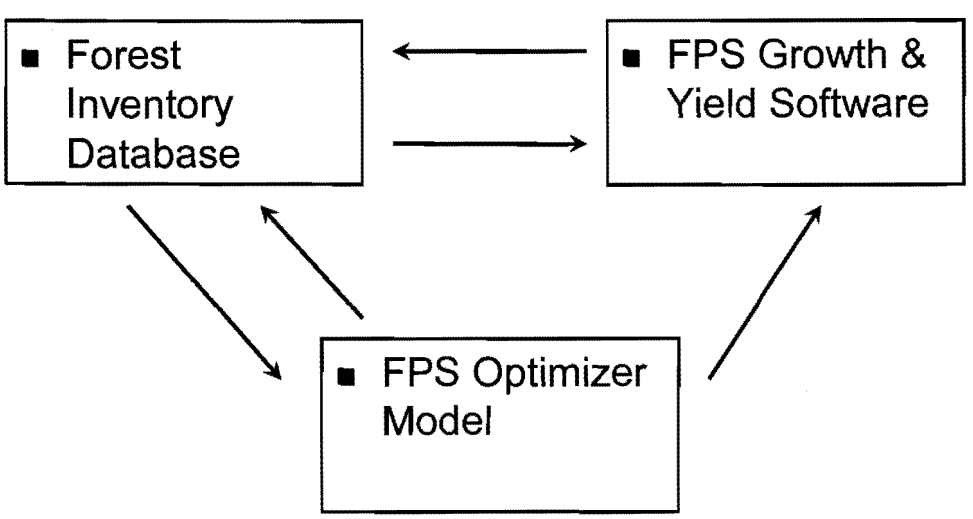


Table 2.1. FPS Optimizer Model Input Variables

Variable	Description	Type
Units	Unit type of I_{og} parameters:1 = metric, 2 = english	Integer
Rate	Real discount rate	Float
Base	Base year for discounting (t_0)	Integer
Yr_All	Years for activity attempts and reporting (up to 20 periods allowed)	Integer
Price_Spc	Tree species for log price array (up to 10 species)	String
Price_Sorts1	Log sort # 1 delivered log price ($/Mbf) array	Integer
Price_Sorts2	Log sort # 2 delivered log price ($/Mbf) array	Integer
Price_Sorts3	Log sort # 3 delivered log price ($/Mbf) array	Integer
Min_Ogi	Minimum Old-Growth Index value constraint for the objective function (0 to 100).	Integer
Annual_Cost	Period recurring annual fixed cost ($/acre)	Integer
Haul_Cost	Delivery cost truck to mill ($/Mbf)	Integer
Dep_Cost	Logging contract price matrix (5x5) used to derive harvest cost equation ($/Mbf - fell, buck, yard, and load)	Integer
Pct_Dep_Cost	Non-commercial thinning contract price matrix (1x5) used to derive non-commercial thinning cost equation ($/acre - cut and slash)	Integer
Bdft_Min	Minimum board foot volume per acre to define break between commercial and non-commercial treatment.	Integer
Sd_Dbh_Young	Mean standard deviation of diameter sampled 'young' stands for I_{og} equation.	Float
Sd_Dbh_Old	Mean standard deviation of diameter sampled 'old' stands for I_{og} equation.	Float
Tpa_Lg_Young	Mean stems per acre (hectare) of Douglas-fir trees > 100cm in sampled 'young' stands for I_{og} equation.	Float
Tpa_Lg_Old	Mean stems per acre (hectare) of Douglas-fir trees > 100cm in sampled 'old' stands for I_{og} equation.	Float
Dbh_Young	Mean average diameter of all trees > 5cm of sampled 'young' stands for I_{og} equation.	Float
Dbh_Old	Mean average diameter of all trees > 5cm of sampled 'old' stands for I_{og} equation.	Float
Tpa_All_Young	Mean stems per acre (hectare) of all trees > 5cm in sampled 'young' stands for I_{og} equation.	Float
Tpa_All_Old	Mean stems per acre (hectare) of all trees > 100cm in sampled 'old' stands for I_{og} equation.	Float
Cut_Interval	Stems per acre cutting interval for model to decrement on.	Integer
Cut_Min	Minimum stems per acre for model to consider activities before shutting off.	Integer
Final_Year	Year assumed for final harvest and to evaluate objective function	Integer
Db_Name	Name and extension of selected database for analysis	String
Econ_Method	Specifies economic valuation method. 1 = Faustmann (LEV) 2 = Modified LEV	Integer
Pd_Zero_Cost	1st year fixed cost (i.e. planting, $/acre)	Integer
Resid_Value	Real value of all subsequent rotations for Modified LEV method ($/Acre)	Integer
Ogi_Inflection	Stems per acre value where 'Tpa_all' inflects in Iog equation	Integer

The user specifies values in the model for all input variables via a graphic user interface (GUI). The critical variables directing the search algorithm are 'Final_Year' and 'Ogi_Min'. These variables represent the anticipated final harvest year (and the year for the MS PATH algorithm to evaluate the objective function), and the minimum acceptable I_{og} value. After data-entry is complete and a successful connection to an ODBC compliant database is made, processing is initiated. After all iterations of the forward recursion are complete, for all years analyzed, the model exits and produces a text file displaying summary statistical data for the stand, at each report year, along with the I_{og} value and calculated I_{og} variables. In addition, all silvicultural activities that were found to increase the value of the objective function (subject to the late-seral constraint) are identified. Figure 2.3 illustrates the framework of the model process. Figures 2.4, 2.5, and 2.6 display computer screenshots of the software application graphic interface. Figure 2.7 is a sample of a partial text file output produced by the program following a completed optimization trial.

Figure 2.3. FPS Optimizer Model Process Schematic

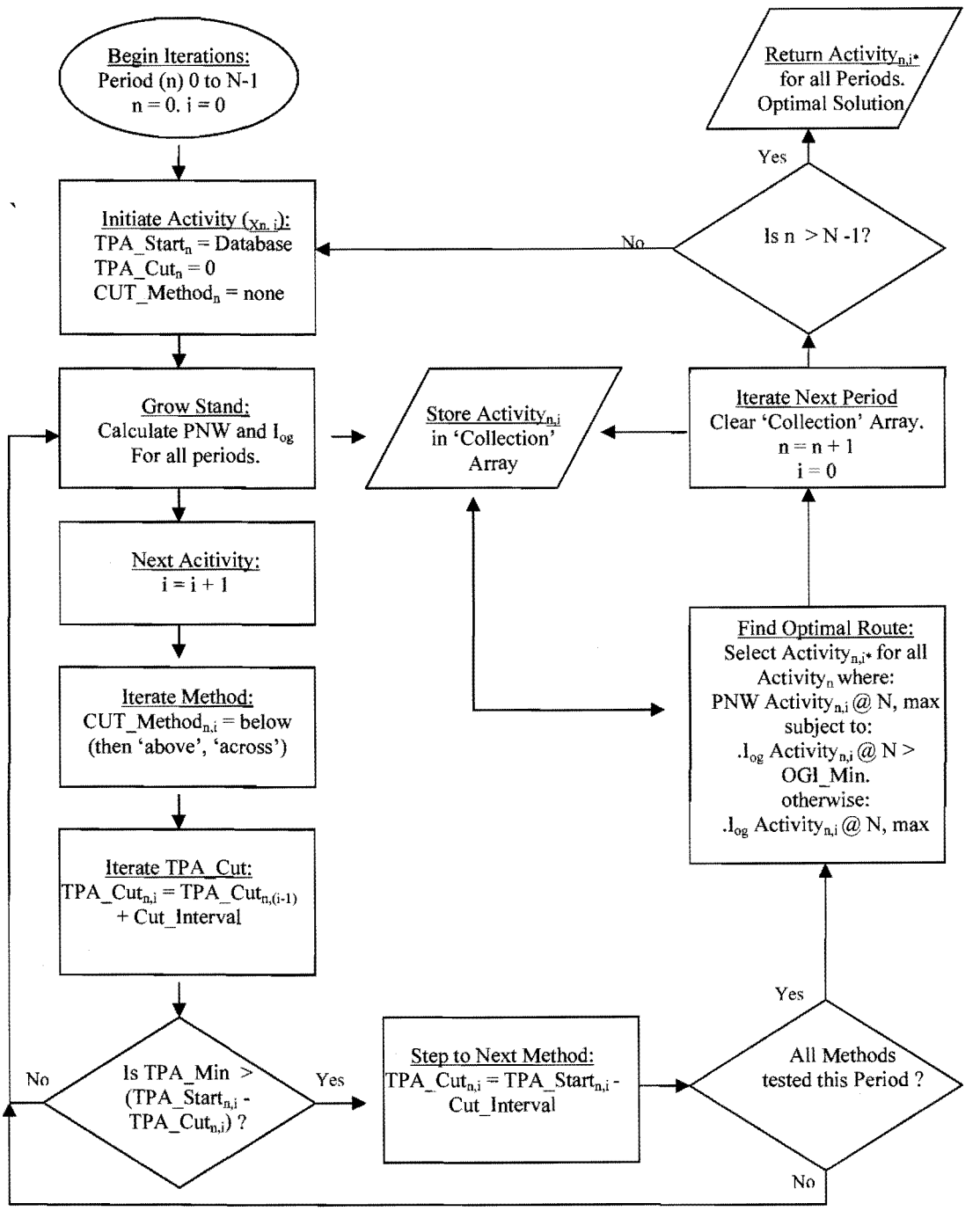


Figure 2.4. FPS Optimizer Model User Interface (1)

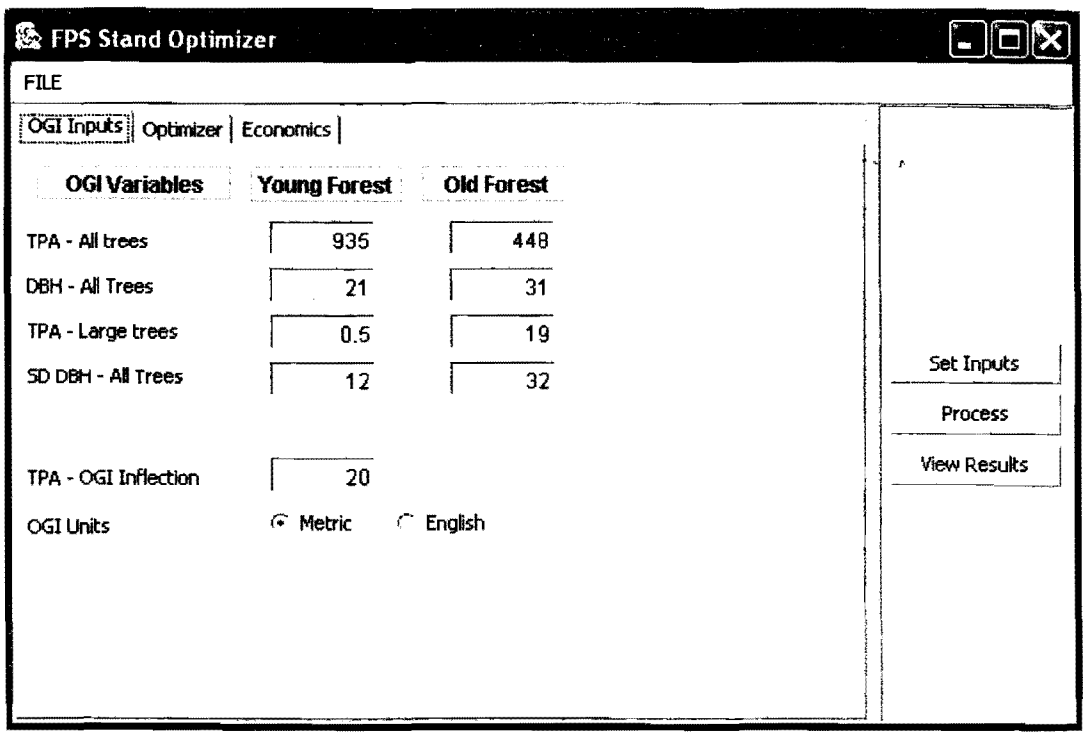


Figure 2.5. FPS Optimizer Model User Interface (2)

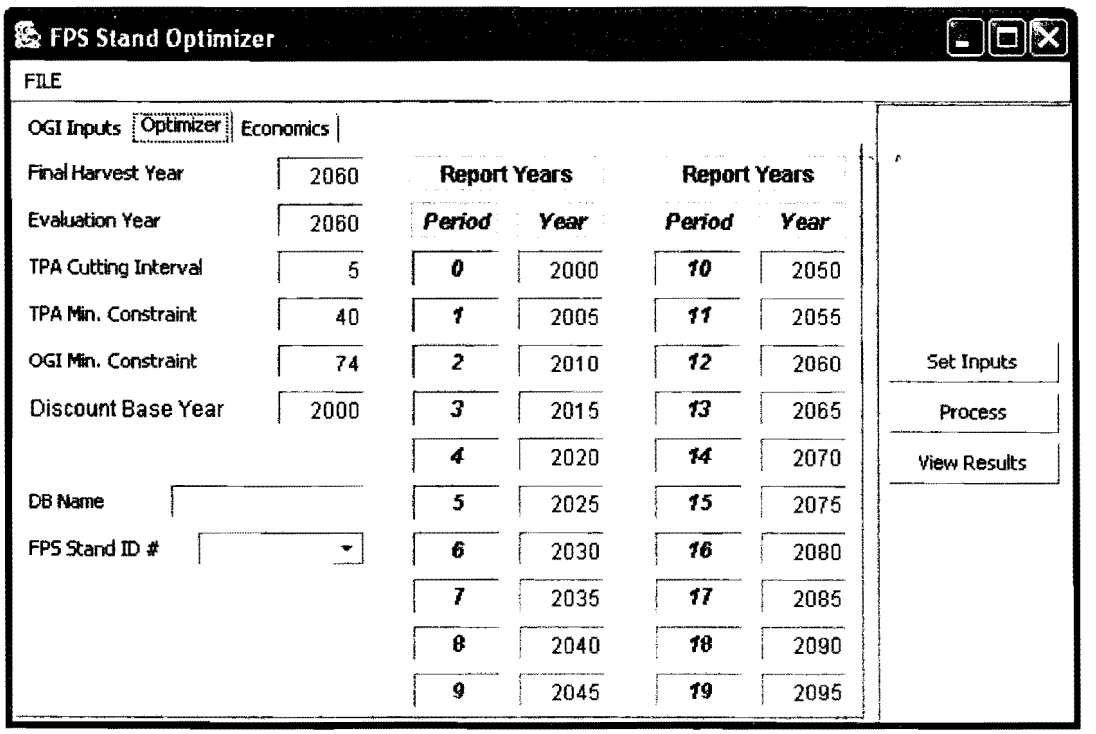


Figure 2.6. FPS Optimizer Model User Interface (3)

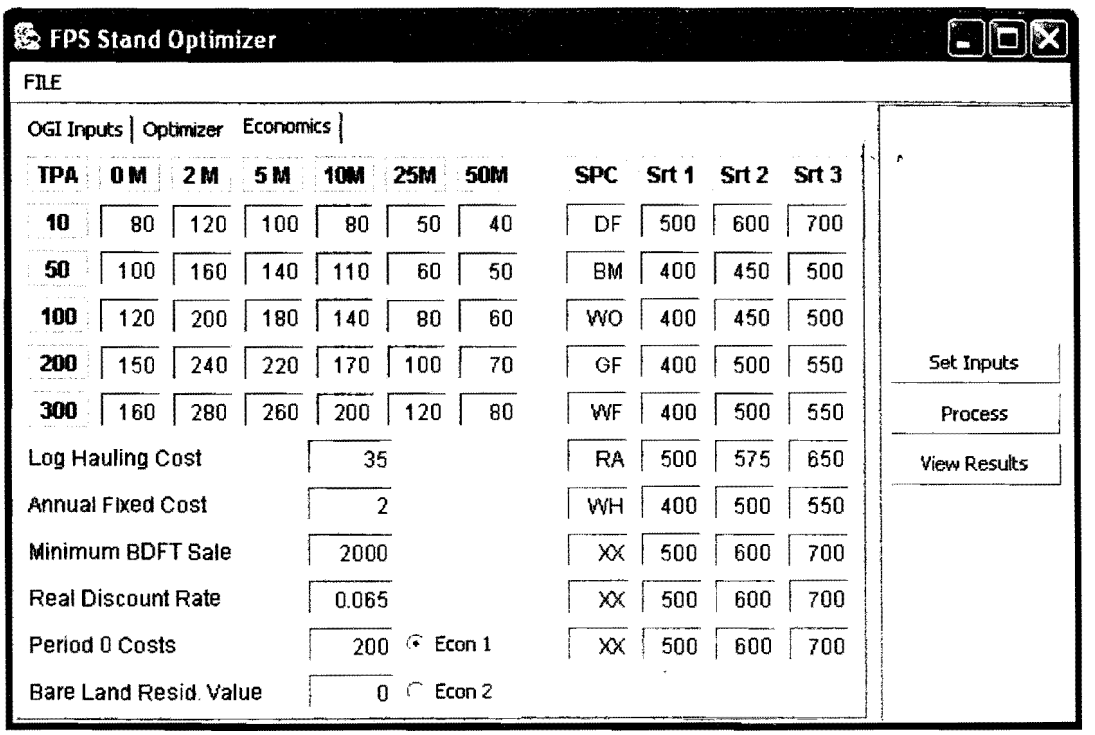


Figure 2.7. FPS Optimizer Model Sample Output

*** FPS OPTIMIZER RESULTS OUTPUT FILE ***

OGI_MIN : 65 END_YR : 2150 EVAL_YEAR : 2150 STAND: 5004
LOGGING COST : 182.5711 TPA Coef : 0.36402306 BFAC Coef : -0.8224207
PRE-LOG COST : 85.64528 TPA Coef : 0.27541456

MESSAGE: The Detailed Winning Rx is ===>

PERIOD: 0 YEAR: 2000 HARVEST? : true TPA CUT: 11.34
PRESCRIPTION => METHOD: 2 MV: 0 LEVEL: 165
TPA pre-CUT: 175.96 TPA post-cut: 164.62
BDFT CUT: 9,644.38 BDFT RESID.: 16,320.47
OGI VALUES => SD: 6.51 TPA_ALL: 175.96 TPA_LG: 0.54 DBH: 12.63
OGI :56.81 PNV: $10,852.36

MESSAGE: The Detailed Winning Rx is ===>

PERIOD: 1 YEAR: 2010 HARVEST? : true TPA CUT: 40.43
PRESCRIPTION => METHOD: 3 MV: 1 LEVEL: 85
TPA pre-CUT: 125.09 TPA post-cut: 84.66
BDFT CUT: 8,847.26 BDFT RESID.: 18,475.90
OGI VALUES => SD: 5.66 TPA_ALL: 125.09 TPA_LG: 0.00 DBH: 15.20
OGI :52.97 PNV: $11,892.71

MESSAGE: The Detailed Winning Rx is ===>

PERIOD: 2 YEAR: 2020 HARVEST? : false TPA CUT: 0.00
PRESCRIPTION => METHOD: 0 MV: 0 LEVEL: 0
TPA pre-CUT: 73.10 TPA post-cut: 73.10
BDFT CUT: 0.00 BDFT RESID.: 24,680.67
OGI VALUES => SD: 5.63 TPA_ALL: 73.10 TPA_LG: 0.00 DBH: 17.77
OGI :52.89 PNV: $10,875.33

STAND OPTIMIZATION RESULTS

In order to evaluate the dynamics of the FPS Optimizer Model and the late-seral index constraint, two approaches were taken. First, a 'generic' stand was created which emulates a typical industrial plantation (i.e. single-species, uniform stocking and age.) This industrial stand was run through the model optimizing on rotation ages from 25 to 200 years. Each period was evaluated under two different objective functions:

1. Maximize PNW_N, s.t. $_{log, N} > 0$
2. Maximize PNW_N, s.t. $_{log, N} >= 100$

In other words, maximize the PNW strictly without consideration for late-seral structure, and maximize late-seral structure only. A third projection was done simulating a 'no-harvest' policy.

The literature regarding the development of late-seral structure points to the prevalence of low-density, mixed species stands at early ages. In order to evaluate this, a second approach was taken using data from a young, naturally established forest in the central Oregon coast range. A single stand was selected from the Dolores Beazell Forest (Benton County Parks Department) and optimized under the same two objective functions as the generic stand. I will now discuss the explicit variables used in all applications and the results obtained.

Input Values for All Stands

All input values were kept consistent for each stand analyzed. Twenty periods were established for reporting, with optimization on a subset of those. Table 3.1 shows the input values used.

Table 3.1. FPS Optimizer Model Input Values Used

VARIABLE	**Value**
Discount rate, real	5.0 percent
TPA cutting interval	10
TPA minimum cutting level	80
Establishment cost (period 0, $/acre)	200
Residual value (final period, $/acre)	500
Commercial harvest cut-off (bf/acre)	2,000
Annual fixed cost ($/acre)	2
Trucking cost ($/Mbf)	35
Commercial harvest cost equation ($/Mbf)	$182.57 + 0.36(TPA_{cut}) - (.82)sqrt(bf/ac_{cut})$
Non-commercial treatment cost equation ($/acre)	$85.64 + 0.27(TPA_{cut})$

The activity treatment costs are obtained from the GUI after the cost matrix has been completed. Matrix values were obtained from personal experience with timber sales and logging contracts, and an interview with Frank Brown, forestry consultant at Duck Creek Associates, Inc. The application utilizes a multiple linear regression to convert the cost matrix values into a continuous cost equation. Commercial treatments (those operations resulting in > 2,000 board foot volume harvested) are a function of trees per acre and the square root of board foot volume per acre (this transformation captures much of the inverse proportionality shown in

the data sets analyzed.) Non-commercial activities are strictly a function of trees per acre removed (and assumed to be left on site).

Log prices are assumed to be constant and no real appreciation is incorporated over the periods analyzed. Table 3.2 shows the log prices used. Three log sorts are possible, based on scaling diameter: Sort # 1, 6 to 11 inches; Sort # 2, 12 to 15 inches; and Sort # 3, 16 inches and greater.

Table 3.2. Delivered Log Prices

SPECIES	Sort # 1	Sort # 2	Sort # 3
Douglas-fir	$500	$600	$700
white fir	$400	$500	$550
western hemlock	$400	$500	$550
red alder	$500	$575	$650
bigleaf maple	$400	$450	$500
Oregon oak	$400	$450	$500
other	$500	$600	$700

The financial impact of the late-seral constraint is evaluated using I_{og}. I_{og} is calibrated with mean values from 'young' and 'old' forests. The authors of the index computed these values from stands located primarily in the central Oregon coast range. I have chosen to use their values directly, although the FPS Optimizer Model allows for local calibration. Table 3.3 displays these values, both in metric and English units. English units are used throughout the remainder of the report. I have incorporated an additional variable 'OGI_Inflection' (set to 20 trees per acre) to represent an inflection point in the 'TPA_all' parameter. This correctly reverses

the I_{og} trend of that variable when an unusually low number of stems are observed (or when evaluating newly established plantations with high stem count but which have no measurable diameter at breast height).

Table 3.3. Old-Growth Index Parameter Values

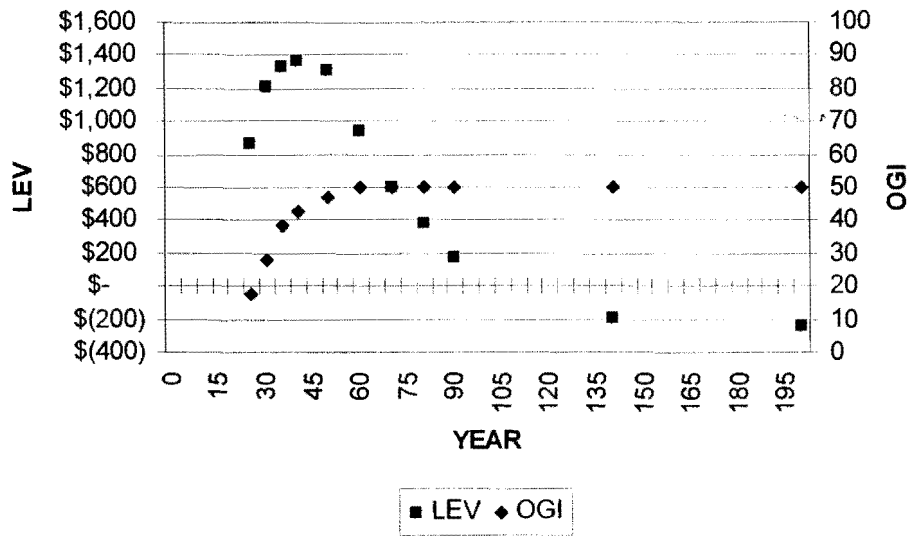
PARAMETER	'young' forest	'old' forest
SD OF DIAMETER (CM/INCH)	12/4.7	32/12.6
# of large trees (hectare/acre)	0.5/.2	19/7.7
Average DBH of all trees (cm/inch)	21/8.3	31/12.2
# of all trees (hectare/acre)	935/377	448/181

Results: Industrial Stand

The 'industrial' stand represents a planted (400 trees per acre) forest of Douglas-fir only. Site preparation, brush control, and uniform spacing was assumed for all projections. Figures' 3.1(a-c) show graphical and tabular results of the optimization runs under both objective functions, as well as the 'no-harvest' policy projection. Note that the two managed scenarios do not represent a single prescription, but a continuous surface of management trajectories. Each period's result is the final condition of the stand, given the optimal management path anticipating final harvest at that period. In theory, we are observing the most efficient pattern of LEV and I_{og} combinations for various final harvest ages, given

two different objectives. The tabular data contains the calculated statistics used in the I_{og} formulation.

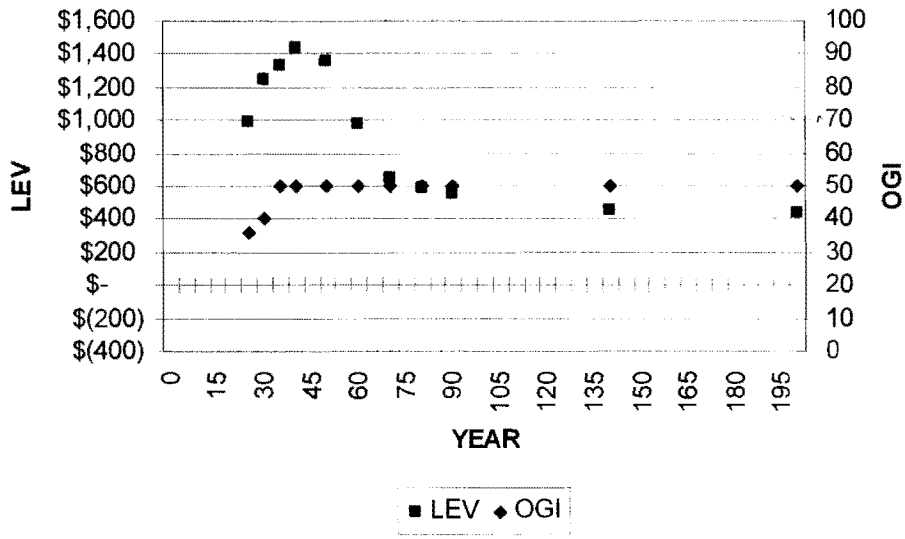
Figure 3.1(a). Industrial Stand, No Harvesting - Optimization Results



Industrial Stand - No Harvesting Activity

YEAR	LEV	OGI	SD_DBH	DBH	TPA_LG	TPA_ALL
25	$ 869	17.4	1.6	9.3	0.0	294.5
30	$ 1,211	28.2	1.6	10.6	0.0	274.2
35	$ 1,326	38.5	1.7	11.9	0.0	254.7
40	$ 1,369	43.0	1.8	12.9	0.0	236.8
50	$ 1,302	46.9	1.9	14.7	0.0	205.9
60	$ 942	50.0	2.1	16.1	0.0	180.9
70	$ 600	50.0	2.3	17.3	0.0	159.8
80	$ 384	50.0	2.4	18.3	0.0	144.6
90	$ 178	50.0	2.5	19.2	0.0	131.2
150	$ (194)	50.0	2.6	22.2	0.0	93.3
190	$ (238)	50.0	3.0	24.9	0.0	64.5

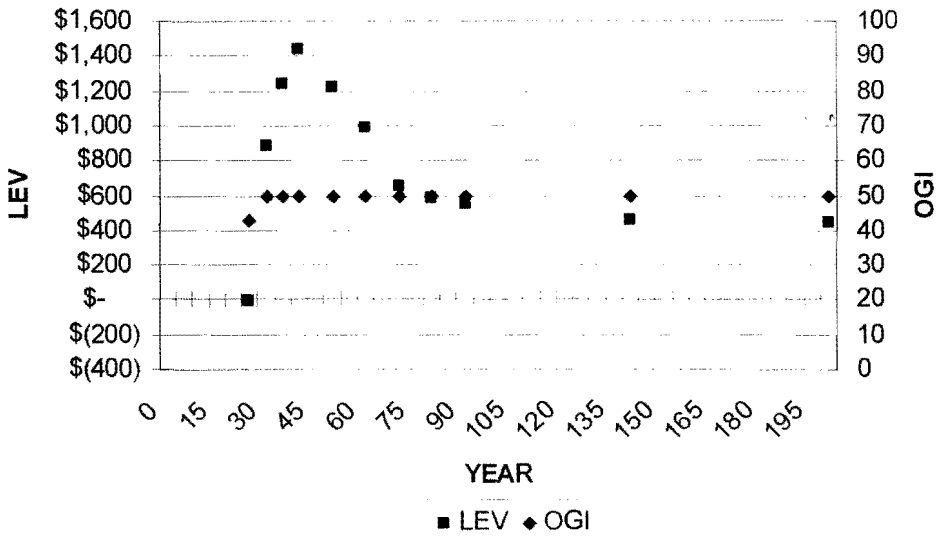
Figure 3.1(b). Industrial Stand, LEV Maximization - Optimization Results



Industrial Stand - LEV Maximization

YEAR	LEV	OGI	SD_DBH	DBH	TPA_LG	TPA_ALL
25	$ 986	36.0	0.6	10.5	0.0	204.7
30	$ 1,241	40.0	1.2	10.7	0.0	183.3
35	$ 1,337	50.0	0.7	12.9	0.0	157.3
40	$ 1,440	50.0	0.7	14.4	0.0	171.1
50	$ 1,352	50.0	0.8	16.2	0.0	139.0
60	$ 984	50.0	1.2	17.3	0.0	83.9
70	$ 653	50.0	1.1	19.2	0.0	72.6
80	$ 584	50.0	1.6	17.5	0.0	43.7
90	$ 549	50.0	1.6	18.6	0.0	41.2
150	$ 456	50.0	1.4	23.7	0.0	39.0
190	$ 445	50.0	1.5	25.8	0.0	34.0

Figure 3.1(c). Industrial Stand, OGI Maximization - Optimization Results



Industrial Stand - OGI Maximization

YEAR	LEV	OGI	SD_DBH	DBH	TPA_LG	TPA_ALL
25	$ (7)	42.8	0.0	11.1	0.0	64.2
30	$ 888	50.0	0.3	12.3	0.0	95.9
35	$ 1,241	50.0	0.8	13.2	0.0	115.3
40	$ 1,440	50.0	0.7	14.4	0.0	171.1
50	$ 1,222	50.0	0.5	15.5	0.0	118.3
60	$ 984	50.0	1.2	17.3	0.0	83.9
70	$ 653	50.0	1.1	19.2	0.0	72.6
80	$ 584	50.0	1.6	17.5	0.0	43.7
90	$ 549	50.0	1.6	18.6	0.0	41.2
150	$ 456	50.0	1.4	23.7	0.0	39.0
190	$ 445	50.0	1.5	25.8	0.0	34.0

The results shown in Figure's 3.1 are generally uninteresting, but support a few intuitive conclusions. It appears that both the LEV maximization and OGI maximization function yield greater economic returns than the no-harvest option, across most time horizons, and the financial return differentials are greatest at the longer final harvest ages ($549 per acre for both managed networks and $176 per acre for the no-harvest network at year 90.) Furthermore, it appears that both managed networks are capable of obtaining greater I_{og} values than the no-harvest network in the first few periods (50.0 for both managed networks and 38.5 for the no-harvest network at year 35.) All three networks appear to converge at an I_{og} value of 50 at or before year 60, and remain at that level for the duration of the analysis timeframe (190 years).

The most significant observation about this trial is the last case, the plateau in the I_{og} development at a value of 50. The results show that some harvesting activity can increase the rate of development in the late-seral index, and that the LEV maximization function is not initially in opposition to the I_{og} maximization function. It does appear that the industrial stand is limited in its ability to obtain complete late-seral structure as measured by the index (at least for the first 190 years). If we observe the four variables in the index, standard deviation of diameter and number of stems per acre of large trees are identified as the limiting attributes. Although growth projections beyond 200 years are likely outside the confidence limits of the underlying FPS model, several iterations beyond 200 years indicate that the industrial stand would not leave the plateau until the middle of the fourth

century. It is also probable that an index value of 75 may pose another plateau since the increase in the standard deviation of diameter variable is proceeding at a relatively slow rate. As the industrial stand is stocked with a single species and natural understory regeneration is not modeled, it is likely that this variable will continue to increase, but at a decreasing rate.

The management strategies taken for any given node depends largely on the timeframe available for management activities. The index variable that responds most rapidly to harvesting is obviously trees per acre of all species. Harvesting (of any of the three methods) has the immediate effect of increasing the index value related to this variable. However, given a longer time horizon, a drastic reduction in growing stock early on can have a negative effect on the ability of the other index variables to increase. In general, the optimization model selects thinning from above (and heavily) as an early treatment given a short time horizon for analysis, but switches to thinning across all diameters (in order to increase the standard deviation of diameter variable) given a longer time frame. As expected, the LEV maximization function tends to favor overstory thinnings as the marginal value growth rate slips below the discount rate, while the LEV maximization function favors understory thinnings in order to increase the stem count of large trees per acre (only in very long time horizons, however.)

A shortcoming in the underlying optimization model can be observed by comparing the results from both managed networks at year 35. Although both achieve an index value of 50 at this year, the optimal route taken for each differs,

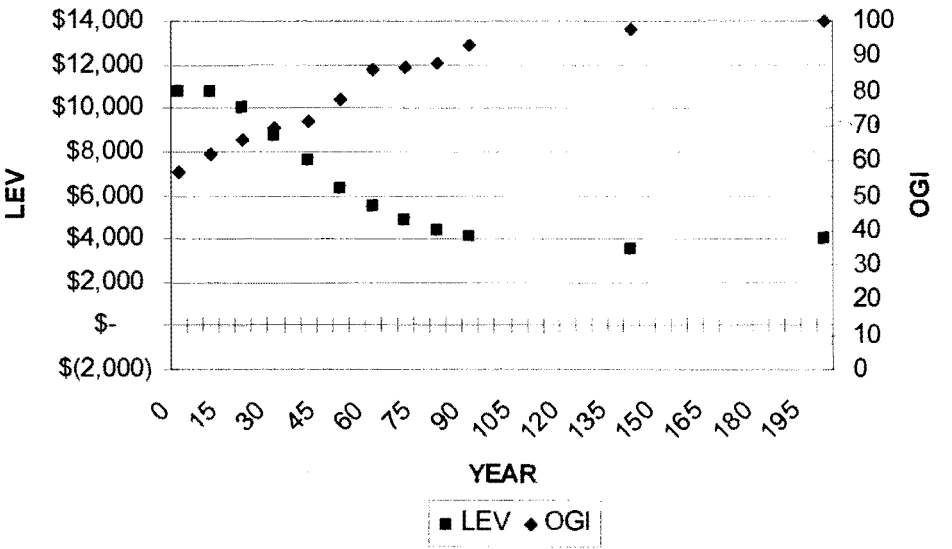
yielding different LEV's ($1,337 for the LEV maximization function and $1,241 for the OGI maximization function). It must be concluded that the optimization model is following the path of greatest marginal return for the objective function, and not necessarily the greatest absolute value (although both networks converge after year 60.)

Results: Natural Stand

The natural stand was selected at random from a set of 16 available stands on the Dolores Beazell Forest. The history of the forest is that typical of many unmanaged small woodland tracts in the Oregon coast range, that is, natural conifer and hardwood regeneration, following a clear-cut harvest prior to the Oregon Forest Practices Act. The stand analyzed (#5004) is predominately Douglas-fir, followed by bigleaf maple, red alder, and grand fir (176 stems and 194 square feet of basal area per acre). This stand contains an array of age classes, with the dominant conifers averaging 64 years of age.

The results of the optimization runs are shown in Figure's 3.2(a-c). The most notable observation is the ability of the naturally regenerated stand to achieve large late-seral index values in a relatively short period of time. The no-harvest network achieves an index value of 92.1 by year 140, and the OGI maximization network shows a value of 97.8 at the same year. If we add on the initial age of the overstory (64 years), we see that the natural stand achieves at least 90 percent of late-seral structure by age 200 (compared to 50 percent for the industrial stand.)

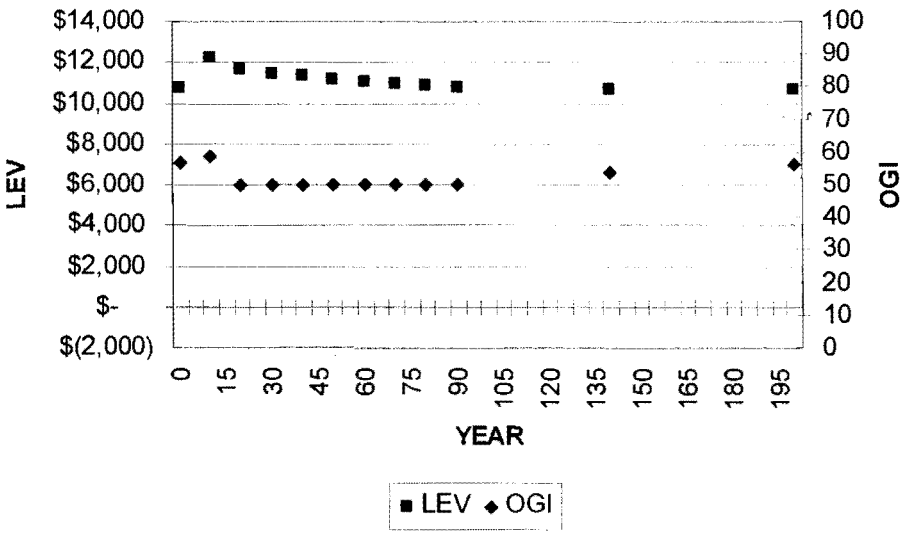
Figure 3.2(a). Natural Stand, No Harvest - Optimization Results



Natural Stand - No Harvesting Activity

YEAR	LEV	OGI	SD_DBH	DBH	TPA_LG	TPA_ALL
0	$10,745	56.8	6.5	12.6	0.5	176.0
10	$12,240	59.2	7.1	16.5	0.7	139.7
20	$10,866	62.9	7.4	19.4	1.5	116.6
30	$ 8,300	63.3	7.6	21.5	1.5	100.7
40	$ 5,872	67.6	7.7	23.1	2.6	89.1
50	$ 4,001	68.0	7.9	24.5	2.6	80.4
60	$ 2,658	73.9	8.0	25.6	4.2	74.1
70	$ 1,731	81.8	8.2	26.4	6.4	70.2
80	$ 1,122	82.4	8.5	27.1	6.4	66.7
90	$ 708	84.7	8.7	27.7	6.8	64.5
140	$ 39	92.1	10.1	29.9	10.0	56.6
200	$ (35)	98.5	12.1	32.5	12.9	50.2

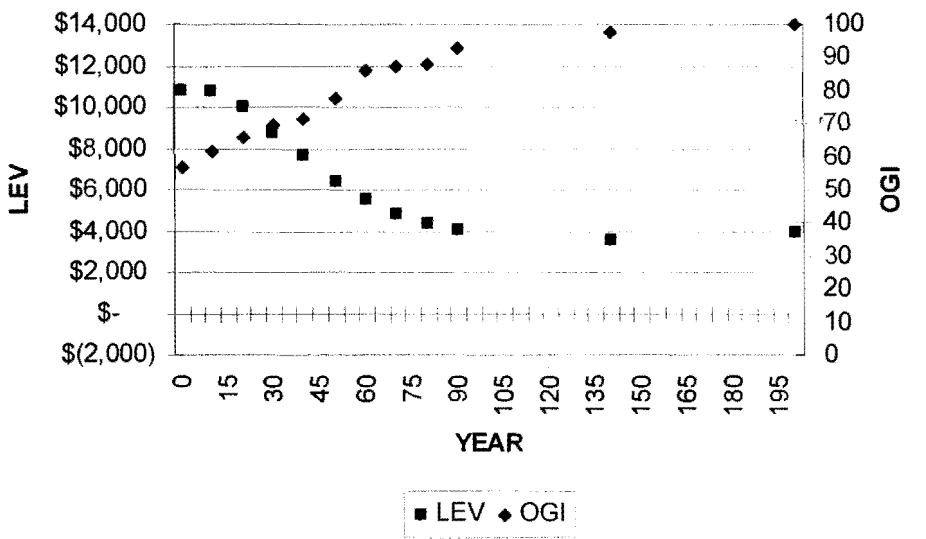
Figure 3.2(b). Natural Stand, LEV Maximization - Optimization Results



Natural Stand - LEV Maximization

YEAR	LEV	OGI	SD_DBH	DBH	TPA_LG	TPA_ALL
0	$10,745	56.8	6.5	12.6	0.5	176.0
10	$12,271	58.4	6.8	17.4	0.7	125.6
20	$11,654	50.0	2.3	13.3	0.0	42.3
30	$11,455	50.0	2.9	15.5	0.0	34.4
40	$11,376	50.0	3.4	17.4	0.0	29.4
50	$11,175	50.0	4.0	19.0	0.0	25.9
60	$11,051	50.0	4.7	20.5	0.0	23.3
70	$10,917	51.4	5.2	21.8	0.0	21.4
80	$10,836	52.8	5.6	22.8	0.0	20.2
90	$10,777	53.0	6.0	23.7	0.0	19.1
140	$10,658	53.5	7.4	27.0	0.0	16.0
200	$10,642	56.0	9.1	30.5	0.0	13.7

Figure 3.2(c). Natural Stand, OGI Maximization - Optimization Results



Natural Stand - OGI Maximization

YEAR	LEV	OGI	SD_DBH	DBH	TPA_LG	TPA_ALL
0	$10,745	56.8	6.5	12.6	0.5	176.0
10	$10,773	61.9	7.9	15.7	0.7	71.5
20	$10,030	65.6	8.3	18.8	1.5	61.3
30	$ 8,751	69.4	8.6	21.3	2.3	54.1
40	$ 7,614	71.4	8.9	23.3	2.6	49.1
50	$ 6,353	77.7	9.2	25.1	4.2	49.2
60	$ 5,495	86.1	9.6	26.6	6.4	46.1
70	$ 4,828	86.9	9.8	27.7	6.4	44.0
80	$ 4,371	87.7	10.1	28.7	6.4	42.3
90	$ 4,062	93.0	10.4	29.5	9.2	41.1
140	$ 3,520	97.8	11.9	33.1	9.4	36.4
200	$ 3,998	100.0	13.9	37.1	11.6	29.8

Another interesting observation is the ability of harvesting as a tool for increasing the late-seral index beyond that found without harvesting. Depending on the evaluation time horizon, the managed network (optimizing for LEV production) yields from 2 to 6 percent greater I_{og} values than the unmanaged counterpart. As interesting, is the impact on LEV - not only does harvesting appear to accelerate the development of late-seral attributes, it yields greater financial returns across most of the evaluation periods. The LEV differential is greatest as the final harvest year increases.

The management strategies selected by the optimization model for the natural stand also depend heavily on the time horizon for evaluation, as did the industrial stand. In general, the same rules seem to apply. Stem count is typically reduced early on to promote diameter growth and increase the index value of the total stems variable. The LEV objective function tends to favor overstory removals, while the OGI objective function promotes stand structure by harvesting across all diameters at early stages (balanced by the operational and financial constraints defined by the analyst).

CONCLUSIONS

A tool for evaluating the trade-off between silvicultural investment efficiency and acceleration of late seral-stage stand characteristics has been developed. Input information into the FPS Optimizer model is likely to be that which is gathered on a regular basis by landowners currently. Actually using it in an operational setting could be done immediately, but only with extreme caution due to the limitations of the FPS Growth and Yield Model silvicultural methods available and simplicity of the four-variable old-growth index. Potentially misleading results were produced by the model under certain conditions.

The limited testing of the FPS Optimizer undertaken for this paper has indicated certain relationships regarding late-seral structure development, some of which appear to be supported by previous research. Most notably, the model appears to support the following conclusions regarding silviculture applied to late-seral structure development:

1. Density management at early ages accelerates late-seral structure development, but has little effect on SEV or optimal rotation age.
2. Thinning across all diameter classes is generally the preferred method, given the limited three methods tested (thinning from above, below, and uniformly across all diameters.)
3. Clumpy and non-uniform stocking is a desirable condition for late-seral structure development.

Referring to Figure 3.2(b) and (c), Thinning was invoked (under most time horizons) within the first two decades. This resulted in an increase in the OGI under both LEV and OGI objectives when compared to the 'no-management' scenarios in Figure 3.2(a). The LEV was not diminished by these activities when observing the 'natural' case-study stand. The results of the 'industrial' stand, however, reveal the potential for misleading results regarding development of the OGI and stand density management. Referring to Figure 3.1(c) we observe that the 'industrial' stand fails to breach an OGI level of 50, even under the optimization objective of that function. This may lead a user of the FPS Optimizer model to incorrectly assume that a forest stand established under typical industrial standards cannot achieve late-seral structure. If we take a closer look at the progression of the four key OGI statistics, it is apparent that they do in fact continue to increase over time. However, they fail to achieve the minimum values required to be registered by OGI formulation. This is partially a function of the growth dynamics of uniformly stocked, even-aged stands, but also a limitation of the four-variable old-growth index and the available silvicultural techniques available in the FPS Growth and Yield Model. If the FPS Growth and Yield Model provided for group selection harvesting or mixed methods for a given operation period, the results may change dramatically. Furthermore, if the OGI formulation were expanded to include additional relevant stand statistics, the observed relationship of OGI development over time may prove to be more dynamic for these stands.

The previously noted shortcoming of the FPS Optimizer model in the 'industrial' stand raises another issue regarding the appropriateness of the MSPATH algorithm for stand-level optimization of an OGI objective function. The MSPATH algorithm is generally accepted as a suitable method for optimizing concave production functions. We have observed a stair-step function in the 'industrial' stand under the OGI objective function, looking at Figure 3.1(c). The MSPATH algorithm may not yield an optimal solution for this type of functional form – more work is needed to validate the appropriateness of this technique in this situation and/or implement a more suitable algorithm.

The second conclusion regarding late-seral development relates to the propensity for the FPS Optimizer to select thinning across all diameters as the most useful method for increasing OGI. This conclusion, however, is only valid given the three silvicultural methods available from the FPS Growth and Yield Model. As previously mentioned, some form of group selection might serve as a superior method for increasing the OGI of a stand due to its effect on standard deviation of diameters (one of the four structural variables in the old-growth index.) Thinning from below was often selected as a treatment under LEV objective function, referring to Figures 3.1(b) and 3.2(b), primarily under short-term outlooks. This is due to the effect on increasing scribner board foot volume. Longer analysis time-frames reverted to thinning from above in order to capture volume (and value) that was forecasted to grow at a slower rate than the user-defined discount rate.

The third conclusion is only loosely supported by the limited testing of the model, but current research suggests similar conclusions (Tappeiner, et al., 1997). The FPS Growth and Yield Model, during compilation of stand plot data, calculates a 'clumpiness index'. This index represents the spatial distribution of tree crowns within a stand. The 'clumpiness index' calculates a value from 0 to 1, in which 1 represents perfect uniform stocking and decreasing values representing an increase in clumpiness. The 'natural' stand in this study had an initial 'clumpiness index' value of 0.91, while the 'industrial' stand started at 1.00. This conclusion could be investigated further with more case-study trials and possibly the incorporation of the clumpiness index into the OGI formulation.

The trial results, due to the notable difference in OGI development between 'natural' and 'industrial' stands, raises many questions. Primarily, are the results a function of the considerable difference in initial conditions of the two stands, or an artifact of this modeling approach. As observed, the industrial stand demonstrated a significant inability to achieve the second stage of late-seral development for an extended period of time. Was this due to the homogenous and uniform initial condition of the planted stand or the limitations of the underlying growth and yield model or OGI formulation? On the other hand, the natural stand showed a fluid, positive development trajectory towards the ideal old-growth condition and provided a greater amount of flexibility in silvicultural options and economic mitigation. Additional work to address these issues could prove insightful. If in fact the initial condition of a stand largely defines the future potential and options

for the forest manager, forest management policy might be affected. It should be stressed that the case-study examples are exploratory in nature and too limited to draw any firm conclusions regarding late-seral development – they merely point out some opportunities and areas for further work. The biological output generated by the model has not been validated with contemporary research data and should be done so before implementing the model as a reliable tool.

A positive aspect of the results is the apparent rejection of an 'all or nothing' proposition regarding late-seral structure development and financial returns. The results appear to indicate that although the late-seral constraint reduces potential economic returns to the landowner, it does not eliminate them entirely.

Much work can still be done utilizing an index-based approached as proposed here. As mentioned previously, the model developed only utilizes four descriptors for assessing late-seral structure. One could envision an eight descriptor model, including the following additional variables: snag density, downed log density, stand clumpiness, and a minor species indicator. All of these variables are currently being inventoried by most landowners in addition to the traditional measurements and can be modeled with moderate modifications to the program, resulting in a more robust index.

Another area worthy of investigation is the marginal cost of increasing levels of the late-seral constraint for various stand types and final harvest ages. This study took a simplistic approach, analyzing two stands of significantly

different character, with either a pure profit maximizing objective or a late-seral development objective. The FPS Optimizer model, however, was designed to allow incremental changes in the old-growth index constraint. Multiple iterations of the model at incremental changes in the old-growth constraint could be used to develop a marginal cost relationship for this variable. It could be used to target regions of efficiency in which late-seral development is either complimentary to financial return objectives, or at a minimum, define areas where the late-seral objective can be achieved for minimal reductions in financial returns.

LITERATURE CITED

Acker, S.A., T.E. Sabin, L.M. Ganio, and W.A. McKee. 1997. Development of Old_Growth Structure and Timber Volume Growth Trends in Maturing Douglas-fir Stands. Forest Ecology and Management. 104: 265-280.

Arney, James D. 1995-2002. Forest Projection System software.

Brodie, J.D., D.M. Adams and C. Kao. 1978. Analysis of Economic Impacts on thinning and Rotation for Douglas-fir using Dynamic Programming. Forest Science. 24: 513-22.

Brodie, J.D., and C. Kao. 1979. Optimizing Thinning in Douglas-fir with Three-Descriptor Dynamic Programming to Account for Accelerated Diameter Growth. Forest Science. 25: 655-672.

Cole, Elizabeth C. 1996. Managing for Mature Habitat in Production Forests of Western Oregon and Washington. Weed Technology. 10: 422-428.

Franklin, Jerry F., and Thomas A. Spies. 1991. Ecological Definitions of Old-Growth Douglas-Fir Forests. USDA Forest Service General Technical Report PNW-GTR-285.

Franklin, Jerry F., and Thomas A. Spies 1991. Compostition, Function, and Structure of Old-Growth Douglas-Fir Forests. USDA Forest Service General Technical Report PNW-GTR-285.

Haight, R.G., J.D. Brodie., W.G. Dahms. 1985. A Dynamic Programming Algorithm for Optimization of Lodgepole Pine Management. Forest Science. 31: 321-330.

Hicks, Lorin L., Henning C. Stabbins, and Dale R. Herter. 1999. Designing Spotted Owl Habitat in a Managed Forest. Journal of Forestry. July: 20-25.

Marcot, Bruce G. 1991. Old-Growth Inventories: Status, Defintions, and Visions for the Future. USDA Forest Service General Technical Report PNW-GTR-285.

McComb, William C., T.A. Spies, and W.H. Emmingham. 1993. Douglas-fir Forests: Managing for Timber and Mature-Forest Habitat. Journal of Forestry. 91: 31-49.

Paredes V., G.L. and J.D. Brodie. 1987. Efficient Specification and Solution of the Even-aged Rotation and Thinning Problem. Forest Science. 33: 14-29.

Perry, David A., and Jumanne Maghembe. 1989. Ecosystem Concepts and Current Trends in Forest Management: Time for Reappraisal. Forest Ecology and Management. 26:123-140.

Semeliss, Wade. 2002. FPS Optimizer Model.

Tappeiner, John C., D. Huffman, D. Marshall, T.A. Spies, and J.D. Bailey. 1997. Density, ages, and growth rates in old-growth and young-growth forests in coastal Oregon. Canadian Journal of Forest Resources. 27: 638-648.

Yoshimoto, A., G.L., Paredes V., and J.D. Brodie. 1988. Efficient Optimization of an Individual Tree Growth Model. USDA Forest Service General Technical Report RM-161: 154-162.

Yoshimoto, A., R.G. Haight, and J.D. Brodie. 1990. A Comparison of the Pattern Search Algorithm and the Modified PATH Algorithm for Optimizing and Individual Tree Model. Forest Science. 36: 394-412.

APPENDICES

Appendix A. FPS Optimizer Model software code

```
package thesis;

/*****************************************************************************
 * 'Master' Class
 *
 * FUNCTION: This class is the main driver for the application
 *****************************************************************************/

import java.text.NumberFormat;
import javax.swing.UIManager;
import java.awt.*;

public class Master
{

  // --- Variable identifying wether Frame should be packed
  boolean packFrame = false;


//============================================================================
=====
  // 'Master' Class Constructor
  //
  // FUNCTION: Activates the GUI for the program

//============================================================================
=====
  public Master ()
  {
    // --- Create instance of the 'OptiFrame' Class
    OptiFrame frame = new OptiFrame ();
    Database.load_driver();

    // --- Pack the frame depending on presets
    if (packFrame)
    {
      frame.pack();
    }
    else
    {
      frame.validate();
    }

    // --- Center the frame and adjust size if necessary
    Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
    Dimension frameSize = frame.getSize();
    if (frameSize.height > screenSize.height)
    {
```

```
    frameSize.height = screenSize.height;
   }
   if (frameSize.width > screenSize.width)
   {
    frameSize.width = screenSize.width;
   }
   frame.setLocation((screenSize.width - frameSize.width) / 2,
    (screenSize.height - frameSize.height) / 2);
   frame.setVisible(true);
  }// FINISH: 'Master' Class Constructor


  public static void main (String[] args) throws Exception
  {
   // --- Get the systems UI properties and set
   try
   {
    UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
   }
   catch (Exception uierror)
   {
    uierror.printStackTrace();
   }

   // --- Call the UI
   new Master ();

   // --- Create Control Instance which starts the entire Analysis
   //System.out.println("Setting Inputs");
   //Inputs.setInputs();
   //System.out.println("Loading DB driver");
   //Database.load_driver();
   //System.out.println("Loading DB path/name");
   //Database.load_database();
   //Control masterControl = new Control();
  }
  // FINISH: 'MAIN' Method


}
// FINSIH: 'MASTER' Public class
```

```
package thesis;

import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
import com.borland.jbcl.layout.*;
import javax.swing.border.*;
import java.io.*;
import java.lang.Runtime;
import java.lang.Process;
import com.borland.jbcl.layout.VerticalFlowLayout;
import java.sql.*;


public class OptiFrame extends JFrame
{
 JPanel contentPane;
 JMenuBar jMenuBarOpti = new JMenuBar();
 JMenu jMenuOpti = new JMenu();
 JMenuItem jMenuItemClose = new JMenuItem();
 BorderLayout borderLayoutGlobal = new BorderLayout();
 JPanel eastPanel = new JPanel();
 JTabbedPane inputsTabbedPane = new JTabbedPane();
 BorderLayout borderLayout1 = new BorderLayout();
 JPanel optiPanel1 = new JPanel();
 VerticalFlowLayout verticalFlowLayout1 = new VerticalFlowLayout();
 JButton optiButton1 = new JButton();
 JButton optiButton2 = new JButton();
 JButton optiButton3 = new JButton();
 BorderLayout borderLayout2 = new BorderLayout();
 BorderLayout borderLayout3 = new BorderLayout();
 JPanel jPanel1 = new JPanel();
 JLabel jLabel1 = new JLabel();
 JLabel jLabel2 = new JLabel();
 JLabel jLabel4 = new JLabel();
 JLabel jLabel3 = new JLabel();
 VerticalFlowLayout verticalFlowLayout2 = new VerticalFlowLayout();
 JPanel jPanel2 = new JPanel();
 JLabel jLabel5 = new JLabel();
 TitledBorder titledBorder1;
 FlowLayout flowLayout1 = new FlowLayout();
 JLabel jLabel6 = new JLabel();
 JLabel jLabel7 = new JLabel();
 TitledBorder titledBorder2;
 JPanel jPanel3 = new JPanel();
 JTextField jTextField1 = new JTextField();
 FlowLayout flowLayout2 = new FlowLayout();
 JTextField jTextField2 = new JTextField();
 JPanel jPanel4 = new JPanel();
 JTextField jTextField3 = new JTextField();
 JTextField jTextField4 = new JTextField();
 JLabel jLabel8 = new JLabel();
```

```
JLabel jLabel9 = new JLabel();
JTextField jTextField5 = new JTextField();
JLabel jLabel10 = new JLabel();
JTextField jTextField6 = new JTextField();
JLabel jLabel11 = new JLabel();
JTextField jTextField7 = new JTextField();
JTextField jTextField8 = new JTextField();
JLabel jLabel12 = new JLabel();
JLabel jLabel13 = new JLabel();
JTextField jTextField9 = new JTextField();
JLabel jLabel14 = new JLabel();
JLabel jLabel15 = new JLabel();
JLabel jLabel16 = new JLabel();
JRadioButton jRadioButton1 = new JRadioButton();
JRadioButton jRadioButton2 = new JRadioButton();
ButtonGroup buttonGroup1 = new ButtonGroup();
JPanel jPanel5 = new JPanel();
JPanel jPanel6 = new JPanel();
JLabel jLabel17 = new JLabel();
JLabel jLabel18 = new JLabel();
JLabel jLabel19 = new JLabel();
JLabel jLabel20 = new JLabel();
JTextField jTextField10 = new JTextField();
JLabel jLabel21 = new JLabel();
JTextField jTextField11 = new JTextField();
JLabel jLabel22 = new JLabel();
JTextField jTextField12 = new JTextField();
JLabel jLabel23 = new JLabel();
JTextField jTextField13 = new JTextField();
JLabel jLabel24 = new JLabel();
JTextField jTextField14 = new JTextField();
JLabel jLabel25 = new JLabel();
JTextField jTextField15 = new JTextField();
JLabel jLabel26 = new JLabel();
JTextField jTextField16 = new JTextField();
JLabel jLabel27 = new JLabel();
JTextField jTextField17 = new JTextField();
JLabel jLabel28 = new JLabel();
JTextField jTextField18 = new JTextField();
JLabel jLabel29 = new JLabel();
JTextField jTextField19 = new JTextField();

JLabel jLabel30 = new JLabel();
JLabel jLabel31 = new JLabel();
JLabel jLabel32 = new JLabel();
JLabel jLabel33 = new JLabel();
JTextField jTextField20 = new JTextField();
JLabel jLabel34 = new JLabel();
JTextField jTextField21 = new JTextField();
JLabel jLabel35 = new JLabel();
JTextField jTextField22 = new JTextField();
JLabel jLabel36 = new JLabel();
```

```
JTextField jTextField23 = new JTextField();
JLabel jLabel37 = new JLabel();
JTextField jTextField24 = new JTextField();
JLabel jLabel38 = new JLabel();
JTextField jTextField25 = new JTextField();
JLabel jLabel39 = new JLabel();
JTextField jTextField26 = new JTextField();
JLabel jLabel40 = new JLabel();
JTextField jTextField27 = new JTextField();
JLabel jLabel41 = new JLabel();
JTextField jTextField28 = new JTextField();
JLabel jLabel42 = new JLabel();
JTextField jTextField29 = new JTextField();

JLabel jLabel43 = new JLabel();
JTextField jTextField30 = new JTextField();
JLabel jLabel44 = new JLabel();
JTextField jTextField31 = new JTextField();
JLabel jLabel45 = new JLabel();
JTextField jTextField32 = new JTextField();
JLabel jLabel46 = new JLabel();
JTextField jTextField33 = new JTextField();
JLabel jLabel47 = new JLabel();
JTextField jTextField34 = new JTextField();
JLabel jLabel48 = new JLabel();
JLabel jLabel49 = new JLabel();
JTextField jTextField36 = new JTextField();

FlowLayout flowLayout3 = new FlowLayout();
JPanel jPanel7 = new JPanel();
FlowLayout flowLayout4 = new FlowLayout();
JPanel jPanel8 = new JPanel();
FlowLayout flowLayout5 = new FlowLayout();
JMenuItem jMenuItem1 = new JMenuItem();

private String dbName = null;
private String dbPath = null;
private String [] stands;
private String theStand = null;

JPanel jPanel9 = new JPanel();
JPanel jPanel10 = new JPanel();
JPanel jPanel11 = new JPanel();
JPanel jPanel12 = new JPanel();
BorderLayout borderLayout4 = new BorderLayout();
FlowLayout flowLayout6 = new FlowLayout();
FlowLayout flowLayout7 = new FlowLayout();

JLabel jLabelspc = new JLabel();
JLabel jLabelsrt1 = new JLabel();
JLabel jLabelsrt2 = new JLabel();
JLabel jLabelsrt3 = new JLabel();
```

```
JTextField jTextFieldp1 = new JTextField();
JTextField jTextFieldp2 = new JTextField();
JTextField jTextFieldp3 = new JTextField();
JTextField jTextFieldp4 = new JTextField();
JTextField jTextFieldp5 = new JTextField();
JTextField jTextFieldp6 = new JTextField();
JTextField jTextFieldp7 = new JTextField();
JTextField jTextFieldp8 = new JTextField();
JTextField jTextFieldp9 = new JTextField();
JTextField jTextFieldp10 = new JTextField();
JTextField jTextFieldp11 = new JTextField();
JTextField jTextFieldp12 = new JTextField();
JTextField jTextFieldp13 = new JTextField();
JTextField jTextFieldp14 = new JTextField();
JTextField jTextFieldp15 = new JTextField();
JTextField jTextFieldp16 = new JTextField();
JTextField jTextFieldp17 = new JTextField();
JTextField jTextFieldp18 = new JTextField();
JTextField jTextFieldp19 = new JTextField();
JTextField jTextFieldp20 = new JTextField();
JTextField jTextFieldp21 = new JTextField();
JTextField jTextFieldp22 = new JTextField();
JTextField jTextFieldp23 = new JTextField();
JTextField jTextFieldp24 = new JTextField();
JTextField jTextFieldp25 = new JTextField();
JTextField jTextFieldp26 = new JTextField();
JTextField jTextFieldp27 = new JTextField();
JTextField jTextFieldp28 = new JTextField();
JTextField jTextFieldp29 = new JTextField();
JTextField jTextFieldp30 = new JTextField();
JTextField jTextFieldp31 = new JTextField();
JTextField jTextFieldp32 = new JTextField();
JTextField jTextFieldp33 = new JTextField();
JTextField jTextFieldp34 = new JTextField();
JTextField jTextFieldp35 = new JTextField();
JTextField jTextFieldp36 = new JTextField();
JTextField jTextFieldp37 = new JTextField();
JTextField jTextFieldp38 = new JTextField();
JTextField jTextFieldp39 = new JTextField();
JTextField jTextFieldp40 = new JTextField();

JLabel jLabeltpa = new JLabel();
JLabel jLabelbf0 = new JLabel();
JLabel jLabelbf1 = new JLabel();
JLabel jLabelbf2 = new JLabel();
JLabel jLabelbf3 = new JLabel();
JLabel jLabelbf4 = new JLabel();
JLabel jLabelbf5 = new JLabel();
JLabel jLabelbf6 = new JLabel();
JTextField jTextFieldc1 = new JTextField();
JTextField jTextFieldc2 = new JTextField();
JTextField jTextFieldc3 = new JTextField();
```

```
JTextField jTextFieldc4 = new JTextField();
JTextField jTextFieldc5 = new JTextField();
JTextField jTextFieldc6 = new JTextField();
JLabel jLabelbf7 = new JLabel();
JTextField jTextFieldc7 = new JTextField();
JTextField jTextFieldc8 = new JTextField();
JTextField jTextFieldc9 = new JTextField();
JTextField jTextFieldc10 = new JTextField();
JTextField jTextFieldc11 = new JTextField();
JTextField jTextFieldc12 = new JTextField();
JLabel jLabelbf8 = new JLabel();
JTextField jTextFieldc13 = new JTextField();
JTextField jTextFieldc14 = new JTextField();
JTextField jTextFieldc15 = new JTextField();
JTextField jTextFieldc16 = new JTextField();
JTextField jTextFieldc17 = new JTextField();
JTextField jTextFieldc18 = new JTextField();
JLabel jLabelbf9 = new JLabel();
JTextField jTextFieldc19 = new JTextField();
JTextField jTextFieldc20 = new JTextField();
JTextField jTextFieldc21 = new JTextField();
JTextField jTextFieldc22 = new JTextField();
JTextField jTextFieldc23 = new JTextField();
JTextField jTextFieldc24 = new JTextField();
JLabel jLabelbf10 = new JLabel();
JTextField jTextFieldc25 = new JTextField();
JTextField jTextFieldc26 = new JTextField();
JTextField jTextFieldc27 = new JTextField();
JTextField jTextFieldc28 = new JTextField();
JTextField jTextFieldc29 = new JTextField();
JTextField jTextFieldc30 = new JTextField();
JLabel jLabelbf11 = new JLabel();
JTextField jTextFieldc31 = new JTextField();
JLabel jLabelbf12 = new JLabel();
JTextField jTextFieldc32 = new JTextField();
JLabel jLabelbf13 = new JLabel();
JTextField jTextFieldc33 = new JTextField();
JLabel jLabelbf15 = new JLabel();
JTextField jTextFieldc35 = new JTextField();
JComboBox jComboBox1 = new JComboBox();
JLabel jLabel410 = new JLabel();
JLabel jLabelbf20 = new JLabel();
JTextField jTextFieldc50 = new JTextField();
JLabel jLabelbf21 = new JLabel();
JTextField jTextFieldc51 = new JTextField();
JRadioButton jRadioButton3 = new JRadioButton();
JRadioButton jRadioButton4 = new JRadioButton();
ButtonGroup buttonGroup2 = new ButtonGroup();
JLabel jLabelbf14 = new JLabel();
JTextField jTextFieldc34 = new JTextField();
```

```
public OptiFrame()
{
 try
 {
  jbInit();
 }
 catch(Exception e)
 {
  e.printStackTrace();
 }
}
private void jbInit() throws Exception
{

 // --- The Action Listener
 jMenuItemClose.addActionListener(new java.awt.event.ActionListener()
 {
  public void actionPerformed(ActionEvent e)
  {
   jMenuItemClose_actionPerformed(e);
  }
 });

 // --- Main UI parameters
 this.setTitle("FPS Stand Optimizer");
 this.setDefaultCloseOperation(3);
 this.setSize(new Dimension(600, 400));
 this.getContentPane().setLayout(borderLayout1);
 borderLayout1.setHgap(5);
 borderLayout1.setVgap(5);

 // --- Menu Bar

 jMenuItemClose.setActionCommand("Exit");
 jMenuItem1.setActionCommand("Select DB");
 jMenuItem1.setText("Select DB");

 jMenuItem1.addActionListener(new java.awt.event.ActionListener()
 {
  public void actionPerformed(ActionEvent e)
  {
   jMenuItem1_actionPerformed(e);
  }
 });

 optiButton1.addActionListener(new java.awt.event.ActionListener()
 {
  public void actionPerformed(ActionEvent e)
  {
   optiButton1_actionPerformed(e);
  }
 });
```

```
jRadioButton1.setActionCommand("1");
jRadioButton2.setActionCommand("2");
optiButton3.addActionListener(new java.awt.event.ActionListener()
{
 public void actionPerformed(ActionEvent e)
 {
  optiButton3_actionPerformed(e);
 }
});

optiButton2.addActionListener(new java.awt.event.ActionListener()
{
 public void actionPerformed(ActionEvent e)
 {
  optiButton2_actionPerformed(e);
 }
});


jComboBox1.setPreferredSize(new Dimension(95, 21));
jComboBox1.setEditable(true);
jComboBox1.addActionListener(new java.awt.event.ActionListener()
{
 public void actionPerformed(ActionEvent e)
 {
  jComboBox1_actionPerformed(e);
 }
});
jLabel410.setPreferredSize(new Dimension(85, 21));
jLabel410.setText("FPS Stand ID #");
jLabel410.setHorizontalAlignment(SwingConstants.LEFT);
jRadioButton3.setPreferredSize(new Dimension(63, 21));
jRadioButton3.setText("Econ 1");
jRadioButton3.setActionCommand("1");
jRadioButton4.setPreferredSize(new Dimension(63, 21));
jRadioButton4.setText("Econ 2");
jRadioButton4.setActionCommand("2");
jLabelbf14.setFont(new java.awt.Font("Dialog", 0, 12));
jLabelbf14.setHorizontalAlignment(SwingConstants.LEFT);
jLabelbf14.setText("Discount Base Year");
jLabelbf14.setPreferredSize(new Dimension(130, 21));
jTextFieldc34.setHorizontalAlignment(SwingConstants.RIGHT);
jTextFieldc34.setText("2000");
jTextFieldc34.setPreferredSize(new Dimension(50, 21));
jMenuBarOpti.add(jMenuOpti);
jMenuOpti.addSeparator();
jMenuOpti.add(jMenuItem1);
jMenuOpti.add(jMenuItemClose);
jMenuOpti.setText("FILE");
this.setJMenuBar(jMenuBarOpti);
jMenuBarOpti.setBorder(BorderFactory.createRaisedBevelBorder());
```

```
jMenuItemClose.setText("Exit");

// --- EAST Panel of MAIN frame and it's Buttons
this.getContentPane().add(eastPanel, BorderLayout.EAST);
eastPanel.setBorder(BorderFactory.createLoweredBevelBorder());
eastPanel.setLayout(verticalFlowLayout1);
verticalFlowLayout1.setAlignment(VerticalFlowLayout.MIDDLE);
flowLayout1.setAlignment(FlowLayout.LEFT);
flowLayout1.setHgap(10);
flowLayout1.setVgap(10);
optiButton1.setText("Set Inputs");
optiButton2.setText("Process");
optiButton3.setText("View Results");
optiButton1.setRequestFocusEnabled(false);
optiButton2.setRequestFocusEnabled(false);
optiButton3.setRequestFocusEnabled(false);
eastPanel.add(optiButton1, null);
eastPanel.add(optiButton2, null);
eastPanel.add(optiButton3, null);

// --- CENTER Panel 'InputsTabbedPane' of MAIN GUI has Tabbed Pane
// --- with multiple Panes
this.getContentPane().add(inputsTabbedPane, BorderLayout.CENTER);
inputsTabbedPane.setBorder(BorderFactory.createEtchedBorder());
inputsTabbedPane.add(optiPanel1, "OGI Inputs");
inputsTabbedPane.add(jPanel5, "Optimizer");
inputsTabbedPane.add(jPanel9, "Economics");


// --- ' OptiPanel 1' : The FIRST Tabbed Pane 'OGI Inputs' which contains 4 panels
optiPanel1.setPreferredSize(new Dimension(50, 10));
optiPanel1.setLayout(borderLayout2);
borderLayout2.setHgap(5);
borderLayout2.setVgap(5);

// --- Add 'Panel 1 - 4' to 'OptiPane' (which is 1st tabbed pane)
optiPanel1.add(jPanel4, BorderLayout.EAST);
optiPanel1.add(jPanel3, BorderLayout.CENTER);
optiPanel1.add(jPanel2, BorderLayout.NORTH);
optiPanel1.add(jPanel1, BorderLayout.WEST);

// --- 'Panel 1' :The Vertical Flow panel on the WEST panel of Tabbed Pane #1
jPanel1.setLayout(verticalFlowLayout2);
verticalFlowLayout2.setVgap(10);
verticalFlowLayout2.setHorizontalFill(false);
verticalFlowLayout2.setVerticalFill(false);
jPanel1.setMinimumSize(new Dimension(90, 100));
jPanel1.setPreferredSize(new Dimension(120, 300));
jLabel12.setPreferredSize(new Dimension(120, 17));
jLabel12.setHorizontalAlignment(SwingConstants.LEFT);
jLabel12.setText("OGI Units");
```

```
jPanel1.add(jLabel3, null);
jPanel1.add(jLabel4, null);
jPanel1.add(jLabel2, null);
jPanel1.add(jLabel1, null);
jPanel1.add(jLabel13, null);
jPanel1.add(jLabel16, null);
jPanel1.add(jLabel12, null);

jLabel1.setHorizontalAlignment(SwingConstants.LEFT);
jLabel1.setMinimumSize(new Dimension(120, 15));
jLabel1.setPreferredSize(new Dimension(120, 15));
jLabel1.setText("SD DBH - All Trees");
jLabel2.setMinimumSize(new Dimension(120, 15));
jLabel2.setHorizontalAlignment(SwingConstants.LEFT);
jLabel2.setPreferredSize(new Dimension(120, 15));
jLabel2.setText("TPA - Large trees");
jLabel4.setMinimumSize(new Dimension(120, 15));
jLabel4.setHorizontalAlignment(SwingConstants.LEFT);
jLabel4.setPreferredSize(new Dimension(120, 15));
jLabel4.setText("DBH - All Trees");
jLabel3.setMinimumSize(new Dimension(120, 15));
jLabel3.setHorizontalAlignment(SwingConstants.LEFT);
jLabel3.setPreferredSize(new Dimension(120, 15));
jLabel3.setText("TPA - All trees");
jTextField9.setPreferredSize(new Dimension(60, 21));
jTextField9.setText("20");
jTextField9.setHorizontalAlignment(SwingConstants.RIGHT);
jLabel14.setPreferredSize(new Dimension(80, 17));
jLabel15.setPreferredSize(new Dimension(140, 22));
jLabel16.setText("TPA - OGI Inflection");
jLabel16.setHorizontalAlignment(SwingConstants.LEFT);
jLabel16.setPreferredSize(new Dimension(120, 17));
jLabel13.setPreferredSize(new Dimension(60, 20));

// --- 'Panel 2' :The Flow panel on the NORTH panel of Tabbed Pane # 1
jPanel2.setPreferredSize(new Dimension(200, 30));
jPanel2.setLayout(flowLayout1);

jPanel2.add(jLabel5, null);
jPanel2.add(jLabel7, null);
jPanel2.add(jLabel6, null);

jLabel5.setBorder(BorderFactory.createEtchedBorder());
jLabel5.setMinimumSize(new Dimension(110, 17));
jLabel5.setPreferredSize(new Dimension(110, 20));
jLabel5.setHorizontalTextPosition(SwingConstants.CENTER);
jLabel5.setFont(new java.awt.Font("Dialog", 1, 12));
jLabel5.setHorizontalAlignment(SwingConstants.CENTER);
jLabel5.setText("OGI Variables");
jLabel6.setFont(new java.awt.Font("Dialog", 1, 12));
jLabel6.setBorder(BorderFactory.createEtchedBorder());
jLabel6.setPreferredSize(new Dimension(85, 21));
```

```
jLabel6.setHorizontalAlignment(SwingConstants.CENTER);
jLabel6.setHorizontalTextPosition(SwingConstants.CENTER);
jLabel6.setText("Old Forest");
jLabel7.setFont(new java.awt.Font("Dialog", 1, 12));
jLabel7.setBorder(BorderFactory.createEtchedBorder());
jLabel7.setPreferredSize(new Dimension(85, 21));
jLabel7.setHorizontalAlignment(SwingConstants.CENTER);
jLabel7.setHorizontalTextPosition(SwingConstants.CENTER);
jLabel7.setText("Young Forest");

// --- 'Panel 3' :The Flow Panel on the CENTER panel of Tabbed Pane # 1
jPanel3.setPreferredSize(new Dimension(140, 300));
jPanel3.setLayout(flowLayout2);
flowLayout2.setAlignment(FlowLayout.LEFT);
flowLayout2.setHgap(15);

jPanel3.add(jTextField1, null);
jPanel3.add(jLabel8, null);
jPanel3.add(jTextField2, null);
jPanel3.add(jTextField3, null);
jPanel3.add(jLabel9, null);
jPanel3.add(jTextField4, null);
jPanel3.add(jTextField5, null);
jPanel3.add(jLabel10, null);
jPanel3.add(jTextField6, null);
jPanel3.add(jTextField7, null);
jPanel3.add(jLabel11, null);
jPanel3.add(jTextField8, null);
jPanel3.add(jLabel15, null);
jPanel3.add(jTextField9, null);
jPanel3.add(jLabel14, null);
jPanel3.add(jRadioButton1, null);
jPanel3.add(jRadioButton2, null);
jTextField1.setPreferredSize(new Dimension(60, 21));
jTextField1.setText("935");
jTextField1.setHorizontalAlignment(SwingConstants.RIGHT);
jTextField2.setPreferredSize(new Dimension(60, 21));
jTextField2.setText("448");
jTextField2.setHorizontalAlignment(SwingConstants.RIGHT);
jPanel4.setPreferredSize(new Dimension(120, 300));
jTextField3.setPreferredSize(new Dimension(60, 21));
jTextField3.setText("21");
jTextField4.setPreferredSize(new Dimension(60, 21));
jTextField4.setText("31");
jLabel8.setPreferredSize(new Dimension(10, 17));
jLabel9.setPreferredSize(new Dimension(10, 17));
jTextField3.setHorizontalAlignment(SwingConstants.RIGHT);
jTextField4.setHorizontalAlignment(SwingConstants.RIGHT);
jTextField5.setHorizontalAlignment(SwingConstants.RIGHT);
jTextField5.setText("0.5");
jTextField5.setPreferredSize(new Dimension(60, 21));
jLabel10.setPreferredSize(new Dimension(10, 17));
```

```
jTextField6.setHorizontalAlignment(SwingConstants.RIGHT);
jTextField6.setText("19");
jTextField6.setPreferredSize(new Dimension(60, 21));
jLabel11.setPreferredSize(new Dimension(10, 17));
jTextField7.setHorizontalAlignment(SwingConstants.RIGHT);
jTextField7.setText("12");
jTextField7.setPreferredSize(new Dimension(60, 21));
jTextField8.setHorizontalAlignment(SwingConstants.RIGHT);
jTextField8.setText("32");
jTextField8.setPreferredSize(new Dimension(60, 21));
jRadioButton1.setText("Metric");
jRadioButton2.setText("English");
buttonGroup1.add(jRadioButton1);
buttonGroup1.add(jRadioButton2);
jRadioButton1.setSelected(true);

// --- ' Panel 5' : The SECOND Tabbed Pane 'OPTIMIZER Inputs' which contains 4 panels
jPanel5.setPreferredSize(new Dimension(50, 10));
jPanel5.setLayout(borderLayout3);
borderLayout3.setHgap(5);
borderLayout3.setVgap(5);
jPanel5.add(jPanel7, BorderLayout.EAST);
jPanel5.add(jPanel6, BorderLayout.CENTER);
jPanel5.add(jPanel8, BorderLayout.WEST);
jPanel8.add(jLabel43, null);


// --- ' Panel 6 ' : The EAST PANEL in 'Panel 5'
jPanel6.setPreferredSize(new Dimension(120, 300));
jPanel6.setLayout(flowLayout3);
flowLayout3.setHgap(9);

jPanel6.add(jLabel17, null);
jPanel6.add(jLabel18, null);
jPanel6.add(jLabel19, null);
jPanel6.add(jLabel20, null);
jPanel6.add(jTextField10, null);
jPanel6.add(jLabel21, null);
jPanel6.add(jTextField11, null);
jPanel6.add(jLabel22, null);
jPanel6.add(jTextField12, null);
jPanel6.add(jLabel23, null);
jPanel6.add(jTextField13, null);
jPanel6.add(jLabel24, null);
jPanel6.add(jTextField14, null);
jPanel6.add(jLabel25, null);
jPanel6.add(jTextField15, null);
jPanel6.add(jLabel26, null);
jPanel6.add(jTextField16, null);
jPanel6.add(jLabel27, null);
jPanel6.add(jTextField17, null);
jPanel6.add(jLabel28, null);
```

```
jPanel6.add(jTextField18, null);
jPanel6.add(jLabel29, null);
jPanel6.add(jTextField19, null);


jLabel17.setFont(new java.awt.Font("Dialog", 1, 12));
jLabel17.setHorizontalAlignment(SwingConstants.CENTER);
jLabel17.setText("Report Years");
jLabel17.setBorder(BorderFactory.createEtchedBorder());
jLabel17.setPreferredSize(new Dimension(110, 21));
jLabel18.setFont(new java.awt.Font("Dialog", 3, 12));
jLabel18.setHorizontalAlignment(SwingConstants.CENTER);
jLabel18.setText("Period");
jLabel18.setBorder(BorderFactory.createEtchedBorder());
jLabel18.setPreferredSize(new Dimension(50, 21));
jLabel19.setFont(new java.awt.Font("Dialog", 3, 12));
jLabel19.setHorizontalAlignment(SwingConstants.CENTER);
jLabel19.setText("Year");
jLabel19.setBorder(BorderFactory.createEtchedBorder());
jLabel19.setPreferredSize(new Dimension(50, 21));

jLabel20.setFont(new java.awt.Font("Dialog", 3, 12));
jLabel20.setHorizontalAlignment(SwingConstants.CENTER);
jLabel20.setText("0");
jLabel20.setBorder(BorderFactory.createLoweredBevelBorder());
jLabel20.setPreferredSize(new Dimension(50, 21));

jTextField10.setHorizontalAlignment(SwingConstants.RIGHT);
jTextField10.setText("2000");
jTextField10.setPreferredSize(new Dimension(50, 21));

jLabel21.setFont(new java.awt.Font("Dialog", 3, 12));
jLabel21.setHorizontalAlignment(SwingConstants.CENTER);
jLabel21.setText("1");
jLabel21.setBorder(BorderFactory.createLoweredBevelBorder());
jLabel21.setPreferredSize(new Dimension(50, 21));

jTextField11.setHorizontalAlignment(SwingConstants.RIGHT);
jTextField11.setText("2005");
jTextField11.setPreferredSize(new Dimension(50, 21));

jLabel22.setFont(new java.awt.Font("Dialog", 3, 12));
jLabel22.setHorizontalAlignment(SwingConstants.CENTER);
jLabel22.setText("2");
jLabel22.setBorder(BorderFactory.createLoweredBevelBorder());
jLabel22.setPreferredSize(new Dimension(50, 21));

jTextField12.setHorizontalAlignment(SwingConstants.RIGHT);
jTextField12.setText("2010");
jTextField12.setPreferredSize(new Dimension(50, 21));

jLabel23.setFont(new java.awt.Font("Dialog", 3, 12));
```

```
jLabel23.setHorizontalAlignment(SwingConstants.CENTER);
jLabel23.setText("3");
jLabel23.setBorder(BorderFactory.createLoweredBevelBorder());
jLabel23.setPreferredSize(new Dimension(50, 21));

jTextField13.setHorizontalAlignment(SwingConstants.RIGHT);
jTextField13.setText("2015");
jTextField13.setPreferredSize(new Dimension(50, 21));

jLabel24.setFont(new java.awt.Font("Dialog", 3, 12));
jLabel24.setHorizontalAlignment(SwingConstants.CENTER);
jLabel24.setText("4");
jLabel24.setBorder(BorderFactory.createLoweredBevelBorder());
jLabel24.setPreferredSize(new Dimension(50, 21));

jTextField14.setHorizontalAlignment(SwingConstants.RIGHT);
jTextField14.setText("2020");
jTextField14.setPreferredSize(new Dimension(50, 21));

jLabel25.setFont(new java.awt.Font("Dialog", 3, 12));
jLabel25.setHorizontalAlignment(SwingConstants.CENTER);
jLabel25.setText("5");
jLabel25.setBorder(BorderFactory.createLoweredBevelBorder());
jLabel25.setPreferredSize(new Dimension(50, 21));

jTextField15.setHorizontalAlignment(SwingConstants.RIGHT);
jTextField15.setText("2025");
jTextField15.setPreferredSize(new Dimension(50, 21));

jLabel26.setFont(new java.awt.Font("Dialog", 3, 12));
jLabel26.setHorizontalAlignment(SwingConstants.CENTER);
jLabel26.setText("6");
jLabel26.setBorder(BorderFactory.createLoweredBevelBorder());
jLabel26.setPreferredSize(new Dimension(50, 21));

jTextField16.setHorizontalAlignment(SwingConstants.RIGHT);
jTextField16.setText("2030");
jTextField16.setPreferredSize(new Dimension(50, 21));

jLabel27.setFont(new java.awt.Font("Dialog", 3, 12));
jLabel27.setHorizontalAlignment(SwingConstants.CENTER);
jLabel27.setText("7");
jLabel27.setBorder(BorderFactory.createLoweredBevelBorder());
jLabel27.setPreferredSize(new Dimension(50, 21));

jTextField17.setHorizontalAlignment(SwingConstants.RIGHT);
jTextField17.setText("2035");
jTextField17.setPreferredSize(new Dimension(50, 21));

jLabel28.setFont(new java.awt.Font("Dialog", 3, 12));
jLabel28.setHorizontalAlignment(SwingConstants.CENTER);
jLabel28.setText("8");
```

```
jLabel28.setBorder(BorderFactory.createLoweredBevelBorder());
jLabel28.setPreferredSize(new Dimension(50, 21));

jTextField18.setHorizontalAlignment(SwingConstants.RIGHT);
jTextField18.setText("2040");
jTextField18.setPreferredSize(new Dimension(50, 21));

jLabel29.setFont(new java.awt.Font("Dialog", 3, 12));
jLabel29.setHorizontalAlignment(SwingConstants.CENTER);
jLabel29.setText("9");
jLabel29.setBorder(BorderFactory.createLoweredBevelBorder());
jLabel29.setPreferredSize(new Dimension(50, 21));

jTextField19.setHorizontalAlignment(SwingConstants.RIGHT);
jTextField19.setText("2045");
jTextField19.setPreferredSize(new Dimension(50, 21));

// --- ' Panel # 7 ' The CENTER Panel in ' Panel # 5'
jPanel7.setPreferredSize(new Dimension(120, 300));
jPanel7.setLayout(flowLayout4);
flowLayout4.setHgap(9);

jPanel7.add(jLabel30, null);
jPanel7.add(jLabel31, null);
jPanel7.add(jLabel32, null);
jPanel7.add(jLabel33, null);
jPanel7.add(jTextField20, null);
jPanel7.add(jLabel34, null);
jPanel7.add(jTextField21, null);
jPanel7.add(jLabel35, null);
jPanel7.add(jTextField22, null);
jPanel7.add(jLabel36, null);
jPanel7.add(jTextField23, null);
jPanel7.add(jLabel37, null);
jPanel7.add(jTextField24, null);
jPanel7.add(jLabel38, null);
jPanel7.add(jTextField25, null);
jPanel7.add(jLabel39, null);
jPanel7.add(jTextField26, null);
jPanel7.add(jLabel40, null);
jPanel7.add(jTextField27, null);
jPanel7.add(jLabel41, null);
jPanel7.add(jTextField28, null);
jPanel7.add(jLabel42, null);
jPanel7.add(jTextField29, null);

jLabel30.setFont(new java.awt.Font("Dialog", 1, 12));
jLabel30.setHorizontalAlignment(SwingConstants.CENTER);
jLabel30.setText("Report Years");
jLabel30.setBorder(BorderFactory.createEtchedBorder());
jLabel30.setPreferredSize(new Dimension(110, 21));
jLabel31.setFont(new java.awt.Font("Dialog", 3, 12));
```

```
jLabel31.setHorizontalAlignment(SwingConstants.CENTER);
jLabel31.setText("Period");
jLabel31.setBorder(BorderFactory.createEtchedBorder());
jLabel31.setPreferredSize(new Dimension(50, 21));
jLabel32.setFont(new java.awt.Font("Dialog", 3, 12));
jLabel32.setHorizontalAlignment(SwingConstants.CENTER);
jLabel32.setText("Year");
jLabel32.setBorder(BorderFactory.createEtchedBorder());
jLabel32.setPreferredSize(new Dimension(50, 21));

jLabel33.setFont(new java.awt.Font("Dialog", 3, 12));
jLabel33.setHorizontalAlignment(SwingConstants.CENTER);
jLabel33.setText("10");
jLabel33.setBorder(BorderFactory.createLoweredBevelBorder());
jLabel33.setPreferredSize(new Dimension(50, 21));

jTextField20.setHorizontalAlignment(SwingConstants.RIGHT);
jTextField20.setText("2050");
jTextField20.setPreferredSize(new Dimension(50, 21));

jLabel34.setFont(new java.awt.Font("Dialog", 3, 12));
jLabel34.setHorizontalAlignment(SwingConstants.CENTER);
jLabel34.setText("11");
jLabel34.setBorder(BorderFactory.createLoweredBevelBorder());
jLabel34.setPreferredSize(new Dimension(50, 21));

jTextField21.setHorizontalAlignment(SwingConstants.RIGHT);
jTextField21.setText("2055");
jTextField21.setPreferredSize(new Dimension(50, 21));

jLabel35.setFont(new java.awt.Font("Dialog", 3, 12));
jLabel35.setHorizontalAlignment(SwingConstants.CENTER);
jLabel35.setText("12");
jLabel35.setBorder(BorderFactory.createLoweredBevelBorder());
jLabel35.setPreferredSize(new Dimension(50, 21));

jTextField22.setHorizontalAlignment(SwingConstants.RIGHT);
jTextField22.setText("2060");
jTextField22.setPreferredSize(new Dimension(50, 21));

jLabel36.setFont(new java.awt.Font("Dialog", 3, 12));
jLabel36.setHorizontalAlignment(SwingConstants.CENTER);
jLabel36.setText("13");
jLabel36.setBorder(BorderFactory.createLoweredBevelBorder());
jLabel36.setPreferredSize(new Dimension(50, 21));

jTextField23.setHorizontalAlignment(SwingConstants.RIGHT);
jTextField23.setText("2065");
jTextField23.setPreferredSize(new Dimension(50, 21));

jLabel37.setFont(new java.awt.Font("Dialog", 3, 12));
jLabel37.setHorizontalAlignment(SwingConstants.CENTER);
```

```
jLabel37.setText("14");
jLabel37.setBorder(BorderFactory.createLoweredBevelBorder());
jLabel37.setPreferredSize(new Dimension(50, 21));

jTextField24.setHorizontalAlignment(SwingConstants.RIGHT);
jTextField24.setText("2070");
jTextField24.setPreferredSize(new Dimension(50, 21));

jLabel38.setFont(new java.awt.Font("Dialog", 3, 12));
jLabel38.setHorizontalAlignment(SwingConstants.CENTER);
jLabel38.setText("15");
jLabel38.setBorder(BorderFactory.createLoweredBevelBorder());
jLabel38.setPreferredSize(new Dimension(50, 21));

jTextField25.setHorizontalAlignment(SwingConstants.RIGHT);
jTextField25.setText("2075");
jTextField25.setPreferredSize(new Dimension(50, 21));

jLabel39.setFont(new java.awt.Font("Dialog", 3, 12));
jLabel39.setHorizontalAlignment(SwingConstants.CENTER);
jLabel39.setText("16");
jLabel39.setBorder(BorderFactory.createLoweredBevelBorder());
jLabel39.setPreferredSize(new Dimension(50, 21));

jTextField26.setHorizontalAlignment(SwingConstants.RIGHT);
jTextField26.setText("2080");
jTextField26.setPreferredSize(new Dimension(50, 21));

jLabel40.setFont(new java.awt.Font("Dialog", 3, 12));
jLabel40.setHorizontalAlignment(SwingConstants.CENTER);
jLabel40.setText("17");
jLabel40.setBorder(BorderFactory.createLoweredBevelBorder());
jLabel40.setPreferredSize(new Dimension(50, 21));

jTextField27.setHorizontalAlignment(SwingConstants.RIGHT);
jTextField27.setText("2085");
jTextField27.setPreferredSize(new Dimension(50, 21));

jLabel41.setFont(new java.awt.Font("Dialog", 3, 12));
jLabel41.setHorizontalAlignment(SwingConstants.CENTER);
jLabel41.setText("18");
jLabel41.setBorder(BorderFactory.createLoweredBevelBorder());
jLabel41.setPreferredSize(new Dimension(50, 21));

jTextField28.setHorizontalAlignment(SwingConstants.RIGHT);
jTextField28.setText("2090");
jTextField28.setPreferredSize(new Dimension(50, 21));

jLabel42.setFont(new java.awt.Font("Dialog", 3, 12));
jLabel42.setHorizontalAlignment(SwingConstants.CENTER);
jLabel42.setText("19");
jLabel42.setBorder(BorderFactory.createLoweredBevelBorder());
```

```
jLabel42.setPreferredSize(new Dimension(50, 21));

jTextField29.setHorizontalAlignment(SwingConstants.RIGHT);
jTextField29.setText("2095");
jTextField29.setPreferredSize(new Dimension(50, 21));

// --- 'Panel # 8' : The WEST Panel of PANEL # 5
jPanel8.setPreferredSize(new Dimension(200, 10));
jPanel8.setLayout(flowLayout5);
flowLayout5.setHgap(9);

jPanel8.add(jTextField30, null);
jPanel8.add(jLabel44, null);
jPanel8.add(jTextField31, null);
jPanel8.add(jLabel45, null);
jPanel8.add(jTextField32, null);
jPanel8.add(jLabel46, null);
jPanel8.add(jTextField33, null);
jPanel8.add(jLabel47, null);
jPanel8.add(jTextField34, null);
jPanel8.add(jLabelbf14, null);
jPanel8.add(jTextFieldc34, null);
jPanel8.add(jLabel48, null);
jPanel8.add(jLabel49, null);
jPanel8.add(jTextField36, null);
jPanel8.add(jLabel410, null);
jPanel8.add(jComboBox1, null);



jLabel43.setHorizontalAlignment(SwingConstants.LEFT);
jLabel43.setText("Final Harvest Year");
jLabel43.setPreferredSize(new Dimension(130, 21));
jTextField30.setHorizontalAlignment(SwingConstants.RIGHT);
jTextField30.setText("2060");
jTextField30.setPreferredSize(new Dimension(50, 21));

jLabel44.setHorizontalAlignment(SwingConstants.LEFT);
jLabel44.setText("Evaluation Year");
jLabel44.setPreferredSize(new Dimension(130, 21));
jTextField31.setHorizontalAlignment(SwingConstants.RIGHT);
jTextField31.setText("2060");
jTextField31.setPreferredSize(new Dimension(50, 21));

jLabel45.setHorizontalAlignment(SwingConstants.LEFT);
jLabel45.setText("TPA Cutting Interval");
jLabel45.setPreferredSize(new Dimension(130, 21));
jTextField32.setHorizontalAlignment(SwingConstants.RIGHT);
jTextField32.setText("5");
jTextField32.setPreferredSize(new Dimension(50, 21));
```

```
jLabel46.setHorizontalAlignment(SwingConstants.LEFT);
jLabel46.setText("TPA Min. Constraint");
jLabel46.setPreferredSize(new Dimension(130, 21));
jTextField33.setHorizontalAlignment(SwingConstants.RIGHT);
jTextField33.setText("40");
jTextField33.setPreferredSize(new Dimension(50, 21));

jLabel47.setHorizontalAlignment(SwingConstants.LEFT);
jLabel47.setText("OGI Min. Constraint");
jLabel47.setPreferredSize(new Dimension(130, 21));
jTextField34.setHorizontalAlignment(SwingConstants.RIGHT);
jTextField34.setText("74");
jTextField34.setPreferredSize(new Dimension(50, 21));

jLabel48.setHorizontalAlignment(SwingConstants.LEFT);
jLabel48.setPreferredSize(new Dimension(180, 21));

jLabel49.setHorizontalAlignment(SwingConstants.LEFT);
jLabel49.setText("DB Name");
jLabel49.setPreferredSize(new Dimension(70, 21));
jTextField36.setHorizontalAlignment(SwingConstants.RIGHT);
jTextField36.setText(dbName);
jTextField36.setPreferredSize(new Dimension(110, 21));

// --- ' Panel 9' : The THIRD Tabbed Pane 'ECON Inputs' which contains 2 panels
jPanel9.setPreferredSize(new Dimension(50, 10));
jPanel9.setLayout(borderLayout4);
borderLayout4.setHgap(5);
borderLayout4.setVgap(5);
jPanel9.add(jPanel10, BorderLayout.EAST);
jPanel9.add(jPanel11, BorderLayout.WEST);
jPanel9.add(jPanel12, BorderLayout.CENTER);

// --- ' PANEL 10' : The EAST panel of Panel 9
jPanel10.setPreferredSize(new Dimension(175, 300));
jPanel10.setLayout(flowLayout6);
flowLayout6.setHgap(5);
flowLayout6.setAlignment(2);

jPanel10.add(jLabelspc, null);
jPanel10.add(jLabelsrt1, null);
jPanel10.add(jLabelsrt2, null);
jPanel10.add(jLabelsrt3, null);
jPanel10.add(jTextFieldp1, null);
jPanel10.add(jTextFieldp2, null);
jPanel10.add(jTextFieldp3, null);
jPanel10.add(jTextFieldp4, null);
jPanel10.add(jTextFieldp5, null);
jPanel10.add(jTextFieldp6, null);
jPanel10.add(jTextFieldp7, null);
jPanel10.add(jTextFieldp8, null);
jPanel10.add(jTextFieldp9, null);
```

```
jPanel10.add(jTextFieldp10, null);
jPanel10.add(jTextFieldp11, null);
jPanel10.add(jTextFieldp12, null);
jPanel10.add(jTextFieldp13, null);
jPanel10.add(jTextFieldp14, null);
jPanel10.add(jTextFieldp15, null);
jPanel10.add(jTextFieldp16, null);
jPanel10.add(jTextFieldp17, null);
jPanel10.add(jTextFieldp18, null);
jPanel10.add(jTextFieldp19, null);
jPanel10.add(jTextFieldp20, null);
jPanel10.add(jTextFieldp21, null);
jPanel10.add(jTextFieldp22, null);
jPanel10.add(jTextFieldp23, null);
jPanel10.add(jTextFieldp24, null);
jPanel10.add(jTextFieldp25, null);
jPanel10.add(jTextFieldp26, null);
jPanel10.add(jTextFieldp27, null);
jPanel10.add(jTextFieldp28, null);
jPanel10.add(jTextFieldp29, null);
jPanel10.add(jTextFieldp30, null);
jPanel10.add(jTextFieldp31, null);
jPanel10.add(jTextFieldp32, null);
jPanel10.add(jTextFieldp33, null);
jPanel10.add(jTextFieldp34, null);
jPanel10.add(jTextFieldp35, null);
jPanel10.add(jTextFieldp36, null);
jPanel10.add(jTextFieldp37, null);
jPanel10.add(jTextFieldp38, null);
jPanel10.add(jTextFieldp39, null);
jPanel10.add(jTextFieldp40, null);

jLabelspc.setFont(new java.awt.Font("Dialog", 1, 12));
jLabelspc.setHorizontalAlignment(SwingConstants.CENTER);
jLabelspc.setText("SPC");
jLabelspc.setBorder(BorderFactory.createEtchedBorder());
jLabelspc.setPreferredSize(new Dimension(36, 21));

jLabelsrt1.setFont(new java.awt.Font("Dialog", 1, 12));
jLabelsrt1.setHorizontalAlignment(SwingConstants.CENTER);
jLabelsrt1.setText("Srt 1");
jLabelsrt1.setBorder(BorderFactory.createEtchedBorder());
jLabelsrt1.setPreferredSize(new Dimension(36, 21));

jLabelsrt2.setFont(new java.awt.Font("Dialog", 1, 12));
jLabelsrt2.setHorizontalAlignment(SwingConstants.CENTER);
jLabelsrt2.setText("Srt 2");
jLabelsrt2.setBorder(BorderFactory.createEtchedBorder());
jLabelsrt2.setPreferredSize(new Dimension(36, 21));

jLabelsrt3.setFont(new java.awt.Font("Dialog", 1, 12));
jLabelsrt3.setHorizontalAlignment(SwingConstants.CENTER);
```

```
jLabelsrt3.setText("Srt 3");
jLabelsrt3.setBorder(BorderFactory.createEtchedBorder());
jLabelsrt3.setPreferredSize(new Dimension(36, 21));

jTextFieldp1.setHorizontalAlignment(SwingConstants.RIGHT);
jTextFieldp1.setText("DF");
jTextFieldp1.setPreferredSize(new Dimension(36, 21));

jTextFieldp2.setHorizontalAlignment(SwingConstants.RIGHT);
jTextFieldp2.setText("500");
jTextFieldp2.setPreferredSize(new Dimension(36, 21));

jTextFieldp3.setHorizontalAlignment(SwingConstants.RIGHT);
jTextFieldp3.setText("600");
jTextFieldp3.setPreferredSize(new Dimension(36, 21));

jTextFieldp4.setHorizontalAlignment(SwingConstants.RIGHT);
jTextFieldp4.setText("700");
jTextFieldp4.setPreferredSize(new Dimension(36, 21));

jTextFieldp5.setHorizontalAlignment(SwingConstants.RIGHT);
jTextFieldp5.setText("BM");
jTextFieldp5.setPreferredSize(new Dimension(36, 21));

jTextFieldp6.setHorizontalAlignment(SwingConstants.RIGHT);
jTextFieldp6.setText("400");
jTextFieldp6.setPreferredSize(new Dimension(36, 21));

jTextFieldp7.setHorizontalAlignment(SwingConstants.RIGHT);
jTextFieldp7.setText("450");
jTextFieldp7.setPreferredSize(new Dimension(36, 21));

jTextFieldp8.setHorizontalAlignment(SwingConstants.RIGHT);
jTextFieldp8.setText("500");
jTextFieldp8.setPreferredSize(new Dimension(36, 21));

jTextFieldp9.setHorizontalAlignment(SwingConstants.RIGHT);
jTextFieldp9.setText("WO");
jTextFieldp9.setPreferredSize(new Dimension(36, 21));

jTextFieldp10.setHorizontalAlignment(SwingConstants.RIGHT);
jTextFieldp10.setText("400");
jTextFieldp10.setPreferredSize(new Dimension(36, 21));

jTextFieldp11.setHorizontalAlignment(SwingConstants.RIGHT);
jTextFieldp11.setText("450");
jTextFieldp11.setPreferredSize(new Dimension(36, 21));

jTextFieldp12.setHorizontalAlignment(SwingConstants.RIGHT);
jTextFieldp12.setText("500");
jTextFieldp12.setPreferredSize(new Dimension(36, 21));
```

```
jTextFieldp13.setHorizontalAlignment(SwingConstants.RIGHT);
jTextFieldp13.setText("GF");
jTextFieldp13.setPreferredSize(new Dimension(36, 21));

jTextFieldp14.setHorizontalAlignment(SwingConstants.RIGHT);
jTextFieldp14.setText("400");
jTextFieldp14.setPreferredSize(new Dimension(36, 21));

jTextFieldp15.setHorizontalAlignment(SwingConstants.RIGHT);
jTextFieldp15.setText("500");
jTextFieldp15.setPreferredSize(new Dimension(36, 21));

jTextFieldp16.setHorizontalAlignment(SwingConstants.RIGHT);
jTextFieldp16.setText("550");
jTextFieldp16.setPreferredSize(new Dimension(36, 21));

jTextFieldp17.setHorizontalAlignment(SwingConstants.RIGHT);
jTextFieldp17.setText("WF");
jTextFieldp17.setPreferredSize(new Dimension(36, 21));

jTextFieldp18.setHorizontalAlignment(SwingConstants.RIGHT);
jTextFieldp18.setText("400");
jTextFieldp18.setPreferredSize(new Dimension(36, 21));

jTextFieldp19.setHorizontalAlignment(SwingConstants.RIGHT);
jTextFieldp19.setText("500");
jTextFieldp19.setPreferredSize(new Dimension(36, 21));

jTextFieldp20.setHorizontalAlignment(SwingConstants.RIGHT);
jTextFieldp20.setText("550");
jTextFieldp20.setPreferredSize(new Dimension(36, 21));

jTextFieldp21.setHorizontalAlignment(SwingConstants.RIGHT);
jTextFieldp21.setText("RA");
jTextFieldp21.setPreferredSize(new Dimension(36, 21));

jTextFieldp22.setHorizontalAlignment(SwingConstants.RIGHT);
jTextFieldp22.setText("500");
jTextFieldp22.setPreferredSize(new Dimension(36, 21));

jTextFieldp23.setHorizontalAlignment(SwingConstants.RIGHT);
jTextFieldp23.setText("575");
jTextFieldp23.setPreferredSize(new Dimension(36, 21));

jTextFieldp24.setHorizontalAlignment(SwingConstants.RIGHT);
jTextFieldp24.setText("650");
jTextFieldp24.setPreferredSize(new Dimension(36, 21));

jTextFieldp25.setHorizontalAlignment(SwingConstants.RIGHT);
jTextFieldp25.setText("WH");
jTextFieldp25.setPreferredSize(new Dimension(36, 21));
```

```
jTextFieldp26.setHorizontalAlignment(SwingConstants.RIGHT);
jTextFieldp26.setText("400");
jTextFieldp26.setPreferredSize(new Dimension(36, 21));

jTextFieldp27.setHorizontalAlignment(SwingConstants.RIGHT);
jTextFieldp27.setText("500");
jTextFieldp27.setPreferredSize(new Dimension(36, 21));

jTextFieldp28.setHorizontalAlignment(SwingConstants.RIGHT);
jTextFieldp28.setText("550");
jTextFieldp28.setPreferredSize(new Dimension(36, 21));

jTextFieldp29.setHorizontalAlignment(SwingConstants.RIGHT);
jTextFieldp29.setText("XX");
jTextFieldp29.setPreferredSize(new Dimension(36, 21));

jTextFieldp30.setHorizontalAlignment(SwingConstants.RIGHT);
jTextFieldp30.setText("500");
jTextFieldp30.setPreferredSize(new Dimension(36, 21));

jTextFieldp31.setHorizontalAlignment(SwingConstants.RIGHT);
jTextFieldp31.setText("600");
jTextFieldp31.setPreferredSize(new Dimension(36, 21));

jTextFieldp32.setHorizontalAlignment(SwingConstants.RIGHT);
jTextFieldp32.setText("700");
jTextFieldp32.setPreferredSize(new Dimension(36, 21));

jTextFieldp33.setHorizontalAlignment(SwingConstants.RIGHT);
jTextFieldp33.setText("XX");
jTextFieldp33.setPreferredSize(new Dimension(36, 21));

jTextFieldp34.setHorizontalAlignment(SwingConstants.RIGHT);
jTextFieldp34.setText("500");
jTextFieldp34.setPreferredSize(new Dimension(36, 21));

jTextFieldp35.setHorizontalAlignment(SwingConstants.RIGHT);
jTextFieldp35.setText("600");
jTextFieldp35.setPreferredSize(new Dimension(36, 21));

jTextFieldp36.setHorizontalAlignment(SwingConstants.RIGHT);
jTextFieldp36.setText("700");
jTextFieldp36.setPreferredSize(new Dimension(36, 21));

jTextFieldp37.setHorizontalAlignment(SwingConstants.RIGHT);
jTextFieldp37.setText("XX");
jTextFieldp37.setPreferredSize(new Dimension(36, 21));

jTextFieldp38.setHorizontalAlignment(SwingConstants.RIGHT);
jTextFieldp38.setText("500");
jTextFieldp38.setPreferredSize(new Dimension(36, 21));
```

```
jTextFieldp39.setHorizontalAlignment(SwingConstants.RIGHT);
jTextFieldp39.setText("600");
jTextFieldp39.setPreferredSize(new Dimension(36, 21));

jTextFieldp40.setHorizontalAlignment(SwingConstants.RIGHT);
jTextFieldp40.setText("700");
jTextFieldp40.setPreferredSize(new Dimension(36, 21));

// --- ' PANEL 11' : The WEST panel of Panel 9
jPanel11.setPreferredSize(new Dimension(300, 300));
jPanel11.setLayout(flowLayout7);
flowLayout7.setHgap(5);
flowLayout7.setAlignment(0);

jPanel11.add(jLabeltpa, null);
jPanel11.add(jLabelbf0, null);
jPanel11.add(jLabelbf1, null);
jPanel11.add(jLabelbf2, null);
jPanel11.add(jLabelbf3, null);
jPanel11.add(jLabelbf4, null);
jPanel11.add(jLabelbf5, null);
jPanel11.add(jLabelbf6, null);
jPanel11.add(jTextFieldc1, null);
jPanel11.add(jTextFieldc2, null);
jPanel11.add(jTextFieldc3, null);
jPanel11.add(jTextFieldc4, null);
jPanel11.add(jTextFieldc5, null);
jPanel11.add(jTextFieldc6, null);
jPanel11.add(jLabelbf7, null);
jPanel11.add(jTextFieldc7, null);
jPanel11.add(jTextFieldc8, null);
jPanel11.add(jTextFieldc9, null);
jPanel11.add(jTextFieldc10, null);
jPanel11.add(jTextFieldc11, null);
jPanel11.add(jTextFieldc12, null);
jPanel11.add(jLabelbf8, null);
jPanel11.add(jTextFieldc13, null);
jPanel11.add(jTextFieldc14, null);
jPanel11.add(jTextFieldc15, null);
jPanel11.add(jTextFieldc16, null);
jPanel11.add(jTextFieldc17, null);
jPanel11.add(jTextFieldc18, null);
jPanel11.add(jLabelbf9, null);
jPanel11.add(jTextFieldc19, null);
jPanel11.add(jTextFieldc20, null);
jPanel11.add(jTextFieldc21, null);
jPanel11.add(jTextFieldc22, null);
jPanel11.add(jTextFieldc23, null);
jPanel11.add(jTextFieldc24, null);
jPanel11.add(jLabelbf10, null);
jPanel11.add(jTextFieldc25, null);
jPanel11.add(jTextFieldc26, null);
```

```
jPanel11.add(jTextFieldc27, null);
jPanel11.add(jTextFieldc28, null);
jPanel11.add(jTextFieldc29, null);
jPanel11.add(jTextFieldc30, null);
jPanel11.add(jLabelbf11, null);
jPanel11.add(jTextFieldc31, null);
jPanel11.add(jLabelbf12, null);
jPanel11.add(jTextFieldc32, null);
jPanel11.add(jLabelbf13, null);
jPanel11.add(jTextFieldc33, null);
jPanel11.add(jLabelbf15, null);
jPanel11.add(jTextFieldc35, null);
jPanel11.add(jLabelbf20, null);
jPanel11.add(jTextFieldc50, null);
jPanel11.add(jRadioButton3, null);
jPanel11.add(jLabelbf21, null);
jPanel11.add(jTextFieldc51, null);
jPanel11.add(jRadioButton4, null);
buttonGroup2.add(jRadioButton3);
buttonGroup2.add(jRadioButton4);
jRadioButton3.setSelected(true);



jLabeltpa.setFont(new java.awt.Font("Dialog", 1, 12));
jLabeltpa.setHorizontalAlignment(SwingConstants.CENTER);
jLabeltpa.setText("TPA");
jLabeltpa.setBorder(BorderFactory.createEtchedBorder());
jLabeltpa.setPreferredSize(new Dimension(35, 21));

jLabelbf0.setFont(new java.awt.Font("Dialog", 1, 12));
jLabelbf0.setHorizontalAlignment(SwingConstants.CENTER);
jLabelbf0.setText("0 M");
jLabelbf0.setBorder(BorderFactory.createEtchedBorder());
jLabelbf0.setPreferredSize(new Dimension(35, 21));

jLabelbf1.setFont(new java.awt.Font("Dialog", 1, 12));
jLabelbf1.setHorizontalAlignment(SwingConstants.CENTER);
jLabelbf1.setText("2 M");
jLabelbf1.setBorder(BorderFactory.createEtchedBorder());
jLabelbf1.setPreferredSize(new Dimension(35, 21));

jLabelbf2.setFont(new java.awt.Font("Dialog", 1, 12));
jLabelbf2.setHorizontalAlignment(SwingConstants.CENTER);
jLabelbf2.setText("5 M");
jLabelbf2.setBorder(BorderFactory.createEtchedBorder());
jLabelbf2.setPreferredSize(new Dimension(35, 21));

jLabelbf3.setFont(new java.awt.Font("Dialog", 1, 12));
jLabelbf3.setHorizontalAlignment(SwingConstants.CENTER);
```

```
jLabelbf3.setText("10M");
jLabelbf3.setBorder(BorderFactory.createEtchedBorder());
jLabelbf3.setPreferredSize(new Dimension(35, 21));

jLabelbf4.setFont(new java.awt.Font("Dialog", 1, 12));
jLabelbf4.setHorizontalAlignment(SwingConstants.CENTER);
jLabelbf4.setText("25M");
jLabelbf4.setBorder(BorderFactory.createEtchedBorder());
jLabelbf4.setPreferredSize(new Dimension(35, 21));

jLabelbf5.setFont(new java.awt.Font("Dialog", 1, 12));
jLabelbf5.setHorizontalAlignment(SwingConstants.CENTER);
jLabelbf5.setText("50M");
jLabelbf5.setBorder(BorderFactory.createEtchedBorder());
jLabelbf5.setPreferredSize(new Dimension(35, 21));

jLabelbf6.setFont(new java.awt.Font("Dialog", 1, 12));
jLabelbf6.setHorizontalAlignment(SwingConstants.CENTER);
jLabelbf6.setText("10");
jLabelbf6.setBorder(BorderFactory.createEtchedBorder());
jLabelbf6.setPreferredSize(new Dimension(35, 21));

jTextFieldc1.setHorizontalAlignment(SwingConstants.RIGHT);
jTextFieldc1.setText("80");
jTextFieldc1.setPreferredSize(new Dimension(35, 21));

jTextFieldc2.setHorizontalAlignment(SwingConstants.RIGHT);
jTextFieldc2.setText("120");
jTextFieldc2.setPreferredSize(new Dimension(35, 21));

jTextFieldc3.setHorizontalAlignment(SwingConstants.RIGHT);
jTextFieldc3.setText("100");
jTextFieldc3.setPreferredSize(new Dimension(35, 21));

jTextFieldc4.setHorizontalAlignment(SwingConstants.RIGHT);
jTextFieldc4.setText("80");
jTextFieldc4.setPreferredSize(new Dimension(35, 21));

jTextFieldc5.setHorizontalAlignment(SwingConstants.RIGHT);
jTextFieldc5.setText("50");
jTextFieldc5.setPreferredSize(new Dimension(35, 21));

jTextFieldc6.setHorizontalAlignment(SwingConstants.RIGHT);
jTextFieldc6.setText("40");
jTextFieldc6.setPreferredSize(new Dimension(35, 21));

jLabelbf7.setFont(new java.awt.Font("Dialog", 1, 12));
jLabelbf7.setHorizontalAlignment(SwingConstants.CENTER);
jLabelbf7.setText("50");
jLabelbf7.setBorder(BorderFactory.createEtchedBorder());
jLabelbf7.setPreferredSize(new Dimension(35, 21));
```

```
jTextFieldc7.setHorizontalAlignment(SwingConstants.RIGHT);
jTextFieldc7.setText("100");
jTextFieldc7.setPreferredSize(new Dimension(35, 21));

jTextFieldc8.setHorizontalAlignment(SwingConstants.RIGHT);
jTextFieldc8.setText("160");
jTextFieldc8.setPreferredSize(new Dimension(35, 21));

jTextFieldc9.setHorizontalAlignment(SwingConstants.RIGHT);
jTextFieldc9.setText("140");
jTextFieldc9.setPreferredSize(new Dimension(35, 21));

jTextFieldc10.setHorizontalAlignment(SwingConstants.RIGHT);
jTextFieldc10.setText("110");
jTextFieldc10.setPreferredSize(new Dimension(35, 21));

jTextFieldc11.setHorizontalAlignment(SwingConstants.RIGHT);
jTextFieldc11.setText("60");
jTextFieldc11.setPreferredSize(new Dimension(35, 21));

jTextFieldc12.setHorizontalAlignment(SwingConstants.RIGHT);
jTextFieldc12.setText("50");
jTextFieldc12.setPreferredSize(new Dimension(35, 21));

jLabelbf8.setFont(new java.awt.Font("Dialog", 1, 12));
jLabelbf8.setHorizontalAlignment(SwingConstants.CENTER);
jLabelbf8.setText("100");
jLabelbf8.setBorder(BorderFactory.createEtchedBorder());
jLabelbf8.setPreferredSize(new Dimension(35, 21));

jTextFieldc13.setHorizontalAlignment(SwingConstants.RIGHT);
jTextFieldc13.setText("120");
jTextFieldc13.setPreferredSize(new Dimension(35, 21));

jTextFieldc14.setHorizontalAlignment(SwingConstants.RIGHT);
jTextFieldc14.setText("200");
jTextFieldc14.setPreferredSize(new Dimension(35, 21));

jTextFieldc15.setHorizontalAlignment(SwingConstants.RIGHT);
jTextFieldc15.setText("180");
jTextFieldc15.setPreferredSize(new Dimension(35, 21));

jTextFieldc16.setHorizontalAlignment(SwingConstants.RIGHT);
jTextFieldc16.setText("140");
jTextFieldc16.setPreferredSize(new Dimension(35, 21));

jTextFieldc17.setHorizontalAlignment(SwingConstants.RIGHT);
jTextFieldc17.setText("80");
jTextFieldc17.setPreferredSize(new Dimension(35, 21));

jTextFieldc18.setHorizontalAlignment(SwingConstants.RIGHT);
jTextFieldc18.setText("60");
```

```
jTextFieldc18.setPreferredSize(new Dimension(35, 21));

jLabelbf9.setFont(new java.awt.Font("Dialog", 1, 12));
jLabelbf9.setHorizontalAlignment(SwingConstants.CENTER);
jLabelbf9.setText("200");
jLabelbf9.setBorder(BorderFactory.createEtchedBorder());
jLabelbf9.setPreferredSize(new Dimension(35, 21));

jTextFieldc19.setHorizontalAlignment(SwingConstants.RIGHT);
jTextFieldc19.setText("150");
jTextFieldc19.setPreferredSize(new Dimension(35, 21));

jTextFieldc20.setHorizontalAlignment(SwingConstants.RIGHT);
jTextFieldc20.setText("240");
jTextFieldc20.setPreferredSize(new Dimension(35, 21));

jTextFieldc21.setHorizontalAlignment(SwingConstants.RIGHT);
jTextFieldc21.setText("220");
jTextFieldc21.setPreferredSize(new Dimension(35, 21));

jTextFieldc22.setHorizontalAlignment(SwingConstants.RIGHT);
jTextFieldc22.setText("170");
jTextFieldc22.setPreferredSize(new Dimension(35, 21));

jTextFieldc23.setHorizontalAlignment(SwingConstants.RIGHT);
jTextFieldc23.setText("100");
jTextFieldc23.setPreferredSize(new Dimension(35, 21));

jTextFieldc24.setHorizontalAlignment(SwingConstants.RIGHT);
jTextFieldc24.setText("70");
jTextFieldc24.setPreferredSize(new Dimension(35, 21));

jLabelbf10.setFont(new java.awt.Font("Dialog", 1, 12));
jLabelbf10.setHorizontalAlignment(SwingConstants.CENTER);
jLabelbf10.setText("300");
jLabelbf10.setBorder(BorderFactory.createEtchedBorder());
jLabelbf10.setPreferredSize(new Dimension(35, 21));

jTextFieldc25.setHorizontalAlignment(SwingConstants.RIGHT);
jTextFieldc25.setText("160");
jTextFieldc25.setPreferredSize(new Dimension(35, 21));

jTextFieldc26.setHorizontalAlignment(SwingConstants.RIGHT);
jTextFieldc26.setText("280");
jTextFieldc26.setPreferredSize(new Dimension(35, 21));

jTextFieldc27.setHorizontalAlignment(SwingConstants.RIGHT);
jTextFieldc27.setText("260");
jTextFieldc27.setPreferredSize(new Dimension(35, 21));

jTextFieldc28.setHorizontalAlignment(SwingConstants.RIGHT);
jTextFieldc28.setText("200");
```

```
jTextFieldc28.setPreferredSize(new Dimension(35, 21));

jTextFieldc29.setHorizontalAlignment(SwingConstants.RIGHT);
jTextFieldc29.setText("120");
jTextFieldc29.setPreferredSize(new Dimension(35, 21));

jTextFieldc30.setHorizontalAlignment(SwingConstants.RIGHT);
jTextFieldc30.setText("80");
jTextFieldc30.setPreferredSize(new Dimension(35, 21));

jLabelbf11.setFont(new java.awt.Font("Dialog", 0, 12));
jLabelbf11.setHorizontalAlignment(SwingConstants.LEFT);
jLabelbf11.setText("Log Hauling Cost");
jLabelbf11.setPreferredSize(new Dimension(150, 21));

jTextFieldc31.setHorizontalAlignment(SwingConstants.RIGHT);
jTextFieldc31.setText("35");
jTextFieldc31.setPreferredSize(new Dimension(50, 21));

jLabelbf12.setFont(new java.awt.Font("Dialog", 0, 12));
jLabelbf12.setHorizontalAlignment(SwingConstants.LEFT);
jLabelbf12.setText("Annual Fixed Cost");
jLabelbf12.setPreferredSize(new Dimension(150, 21));

jTextFieldc32.setHorizontalAlignment(SwingConstants.RIGHT);
jTextFieldc32.setText("2");
jTextFieldc32.setPreferredSize(new Dimension(50, 21));

jLabelbf13.setFont(new java.awt.Font("Dialog", 0, 12));
jLabelbf13.setHorizontalAlignment(SwingConstants.LEFT);
jLabelbf13.setText("Minimum BDFT Sale");
jLabelbf13.setPreferredSize(new Dimension(150, 21));

jTextFieldc33.setHorizontalAlignment(SwingConstants.RIGHT);
jTextFieldc33.setText("2000");
jTextFieldc33.setPreferredSize(new Dimension(50, 21));



jLabelbf15.setFont(new java.awt.Font("Dialog", 0, 12));
jLabelbf15.setHorizontalAlignment(SwingConstants.LEFT);
jLabelbf15.setText("Real Discount Rate");
jLabelbf15.setPreferredSize(new Dimension(150, 21));

jTextFieldc35.setHorizontalAlignment(SwingConstants.RIGHT);
jTextFieldc35.setText("0.065");
jTextFieldc35.setPreferredSize(new Dimension(50, 21));

jLabelbf20.setFont(new java.awt.Font("Dialog", 0, 12));
jLabelbf20.setHorizontalAlignment(SwingConstants.LEFT);
jLabelbf20.setText("Period 0 Costs");
jLabelbf20.setPreferredSize(new Dimension(150, 21));
```

```
        jTextFieldc50.setHorizontalAlignment(SwingConstants.RIGHT);
        jTextFieldc50.setText("200");
        jTextFieldc50.setPreferredSize(new Dimension(50, 21));

        jLabelbf21.setFont(new java.awt.Font("Dialog", 0, 12));
        jLabelbf21.setHorizontalAlignment(SwingConstants.LEFT);
        jLabelbf21.setText("Bare Land Resid. Value");
        jLabelbf21.setPreferredSize(new Dimension(150, 21));

        jTextFieldc51.setHorizontalAlignment(SwingConstants.RIGHT);
        jTextFieldc51.setText("0");
        jTextFieldc51.setPreferredSize(new Dimension(50, 21));


}

// --- Close the GUI Button
void jMenuItemClose_actionPerformed(ActionEvent e)
{
 System.exit(0);
}

// --- Select the DB Picker and send to textBox
void jMenuItem1_actionPerformed(ActionEvent e)
{
 JFileChooser d = new JFileChooser ();
 d.setCurrentDirectory(new File("."));
 int result = d.showOpenDialog(this);
 if (result == 0)
 {
  dbName = d.getSelectedFile().getName();
  jTextField36.setText(dbName);

  // --- Query the DB and get Stand Ids from ADMIN Table
  optiButton1.doClick();
  int waitin = Inputs.setInputs(); // passes the DB name across to text file
  Database.load_database();
  try
  {
   Database.con_go();
   PreparedStatement ps1 = Database.con1.prepareStatement(
    "SELECT COUNT(STD_ID) AS amount FROM ADMIN");
   PreparedStatement ps2 = Database.con1.prepareStatement(
    "SELECT STD_ID FROM ADMIN");
   // --- Get number unique stands
   ResultSet rs = ps1.executeQuery();
   rs.next();
   int numStands = rs.getInt("amount");
   rs.close();
   ps1.close();
   // --- Get the stand numbers
```

```
    stands = new String [numStands];
    jComboBox1.removeAllItems(); // Gets rid of old numbers
    int counter = 0;
    rs = ps2.executeQuery();
    while (rs.next())
    {
     int temp = rs.getInt("STD_ID");
     stands [counter] = Integer.toString(temp);
     //System.out.println("stand " + stands[counter]);
     // --- update list of stands on GUI
     jComboBox1.addItem(stands [counter]);
     counter ++;
    }
    rs.close();
    ps2.close();
    Database.con_no();
    // --- update list of stands on GUI
    jComboBox1.addItem(stands [counter]);

   }
   catch (Exception k)
   {
   }
  }

}//FINISH: FileChooser Action

// --- 'SetInputs' Button
void optiButton1_actionPerformed(ActionEvent e)
{
 PrintWriter outputStream = null;
 try
 {
  outputStream = new PrintWriter( new FileOutputStream("Opti_in.txt"));
 }
 catch(FileNotFoundException fnf)
 {
  System.out.println("*** ERROR *** : Output File Not Found");
  System.exit(0);
 }
 outputStream.println("UNITS " + buttonGroup1.getSelection().getActionCommand());
 outputStream.println("RATE "  + jTextFieldc35.getText());
 outputStream.println("BASE "  + jTextFieldc34.getText());
 outputStream.println("YR_ALL "  + jTextField10.getText() + " " + jTextField11.getText() + " " +
  jTextField12.getText() + " " + jTextField13.getText() + " " + jTextField14.getText() + " " +
  jTextField15.getText() + " " + jTextField16.getText() + " " + jTextField17.getText() + " " +
  jTextField18.getText() + " " + jTextField19.getText() + " " + jTextField20.getText() + " " +
  jTextField21.getText() + " " + jTextField22.getText() + " " + jTextField23.getText() + " " +
  jTextField24.getText() + " " + jTextField25.getText() + " " + jTextField26.getText() + " " +
  jTextField27.getText() + " " + jTextField28.getText() + " " + jTextField29.getText() + " ");
 outputStream.println("PRICE_SPC " + jTextFieldp1.getText() + " " + jTextFieldp5.getText() +
```

```
    " " + jTextFieldp9.getText() + " " + jTextFieldp13.getText() + " " + jTextFieldp17.getText() + "
" +
    jTextFieldp21.getText() + " " + jTextFieldp25.getText() + " " + jTextFieldp29.getText() + " " +
    jTextFieldp33.getText() + " " + jTextFieldp37.getText());
   outputStream.println("PRICE_SORTS1 " + jTextFieldp2.getText() + " " + jTextFieldp6.getText()
+
    " " + jTextFieldp10.getText() + " " + jTextFieldp14.getText() + " " + jTextFieldp18.getText() +
" " +
    jTextFieldp22.getText() + " " + jTextFieldp26.getText() + " " + jTextFieldp30.getText() + " " +
    jTextFieldp34.getText() + " " + jTextFieldp38.getText());
   outputStream.println("PRICE_SORTS2 " + jTextFieldp3.getText() + " " + jTextFieldp7.getText()
+
    " " + jTextFieldp11.getText() + " " + jTextFieldp15.getText() + " " + jTextFieldp19.getText() +
" " +
    jTextFieldp23.getText() + " " + jTextFieldp27.getText() + " " + jTextFieldp31.getText() + " " +
    jTextFieldp35.getText() + " " + jTextFieldp39.getText());
   outputStream.println("PRICE_SORTS3 " + jTextFieldp4.getText() + " " + jTextFieldp8.getText()
+
    " " + jTextFieldp12.getText() + " " + jTextFieldp16.getText() + " " + jTextFieldp20.getText() +
" " +
    jTextFieldp24.getText() + " " + jTextFieldp28.getText() + " " + jTextFieldp32.getText() + " " +
    jTextFieldp36.getText() + " " + jTextFieldp40.getText());
   outputStream.println("MIN_OGI " + jTextField34.getText());
   outputStream.println("ANNUAL_COST " + jTextFieldc32.getText());
   outputStream.println("HAUL_COST " + jTextFieldc31.getText());
   outputStream.println("DEP_COST " + jTextFieldc2.getText() + " " + jTextFieldc3.getText() +
    " " + jTextFieldc4.getText() + " " + jTextFieldc5.getText() + " " + jTextFieldc6.getText() + " " +
    jTextFieldc8.getText() + " " + jTextFieldc9.getText() + " " + jTextFieldc10.getText() + " " +
    jTextFieldc11.getText() + " " + jTextFieldc12.getText() + " " + jTextFieldc14.getText() +
    " " + jTextFieldc15.getText() + " " + jTextFieldc16.getText() + " " + jTextFieldc17.getText() + "
" +
    jTextFieldc18.getText() + " " + jTextFieldc20.getText() + " " + jTextFieldc21.getText() + " " +
    jTextFieldc22.getText() + " " + jTextFieldc23.getText() + " " + jTextFieldc24.getText() +
    " " + jTextFieldc26.getText() + " " + jTextFieldc27.getText() + " " + jTextFieldc28.getText() + "
" +
    jTextFieldc29.getText() + " " + jTextFieldc30.getText());
   outputStream.println("PCT_DEP_COST " + jTextFieldc1.getText() + " " +
jTextFieldc7.getText() +
    " " + jTextFieldc13.getText() + " " + jTextFieldc19.getText() + " " + jTextFieldc25.getText());
   outputStream.println("BDFT_MIN " + jTextFieldc33.getText());
   outputStream.println("SD_DBH_YOUNG " + jTextField7.getText());
   outputStream.println("SD_DBH_OLD " + jTextField8.getText());
   outputStream.println("TPA_LG_YOUNG " + jTextField5.getText());
   outputStream.println("TPA_LG_OLD " + jTextField6.getText());
   outputStream.println("DBH_YOUNG " + jTextField3.getText());
   outputStream.println("DBH_OLD " + jTextField4.getText());
   outputStream.println("TPA_ALL_YOUNG " + jTextField1.getText());
   outputStream.println("TPA_ALL_OLD " + jTextField2.getText());
   outputStream.println("TPA_MIN " + jTextField9.getText());
   outputStream.println("CUT_INTERVAL " + jTextField32.getText());
   outputStream.println("CUT_MIN " + jTextField33.getText());
   outputStream.println("LOOK_AHEAD " + jTextField31.getText());
```

```
    if (theStand == null) theStand = "0";
    outputStream.println("STAND_ID "  + theStand);
    outputStream.println("FINAL_YEAR "  + jTextField30.getText());
    outputStream.println("DB_NAME " + dbName);
    outputStream.println("ECON_METHOD " +
  buttonGroup2.getSelection().getActionCommand());
    outputStream.println("PD_ZERO_COST "  + jTextFieldc50.getText());
    outputStream.println("RESID_VALUE "  + jTextFieldc51.getText());
    outputStream.println("OGI_INFLECTION "  + jTextField9.getText());


    outputStream.close();
   }

   // --- The 'View Results' Button Method
   void optiButton3_actionPerformed(ActionEvent e)
   {
    Runtime r = Runtime.getRuntime();
    Process p = null;

    try
    {
     p = r.exec("notepad");
    }
    catch (Exception t)
    {
     System.out.println("ERROR opening NOTEPAD");
    }

   }

   // The 'PROCESS' Button event
   void optiButton2_actionPerformed(ActionEvent e)
   {
    // Write Input File
    optiButton1.doClick();

    // Set Global Variables
    Inputs.setInputs();

    this.hide();

    Control masterControl = new Control();

    this.setVisible(true);

   }

   // The Stand ID combo box event
   void jComboBox1_actionPerformed(ActionEvent e)
   {
    JComboBox source = (JComboBox)e.getSource();
```

```
    if (source.getItemCount() < 1)
    {
     theStand = "0";
     System.out.println("No stands available");
     return;
    }
    else
    {
    theStand = (String)source.getSelectedItem();
    }

    System.out.println("The button says stand is: " + theStand);
   }

}
```

```
package thesis;

/*****************************************************************************
 * 'CONTROL' Class
 *
 * FUNCTION: This class serves as the engine for driving the algorithm
 * utilizing several methods and calling upon the requisite classes.
 *
 *****************************************************************************/

import java.util.*;
import java.text.NumberFormat;
import java.io.*;

public class Control
{
 // --- Declare Static Global Variables
 private static int [] years; // Report years, up to 20
 private static int standID; // FPS Stand ID number
 private static int end_year; // The Year to C.C. and end the model

 // --- Declare Tracking variables
 private int [][] rxLog; // FPS Silvics prescription by year
 private boolean [] harvestLog; // harvest occurence?

 // --- Create a PrintWriter object for building output text file
 PrintWriter outputStream = null;

 // --- Declare Silvics Class object
 Silvics silvicsMaster;


//=============================================================================
=====
 // 'CONTROL' Constructor
 //
 // FUNCTION: Instantiate objects and global variables, start processing.

//=============================================================================
=====
 public Control()
 {

  // --- Call static 'INPUTS' class and load global variables - probably done.
  //Inputs.setInputs();

  // --- Call static method to create FPS growth startup file
  Inputs.setGoplan();

  // --- Call static 'DATABASE' class to define DB connection
  Database.load_database();
```

```
// --- Instantiate 'SILVICS' class
silvicsMaster = new Silvics();

// --- Instantiate static Global Variables from 'INPUTS' class
years = Inputs.getYr_all();
standID = Inputs.getStandId();
end_year = Inputs.getFinal_year();
int end_period = 0;

// --- Figure out which period end_year corresponds to and set marker
while (end_year != years[end_period])
{
 end_period ++;
}

// --- Instantiate Tracking variables for 'Control' Class
rxLog = new int [20][3]; // Dynamic Array to Hold RX's by period
harvestLog = new boolean [20]; // Dynamic Array to determine harvest period

// --- Execute 'step_1' Method to prep Stand and FPS Database
step_1();

// --- Create a Scenario object to keep only the optimum Scenario
Scenario winner = new Scenario();

// --- Obtain and format a numberFormat for output
NumberFormat nf = NumberFormat.getNumberInstance();
nf.setMaximumFractionDigits(2);
nf.setMinimumFractionDigits(2);


//=============================================================================
===
  // LOOP # 1 :Outermost Loop
  //
  // Start outermost loop to pace thru all periods.  Stop loop when a zero
  // year is encountered or all 20 years maximum are cycled.

//=============================================================================
===

  // --- Instantiate marker in Years array to first Report Year
  int yearsPeriod = 0;

  while(yearsPeriod < end_period && years[yearsPeriod] != 0)
  {
   // --- Create File and Header for Text File Output if first Process Year
   if ( yearsPeriod == 0)
   {
    try
    {
     //outputStream = new PrintWriter( new FileOutputStream("OGI_" +
```

```
      // Inputs.getMin_ogi() + "_ENDYEAR_" + end_year + ".txt"));
       outputStream = new PrintWriter( new FileOutputStream("Opti_out.txt"));
      }
      catch(FileNotFoundException fnf)
      {
       System.out.println("*** ERROR *** : Output File Not Found");
       System.exit(0);
      }
      outputStream.println("\t\t\t*** FPS OPTIMIZER RESULTS OUTPUT FILE *** ");
      outputStream.println();
      outputStream.println("OGI_MIN : " + Inputs.getMin_ogi() + "\tEND_YR : " +
       end_year + "\tEVAL_YEAR : " + Inputs.getlook_ahead() +
       "\tSTAND: " + Inputs.getStandId());
      outputStream.println("LOGGING COST : " + Inputs.getIntercept() + "\tTPA Coef : " +
       Inputs.getCoeffA() + "\tBFAC Coef : " + Inputs.getCoeffB());
      outputStream.println("PRE-LOG COST : " + Inputs.getIntercept_p() +
       "\tTPA Coef : " + Inputs.getCoeffA_p());
      outputStream.println(

"=============================================================================
=");
      outputStream.close();
     }

     // --- Print Process Year header to console
     System.out.println(
     "-----------------------------------------------------------------------");
     System.out.println(
     "MESSAGE: Starting Analysis for Process Year " + years[yearsPeriod]);
     System.out.println(
     "-----------------------------------------------------------------------");


//============================================================================
=
     // Determine Live TPA to start with at this year
     //
     // FUNCTION: If 1st period do method 1. For all other periods save a
     // little time by just looking up tpa from the winner object at year.

//============================================================================
=
     float startTpa = (float)0.00;

     switch (yearsPeriod)
     {
      case 0:
       Database.con_go();
       startTpa = silvicsMaster.queryTpa(years[yearsPeriod], standID);
       break;

      default:
```

```
    startTpa = winner.getTpa_postCut(yearsPeriod);
    break;
  }

  // ---------------------------------------------------------
  // Calculate # of cuts allowable per method this year.
  // FORMULA 1: (Starting Live TPA - Minimum TPA to cut to) /
  //          The TPA cutting interval.
  // FORMULA 2: Multiply Formula 1 answer by 3 to account for the three
  //          cutting methods and add 1 for the no-cut scenario.
  //----------------------------------------------------------
  int cutsAllowable = (int)((startTpa - Inputs.getcut_min()) /
   Inputs.getcut_interval());
  int scenariosTotal = (cutsAllowable * 3) + 1;

  // --- Check to make sure total scenarios is not negative and correct
  if (scenariosTotal < 1)
  {
   scenariosTotal = 1;
  }

  //System.out.println("*** DEBUG: Live tpa at year is : " + startTpa);
  //System.out.println("*** DEBUG: ScenariosTotal at year is : " + scenariosTotal);
  //System.out.println("*** DEBUG: cutsAllowed at year is : " + cutsAllowable);

  // --- Create an Array of Scenario objects to store a full period of runs
  Scenario [] collectionScenario = new Scenario [scenariosTotal];

  // --- Create a variable to track the scenario number
  int scenarioNumber = 0;

  // ---------------------------------------------------------
  // LOOP # 2: 2nd Outermost Loop
  //
  // Populates collectionScenario srray with all possible Scenarios
  // for this period.
  //
  // ---------------------------------------------------------
  while(scenarioNumber < scenariosTotal)
  {
   System.out.println("MESSAGE: Analyzing Scenario " + scenarioNumber +
     "\tat Process Year " + years[yearsPeriod]);
   // --- Initialize Sivlics RX variables for use in Loop # 3 switch
   int method = 0;
   int mv = 0;
   int level = 0;

   // --- Set boolean harvest value (1st scenario per period is no-cut)
   if (scenarioNumber == 0)
    harvestLog[yearsPeriod] = false;
   else
    harvestLog[yearsPeriod] = true;
```

```
//-----------------------------------------------------------------
// Loop #3: Determine type of harvest, if any
//
// Use Switch statement to determine harvest type and TPA to cut
//-----------------------------------------------------------------
if (harvestLog[yearsPeriod] == false)
 method = 0; // Signals no-cut
else if (scenarioNumber <= ((scenariosTotal - 1) / 3))
 method = 1; // Signals cut from BELOW
else if (scenarioNumber > ((scenariosTotal - 1) / 3) * 2)
 method = 3; // Signals cut ACROSS all diameters
else
 method = 2; // Signals cut from ABOVE

switch (method)
{
 case 0:
  break;
 case 1:
  mv = 99; // cut from below up to 99 inch dbh
  level = (int)startTpa - (Inputs.getcut_interval() *
       scenarioNumber);
  break;
 case 2:
  mv = 0; // cut from above down to 0 inch dbh
  level = (int)startTpa - (Inputs.getcut_interval() *
       (scenarioNumber - (scenariosTotal / 3)));
  break;
 case 3:
  mv = 1; // cut evenly across dbh with cut to residual ratio of 1
  level = (int)startTpa - (Inputs.getcut_interval() *
       (scenarioNumber - ((scenariosTotal / 3)*2)));
  break;
}

// --- Set the tracking rxLog array with this specific Silvics Rx
rxLog[yearsPeriod][0] = method;
rxLog[yearsPeriod][1] = mv;
rxLog[yearsPeriod][2] = level;

// ----------------------------------------------------------------
// Execute FPS Growth Model with previous periods winning silvics and
// current periods Scenario RX attempt. Create a Scenario object to
// store all values and place in collections array.
// ----------------------------------------------------------------

// --- Clear existing FPS Silvics Table prior to loading new RX
int status_clrSilvics = silvicsMaster.clrSilvics();

// --- Send RX and Harvest Logs to 'setRx' to update FPS SILVICS table
int status_setSilvics = silvicsMaster.setRX(rxLog, harvestLog, years);
```

```
// --- Close DB prior to running FPS modules
Database.con_no();

// --- Run the FPS growth module based on the newly created SILVICS
int status_growFps = Fps.fps_grow();

// --- Create a new Scenario based on results and put in Collection
collectionScenario [scenarioNumber] = loop_4(yearsPeriod, scenarioNumber);

// --- Print quick results at lookahead for Scenario just analyzed
int evalYear = Inputs.getlook_ahead();
int lookP = 0;

// --- Cycle thru Process Years array to find period matching look ahead
while (evalYear != years[lookP])
{
 lookP ++;
}

System.out.println("MESSAGE: Results (Look to Year " + Inputs.getlook_ahead()+ " ) => " +
  "\tOGI: " + nf.format(collectionScenario[scenarioNumber].getOgi(lookP)) +
  "\tPNV: " + nf.format(collectionScenario[scenarioNumber].getPnv(lookP)));
System.out.println(
"-------------------------------------------------------------------------");

// --- Increment 2nd Outer loop to the next scenario in this period
scenarioNumber ++;
}
// FINISH: 2nd Outer loop

// --- Print Results for Process Year of all Scenarios attempted
System.out.println("MESSAGE: Detailed Results for all Scenarios " +
 "at Process Year " + years[yearsPeriod]);

for(int a = 0; a < scenariosTotal; a++)
{
 collectionScenario[a].printTo(yearsPeriod);
}

// --- Run Optimization Algorithm to pick winning scenario for this period
int theWinner = optimize(yearsPeriod, scenariosTotal, collectionScenario);

// --- Copy winning Scenario to 'winner' object
winner = collectionScenario[theWinner];

// --- Print this Periods Winner to Console
System.out.println("MESSAGE: Winning Scenario at Process Year " +
 years[yearsPeriod] + ", Evaluation Year " +
 Inputs.getlook_ahead() + " is # " + theWinner);

// --- Print this Periods Winner to Output Text File
```

```
    try
    {
     //outputStream = new PrintWriter( new FileOutputStream("OGI_" +
     //  Inputs.getMin_ogi() + "_ENDYEAR_" + end_year + ".txt" , true));
     outputStream = new PrintWriter( new FileOutputStream("Opti_out.txt" , true));
    }
    catch(FileNotFoundException fnf)
    {
     System.out.println("*** ERROR *** : Output File Not Found");
     System.exit(0);
    }
    /*outputStream.println("MESSAGE: The Winning Scenario at Process Year " +
     years[yearsPeriod] + ", Evaluating at Year " +
     Inputs.getlook_ahead() + " is SCENARIO " + theWinner);
    outputStream.println(

"=============================================================================
===");*/
    outputStream.close();

    // --- Copy the rx_Log of the winner to the global rx_Log
    System.arraycopy((winner.getRx_variables()), 0, rxLog, 0, 20);

    // --- Copy the harvestLog of the winner to the global harvestLog
    System.arraycopy((winner.getHarvest()), 0, harvestLog, 0, 20);

    // --- Increment Outermost Loop to next period for analysis
    yearsPeriod ++;
   }
   // FINISH: Outermost Loop

   // --- Print out winning path to console
   System.out.println("MESSAGE: The Detailed Winning Path is ===> " );
   for(int zz = 0; zz < 20; zz ++)
   {
    winner.printTo(zz);
   }
   System.out.println(" ****** FINISHED OPTIMIZATION SUCCESSFULLY ****** ");

   // --- Print out winning path to Output Text File
   for(int zz = 0; zz < 20; zz ++)
   {
    winner.printFile(zz);
   }

  }
  // FINISH: 'CONTROL' Constructor


//=============================================================================
===
  // 'STEP_1' Method
```

```
//
// FUNCTION: Cleans FPS Database tables and executes FPS methods to
// establish starting stand conditions.

//=============================================================================
===
private void step_1 ()
{
 // --- Open DB Connection
 Database.con_go();
 int status = 0; // Dynamic variable to determine FPS execution status

 // --- Clear FPS STAND Table
 silvicsMaster.clrStand();

 // --- Clear FPS DBHCLS Table
 silvicsMaster.clrDbhcls();

 // --- Clear FPS SILVICS Table
 silvicsMaster.clrSilvics();

 // --- De-Flag all Stands in FPS ADMIN and re-flag selected Stand_ID
 silvicsMaster.setFlags();

 // --- Load new SILVICS table (Treatement Number 0)
 silvicsMaster.setRX(rxLog, harvestLog, years);

 // --- Close DB Connection
 Database.con_no();

 // --- Run FPS CRUS to compile selected Stand
 try
 {
  status = Fps.fps_Crus();
 }
 catch (Exception crus)
 {
 }

 if(status == 0)
 {
 }
 else
 {
  System.out.println("*** ERROR *** : FPS Crus Failed");
  System.exit(1);
 }

 // --- Update FPS STAND and DBHCLS tables with 'OPTI' as Regime

 status = silvicsMaster.setRegime();
```

```
// --- Run FPS GROW to grow selected Stand to base year
try
{
 status = Fps.fps_growBase(Inputs.getBase());
}
catch (Exception grow)
{
}

if(status == 0)
{
}
else
{
 System.out.println("*** ERROR *** : FPS Grow Failed.");
 System.exit(1);
}
}
// FINISH: 'STEP_1' Method


//=============================================================================
===
// 'LOOP #4' - The innermost looping method
//
// Create a single 'Scenario' object and loop thru each Report Year.
// Populate this 'Scenario' instance variables arrays for each year.
//
// FUNCTION: This object is then stored in an array of 'Scenarios'
// holding all possible RX combos for the current year. The 'Scenario'
// meeting the OGI/PNV constraints out of the array of Scenarios will
// advance to the next period of RX analysis.

//=============================================================================
===
private Scenario loop_4 (int passed_Period, int passed_ScenarioNumber)
{
 // --- Instantiate a blank 'Scenario' object
 Scenario thisScenario = new Scenario();

 // --- Instantiate a 'Calculations' object
 Calculations CalcsMaster = new Calculations();

 // --- Set the period marker to zero to start at beginning of arrays
 int period = 0;

 // --- Set the scenarioNumber passed for this global period of analysis
 thisScenario.setRXnumber(passed_Period, passed_ScenarioNumber);

 // --- Open DB Connection
 Database.con_go();
```

```
// --- Cycle thru each report year until complete (20) or out of years (0)
// --- to calculate the statistics.
while (period < 20 && years[period] != 0)
{
 // --- Set key variables
 thisScenario.setYears(period, years[period]);

 // --- Run ECOLOGY, get variables, populate thisScenario for this year
 float ogi = CalcsMaster.ecology(years[period], standID);
 thisScenario.setOGIvariables(period, CalcsMaster.getSD(),
  CalcsMaster.getTpaAll(), CalcsMaster.getTpaLg(), CalcsMaster.getDbh());
 thisScenario.setOGI(period, ogi);

 // --- Run APPRAISE, get variables, populate thisScenario for this year
 float pnv = CalcsMaster.appraise(years[period], standID);
 thisScenario.setBDFT_cut(period, CalcsMaster.getCut_vol());
 thisScenario.setBDFT_resid(period, CalcsMaster.getResid_vol());
 thisScenario.setPNV(period, pnv);

 // --- Set harvest occurence variable
 thisScenario.setHarvest(period, harvestLog[period]);

 // --- Set RX variables
 thisScenario.setRXvariables(period, rxLog[period][0], rxLog[period][1],
  rxLog[period][2]);

 // --- Set TPA levels pre and post cutting this period
 thisScenario.setTPA_postCut(
  period, silvicsMaster.queryTpa(years[period], standID));
 thisScenario.setTPA_Cut(
  period, silvicsMaster.queryCutTpa(years[period], standID));

 // --- Print all variable info for this Scenario and this period
 //thisScenario.printTo(period);

 // --- Increment to next period and back up to start of while loop
 period ++;
}
// FINISH: While loop for single scenario, all years

// --- Close DB Connection
Database.con_no();

// --- Return thisScenario Object
return thisScenario;
}
// FINISH: 'LOOP4' Method


//=================================================================================
=====
```

```
// 'Optimize' Method
//
// FUNCTION: This method is called in the main control constructor after an
// entire sequence of RX Scenarios have been created for a given year.  It
// then calls the necessary global variables and sends the array of
// Scenarios to the 'Optimize' and 'OptiComparator' classes to be sorted
// in order based on the criteria set there.  It then takes the sorted
// array and finds the winning Scenario number and return this value.

//=============================================================================
=====
private int optimize (int yr, int scenTot, Scenario [] collection)
{
   // --- Set variables for running optimization
   int minOgi = Inputs.getMin_ogi(); // Minimum OGI acceptable
   int lookYear = Inputs.getlook_ahead(); // Actual look ahead year
   int lookPeriod = 0; // The period related to look ahead year
   int win = 0; // The winning Scenario number
   float maxOgi = (float) 0.00; // Max OGI value if Min OGI not met
   int maxOgiRow = 0; // The Scenario number having Max OGI

   // --- Cycle thru Process Years array to find period matching look ahead
   while (lookYear != years[lookPeriod])
   {
    lookPeriod ++;
   }

   // --- Create an instance array of optiClass objects
   Optimize [] quickList = new Optimize[scenTot];

   //-------------------------------------------------------------------
   // Populate 'quicklist' array with scenario results at lookahead year
   //
   // Array consists of Optimize Objects with variables: Scenario #, OGI,
   // and PNV.
   //-------------------------------------------------------------------
   for (int counter = 0; counter < scenTot; counter ++)
   {
    quickList[counter] = new Optimize(collection[counter].getRxNumber(yr),
      collection[counter].getOgi(lookPeriod),
      collection[counter].getPnv(lookPeriod));
   }

   //-------------------------------------------------------------------
   // Sort quickList array or return if no cutting possible this period.
   //
   // If only one Scenario attempted (no-cut), no need to sort.  If more
   // than one Scenario, sort using Comparator interface/class rules.
   //-------------------------------------------------------------------
   if (scenTot == 1 )
   {
    return win;
```

```
}
else
{
 Arrays.sort(quickList, new OptiComparator());
}

// --- Print out sorted quickList to console
System.out.println("MESSAGE: The Sorted Scenarios for Process Year " +
 yr + ", Evaluated at Year " +
 lookYear);
System.out.println("-------------------------------------------------");
for (int row = 0; row < scenTot; row ++)
{
 System.out.println(quickList[row]);
}
System.out.println("-------------------------------------------------");

// --- Print out sorted quickList to Output Text File
try
{
 //outputStream = new PrintWriter( new FileOutputStream("OGI_" +
 //  Inputs.getMin_ogi() + "_ENDYEAR_" + end_year + ".txt" , true));
 outputStream = new PrintWriter( new FileOutputStream("Opti_out.txt", true));
}
catch(FileNotFoundException fnf)
{
 System.out.println("*** ERROR *** : Output File Not Found");
 System.exit(0);
}
/*outputStream.println("The Sorted Scenarios for Process Year : " +
 years[yr] + ", Evaluated at Year : " +
 lookYear);
outputStream.println(
"-------------------------------------------------------------------");
for (int row = 0; row < scenTot; row ++)
{
 outputStream.println(quickList[row]);
}
outputStream.println("-------------------------------------------------");*/
outputStream.close();

//-------------------------------------------------------------------
// Choose winner based on Maximum PNV s.t. Minimum OGI
//
// If Minimum OGI not met for any Scenario, select Maximum OGI
//-------------------------------------------------------------------
for (int row = 0; row < scenTot; row ++)
{
 // --- Keep tabs on Scenario with Max OGI in case Min OGI not met
 if (quickList[row].getOgi() > maxOgi)
 {
  maxOgi = quickList[row].getOgi();
```

```
        maxOgiRow = row;
       }
       // --- Check lagest PNV scenarios in descending order to see if min OGI
       //     constraint met.  If so, send that Scenario as winner.
       if (quickList[row].getOgi() >= Inputs.getMin_ogi())
       {
        win = quickList[row].getRx();
        return win;
       }
      }

      // --- If the code reaches this point, Min OGI not met, return Scenario
      //     with Max OGI.
      return quickList[maxOgiRow].getRx();
   }
   // FINISH: 'Optimize' Method

}
// FINSIH: 'CONTROL' Public Class
```

```
package thesis;

/*****************************************************************************
 * 'INPUTS' class
 *
 * FUNCTION: This class holds global variables which are referenced by the other
 * classes.  Its constructor instantiates these variables by calling the
 * class methods to read the data files that contain the variable data.
 * This class is instantiated by the MASTER class upon opening.
 *
 *****************************************************************************/

import drasys.or.matrix.*;
import drasys.or.stat.model.*;
import java.io.*;
import java.util.*;
import javax.swing.JOptionPane;
import java.math.*;

public class Inputs
{
 // Economic variables
 private static int units; // 1 = inputs are metric, 2 = inputs are english
 private static float rate; // The 'real' discount rate for economics
 private static short base; // The current year used as a base for economics
 private static int [] yr_all = new int [20]; // Array holding report years
 private static String [] price_spc = new String [10]; // Log price species
 private static int [] []  price_sorts = new int [10] [3]; // Log prices
 private static int min_ogi; // Minimum OGI acceptable for optimization
 private static int annualCost; // Annual recurring cost amount
 private static int haulCost; // $/MBF trucking cost to the mill
 private static double [] dep = new double [25]; // Matrix logging costs
 private static double [][] indep = new double [25][2]; // Matrix logging cost value pairs
 private static double intercept;
 private static double coeffA;
 private static double coeffB;
 private static double [] dep_p = new double [5]; // ditto for pre-commercial
 private static double [][] indep_p = new double [5][1]; // ditto for pre-commercial
 private static double intercept_p;
 private static double coeffA_p;
 private static float bdft_min; // Minimum bdft cut volume triggering P.C.T.
 private static int pd_zero_cost; // Period zero cost
 private static int resid_value; // Residual value at end of rotation & cc
 private static int econ_method; // The LEV calculation method


 // Min/Max values from OGI publication
 private static float sd_dbh_young;
 private static float sd_dbh_old;
 private static float tpa_lg_young;
 private static float tpa_lg_old;
 private static float dbh_young;
```

```
private static float dbh_old;
private static float tpa_all_young;
private static float tpa_all_old;
private static float tpa_inflection; //The tpa_all where OGI reverses

// Additional variables
private static float tpa_min; // Optional minimum TPA for OGI formula
private static int cut_interval; // TPA decrement level for harvesting
private static int cut_min; // Minimum TPA to cut down to
private static int look_ahead; // Year to evaluate OGI & PNV constraints
private static float a; // Metric diameter conversion factor
private static float b; // Metric area converion factor
private static int standId; // The FPS StandID
private static int final_year; // The year to 'CC' at and stop model
private static String dbName = null; // GUI provided DB name for DBQ method

/*****************************************************************************
 * 'INPUTS' Constructor
 *
 * FUNCTION:  Nothing right now
 *****************************************************************************/
public Inputs()
{
}

/*****************************************************************************
 * 'SETINPUTS' Methods
 *
 * FUNCTION: Method used to populate the global static variables.  Looks
 * for a file called 'OPTI_IN.TXT' in the 'OPTI' directory in the FPS Path.
 * Reads all the variables stored there and instantiates global variables.
 * This file is created by the GUI, which handles validity checks.
 *
 *****************************************************************************/
public static int setInputs()
{
 System.out.println("Setting Inputs Global Variables...");

 // START: Find FPS Directory and define URL to the 'Opti_In.txt' file
 try
 {
  String user_dir = System.getProperty("user.dir");
  File f1 = new File(user_dir);
  String FPSopti_dir = user_dir;

  // *** Temporarily hardwire FPS Opti Path ***
  //FPSopti_dir = "C:\\FPS\\Opti";

  File database = new File(FPSopti_dir + "\\Opti_in.txt");
  if (database.exists())
  {
   System.out.println("Found Opti_in file, reading variables now...");
```

```
String value;
int variables = 33; // # of variable lines in 'OPTI_IN.TXT' file
FileReader databaseReader = new FileReader(database);
BufferedReader reader = new BufferedReader(databaseReader);

for (int row = 1; row <= variables; row ++)
{
 // --- Read in a line of data from file
 value = reader.readLine();
 System.out.println("Reading inputs row :" + row);

 if (value.startsWith("UNITS"))
 {
  StringTokenizer t = new StringTokenizer(value);
  t.nextToken();
  units = Integer.parseInt(t.nextToken());
  System.out.println("UNITS read from file : " + units);
 }

 else if (value.startsWith("MIN_OGI"))
 {
  StringTokenizer t = new StringTokenizer(value);
  t.nextToken();
  min_ogi = Integer.parseInt(t.nextToken());
  System.out.println("MIN_OGI read from file : " + min_ogi);
 }

 else if (value.startsWith("STAND_ID"))
 {
  StringTokenizer t = new StringTokenizer(value);
  t.nextToken();
  standId = Integer.parseInt(t.nextToken());
  System.out.println("STANDID read from file : " + standId);

  //else standId = 0;
 }

 else if (value.startsWith("ANNUAL_COST"))
 {
  StringTokenizer t = new StringTokenizer(value);
  t.nextToken();
  annualCost = Integer.parseInt(t.nextToken());
  System.out.println("ANNUAL_COST read from file : " + annualCost);
 }

 else if (value.startsWith("HAUL_COST"))
 {
  StringTokenizer t = new StringTokenizer(value);
  t.nextToken();
  haulCost = Integer.parseInt(t.nextToken());
  System.out.println("HAUL_COST read from file : " + haulCost);
 }
```

```
else if (value.startsWith("BDFT_MIN"))
{
 StringTokenizer t = new StringTokenizer(value);
 t.nextToken();
 bdft_min = Integer.parseInt(t.nextToken());
 System.out.println("BDFT_MIN read from file : " + bdft_min);
}

else if (value.startsWith("CUT_INTERVAL"))
{
 StringTokenizer t = new StringTokenizer(value);
 t.nextToken();
 cut_interval = Integer.parseInt(t.nextToken());
 System.out.println("CUT_INTERVAL read from file : " + cut_interval);
}

else if (value.startsWith("CUT_MIN"))
{
 StringTokenizer t = new StringTokenizer(value);
 t.nextToken();
 cut_min = Integer.parseInt(t.nextToken());
 System.out.println("CUT_MIN read from file : " + cut_min);
}

else if (value.startsWith("LOOK_AHEAD"))
{
 StringTokenizer t = new StringTokenizer(value);
 t.nextToken();
 look_ahead = Integer.parseInt(t.nextToken());
 System.out.println("LOOK_AHEAD read from file : " + look_ahead);
}

else if (value.startsWith("FINAL_YEAR"))
{
 StringTokenizer t = new StringTokenizer(value);
 t.nextToken();
 final_year = Integer.parseInt(t.nextToken());
 System.out.println("FINAL_YEAR read from file : " + final_year);
}

else if (value.startsWith("TPA_MIN"))
{
 StringTokenizer t = new StringTokenizer(value);
 t.nextToken();
 tpa_min = Integer.parseInt(t.nextToken());
 System.out.println("TPA_MIN read from file : " + tpa_min);
}

else if (value.startsWith("RATE"))
{
 StringTokenizer t = new StringTokenizer(value);
```

```
 t.nextToken();
 rate = Float.parseFloat(t.nextToken());
 System.out.println("RATE read from file : " + rate);
}

else if (value.startsWith("BASE"))
{
 StringTokenizer t = new StringTokenizer(value);
 t.nextToken();
 base = Short.parseShort(t.nextToken());
 System.out.println("BASE read from file : " + base);
}

else if (value.startsWith("SD_DBH_YOUNG"))
{
 StringTokenizer t = new StringTokenizer(value);
 t.nextToken();
 sd_dbh_young = Float.parseFloat(t.nextToken());
 System.out.println("SD_DBH_YOUNG read from file : " + sd_dbh_young);
}

else if (value.startsWith("SD_DBH_OLD"))
{
 StringTokenizer t = new StringTokenizer(value);
 t.nextToken();
 sd_dbh_old = Float.parseFloat(t.nextToken());
 System.out.println("SD_DBH_OLD read from file : " + sd_dbh_old);
}

else if (value.startsWith("TPA_LG_YOUNG"))
{
 StringTokenizer t = new StringTokenizer(value);
 t.nextToken();
 tpa_lg_young = Float.parseFloat(t.nextToken());
 System.out.println("TPA_LG_YOUNG read from file : " + tpa_lg_young);
}

else if (value.startsWith("TPA_LG_OLD"))
{
 StringTokenizer t = new StringTokenizer(value);
 t.nextToken();
 tpa_lg_old = Float.parseFloat(t.nextToken());
 System.out.println("TPA_LG_OLD read from file : " + tpa_lg_old);
}

else if (value.startsWith("DBH_YOUNG"))
{
 StringTokenizer t = new StringTokenizer(value);
 t.nextToken();
 dbh_young = Float.parseFloat(t.nextToken());
 System.out.println("DBH_YOUNG read from file : " + dbh_young);
}
```

```
else if (value.startsWith("DBH_OLD"))
{
 StringTokenizer t = new StringTokenizer(value);
 t.nextToken();
 dbh_old = Float.parseFloat(t.nextToken());
 System.out.println("DBH_OLD read from file : " + dbh_old);
}

else if (value.startsWith("TPA_ALL_YOUNG"))
{
 StringTokenizer t = new StringTokenizer(value);
 t.nextToken();
 tpa_all_young = Float.parseFloat(t.nextToken());
 System.out.println("TPA_ALL_YOUNG read from file : " + tpa_all_young);
}

else if (value.startsWith("TPA_ALL_OLD"))
{
 StringTokenizer t = new StringTokenizer(value);
 t.nextToken();
 tpa_all_old = Float.parseFloat(t.nextToken());
 System.out.println("TPA_ALL_OLD read from file : " + tpa_all_old);
}

else if (value.startsWith("YR_ALL"))
{
 StringTokenizer t = new StringTokenizer(value);
 t.nextToken();
 for (int x = 0; x < 20; x++) // * NOTE: Requires 20 values
 {
  yr_all [x] = Integer.parseInt(t.nextToken());
  //System.out.println("YEAR_ALL read from file : " + yr_all [x]);
 }
}

else if (value.startsWith("PRICE_SPC"))
{
 StringTokenizer t = new StringTokenizer(value);
 t.nextToken();
 for (int x = 0; x < 10; x++) // * NOTE: Requires 10 values
 {
  price_spc [x] = t.nextToken();
  //System.out.println("PRICE_SPC read from file : " + price_spc [x]);
 }
}

else if (value.startsWith("PRICE_SORTS1"))
{
 StringTokenizer t = new StringTokenizer(value);
 t.nextToken();
 for (int x = 0; x < 10; x++) // * NOTE: Requires 10 values
```

```
  {
   price_sorts [x][0] = Integer.parseInt(t.nextToken());
   //System.out.println("PRICE_SORTS1 read from file : " + price_sorts [x][0]);
  }
 }

 else if (value.startsWith("PRICE_SORTS2"))
 {
  StringTokenizer t = new StringTokenizer(value);
  t.nextToken();
  for (int x = 0; x < 10; x++) // * NOTE: Requires 10 values
  {
   price_sorts [x][1] = Integer.parseInt(t.nextToken());
   //System.out.println("PRICE_SORTS2 read from file : " + price_sorts [x][1]);
  }
 }

 else if (value.startsWith("PRICE_SORTS3"))
 {
  StringTokenizer t = new StringTokenizer(value);
  t.nextToken();
  for (int x = 0; x < 10; x++) // * NOTE: Requires 10 values
  {
   price_sorts [x][2] = Integer.parseInt(t.nextToken());
   //System.out.println("PRICE_SORTS3 read from file : " + price_sorts [x][2]);
  }
 }

 else if (value.startsWith("DEP_COST"))
 {
  StringTokenizer t = new StringTokenizer(value);
  t.nextToken();
  for (int x = 0; x < 25; x++) // * NOTE: Requires 25 values
  {
   dep [x] = Integer.parseInt(t.nextToken());
   //System.out.println("DEP costs read from file : " + dep [x]);
  }
 }

 else if (value.startsWith("PCT_DEP_COST"))
 {
  StringTokenizer t = new StringTokenizer(value);
  t.nextToken();
  for (int x = 0; x < 5; x++) // * NOTE: Requires 5 values
  {
   dep_p [x] = Integer.parseInt(t.nextToken());
   //System.out.println("DEP_P costs read from file : " + dep_p [x]);
  }
 }

 else if (value.startsWith("DB_NAME"))
 {
```

```
  StringTokenizer t = new StringTokenizer(value);
  t.nextToken();
  dbName = t.nextToken();
  System.out.println("DB_Name read from Opti_In file : " + dbName);
 }

 else if (value.startsWith("PD_ZERO_COST"))
 {
  StringTokenizer t = new StringTokenizer(value);
  t.nextToken();
  pd_zero_cost = Integer.parseInt(t.nextToken());
  System.out.println("PD_ZERO_COST read from file : " + pd_zero_cost);
 }

 else if (value.startsWith("RESID_VALUE"))
 {
  StringTokenizer t = new StringTokenizer(value);
  t.nextToken();
  resid_value = Integer.parseInt(t.nextToken());
  System.out.println("RESID_VALUE read from file : " + resid_value);
 }

 else if (value.startsWith("ECON_METHOD"))
 {
  StringTokenizer t = new StringTokenizer(value);
  t.nextToken();
  econ_method = Integer.parseInt(t.nextToken());
  System.out.println("ECON_METHOD read from file : " + econ_method);
 }

 else if (value.startsWith("OGI_INFLECTION"))
 {
  StringTokenizer t = new StringTokenizer(value);
  t.nextToken();
  tpa_inflection = Integer.parseInt(t.nextToken());
  System.out.println("TPA_INFLECTION read from file : " + tpa_inflection);
 }


 // --- Catch Incomplete or Missing 'Opt_In.txt' File
 else
 {
 JOptionPane.showMessageDialog(null,"'Opti_In.txt' File Not Found " ,
  "Or File Incomplete",
  JOptionPane.INFORMATION_MESSAGE);
 System.exit(0);
 }

}// END: 'for loop' cycling thru file

// --- Close Reader and File
reader.close();
```

```
   databaseReader.close();
  }
 }
 catch (Exception noURL)
 {
  //System.out.println("** ERROR ** : Unable to find URL");
 }

 // --- The Independent variables for Logging and PCT costs are hard-wired
 indep [0][0] = 10; // Example: 10 TPA
 indep [0][1] = Math.sqrt(2000); // Example: Matched with 2000 BDFT/ACRE logged
 indep [1][0] = 10;
 indep [1][1] = Math.sqrt(5000);
 indep [2][0] = 10;
 indep [2][1] = Math.sqrt(10000);
 indep [3][0] = 10;
 indep [3][1] = Math.sqrt(25000);
 indep [4][0] = 10;
 indep [4][1] = Math.sqrt(50000);

 indep [5][0] = 50;
 indep [5][1] = Math.sqrt(2000);
 indep [6][0] = 50;
 indep [6][1] = Math.sqrt(5000);
 indep [7][0] = 50;
 indep [7][1] = Math.sqrt(10000);
 indep [8][0] = 50;
 indep [8][1] = Math.sqrt(25000);
 indep [9][0] = 50;
 indep [9][1] = Math.sqrt(50000);

 indep [10][0] = 100;
 indep [10][1] = Math.sqrt(2000);
 indep [11][0] = 100;
 indep [11][1] = Math.sqrt(5000);
 indep [12][0] = 100;
 indep [12][1] = Math.sqrt(10000);
 indep [13][0] = 100;
 indep [13][1] = Math.sqrt(25000);
 indep [14][0] = 100;
 indep [14][1] = Math.sqrt(50000);

 indep [15][0] = 200;
 indep [15][1] = Math.sqrt(2000);
 indep [16][0] = 200;
 indep [16][1] = Math.sqrt(5000);
 indep [17][0] = 200;
 indep [17][1] = Math.sqrt(10000);
 indep [18][0] = 200;
 indep [18][1] = Math.sqrt(25000);
 indep [19][0] = 200;
 indep [19][1] = Math.sqrt(50000);
```

```
indep [20][0] = 300;
indep [20][1] = Math.sqrt(2000);
indep [21][0] = 300;
indep [21][1] = Math.sqrt(5000);
indep [22][0] = 300;
indep [22][1] = Math.sqrt(10000);
indep [23][0] = 300;
indep [23][1] = Math.sqrt(25000);
indep [24][0] = 300;
indep [24][1] = Math.sqrt(50000);

indep_p [0][0] = 10;
indep_p [1][0] = 50;
indep_p [2][0] = 100;
indep_p [3][0] = 200;
indep_p [4][0] = 300;

// START: Metric to English Conversion of OGI variables, if needed
if(units == 1)
{
 a = (float) 2.54; // inch to centimeter conversion factor
 b = (float) .40468; // acre to hectare conversion factor
}
else
{
 a = 1;
 b = 1;
}

sd_dbh_young = sd_dbh_young / a;
sd_dbh_old = sd_dbh_old / a;
tpa_lg_young = tpa_lg_young * b;
tpa_lg_old = tpa_lg_old * b;
dbh_young = dbh_young / a;
dbh_old = dbh_old / a;
tpa_all_young = tpa_all_young * b;
tpa_all_old = tpa_all_old * b;
tpa_min = tpa_min * b;
// FINISH: Metric to English Conversion of OGI, if needed

// --- Call Regression Method to get Logging and PCT cost coefficients
try
{
 regressCosts();
}
catch (Exception costs)
{
 System.err.println("*** ERROR *** : Cost Regression Failed");
 costs.printStackTrace();
}
return 1;
```

```
}
// FINISH: 'SETINPUTS' Method


//=============================================================================
=====
 // 'regressCosts' Method
 //
 // FUNCTION: performs a multiple linear regression on two independent
 // variables (tpa and bdft/acre harvested) against the dependent variable
 // $/MBF logging cost. The array of variables are set from the 'setInputs'
 // Method which reads the tables from a text file, which were intially
 // provided by a user in the GUI.
 // The result is the instantiating of the Class variables representing
 // the intercept and a and b coefficients of the logging cost formula.
 // The procedure is replicated for another single dependent variable
 // array representing pre-commercial thinning costs (TPA only)

//=============================================================================
=====
 public static void regressCosts ()
 {
  // **** NOTE: Implement a dependendent transformation if needed!!! And
  // **** Make sure to re-transform in 'Appraise' section when applying!

  // --- Generate Commercial Logging Costs
  VectorI dependent = new DenseVector(dep);
  MatrixI independent = new RowArrayMatrix(indep, true);

  LinearRegressionI reg = new ReverseLinear(dependent, independent);

  VectorI coef = reg.getCoefficients();
  intercept = coef.elementAt(2);
  coeffA = coef.elementAt(0);
  coeffB = coef.elementAt(1);
  //System.out.println(reg.toString());

  // --- Generate PRE-Commercial thinning costs
  dependent = new DenseVector (dep_p);
  independent = new RowArrayMatrix(indep_p, true);

  reg = new ReverseLinear(dependent, independent);

  coef = reg.getCoefficients();
  intercept_p = coef.elementAt(1);
  coeffA_p = coef.elementAt(0);
  //System.out.println(reg.toString());
 }

 /*****************************************************************************
  * 'setGoplan' Method
```

```
*
* FUNCTION: Takes the new 'yr_all' array and re-writes the FPS growth
* startup file 'goPlan.ini' with these new report years.
*****************************************************************************/
public static void setGoplan ()
{
 PrintWriter outputStream = null;
 try
 {
  String user_dir = System.getProperty("user.dir");
  File f1 = new File(user_dir);
  String FPSdata_dir = (String)f1.getParent() + "\\Data";
  outputStream = new PrintWriter( new FileOutputStream(FPSdata_dir + "\\GoOpti.ini"));
  outputStream.println(
   "START      .... DF    100 1990    1 E 2 N N Y Y .850");
  outputStream.println("LABEL");
  outputStream.println("FILE DATABASE");
  outputStream.print("YEARS    1");
  for (int b = 0; b < 10; b ++)
  {
   if (yr_all[b] != 0)
   {
    outputStream.print(" " + yr_all[b]);
   }
   else
   {
    outputStream.print("    " + yr_all[b]);
   }
  }
  outputStream.println();
  outputStream.print("YEARS    2");
  for (int b = 10; b < 20; b ++)
  {
   if (yr_all[b] != 0)
   {
    outputStream.print(" " + yr_all[b]);
   }
   else
   {
    outputStream.print("    " + yr_all[b]);
   }
  }
  outputStream.close();
 }
 catch(FileNotFoundException fnf)
 {
  System.out.println("*** ERROR *** : 'GoOpti.ini' File Not Found");
  System.exit(0);
 }
}
// FINISH: 'setGoplan' Method
```

```
/*****************************************************************************
 * 'GETTER' Methods
 *
 * FUNCTION: Return the private global variables to calling class
 *****************************************************************************/

public static int getUnits ()
{
 return units;
}

public static float getRate ()
{
 return rate;
}

public static int getAnnualCost ()
{
 return annualCost;
}

public static int getHaulCost ()
{
 return haulCost;
}

public static float getIntercept ()
{
 return (float)intercept;
}

public static float getCoeffA ()
{
 return (float)coeffA;
}

public static float getCoeffB ()
{
 return (float)coeffB;
}

public static float getIntercept_p ()
{
 return (float)intercept_p;
}

public static float getCoeffA_p ()
{
 return (float)coeffA_p;
}

public static float getBdftMin ()
```

```
{
 return bdft_min;
}

public static short getBase ()
{
 return base;
}

public static int getcut_interval ()
{
 return cut_interval;
}

public static int getPdZeroCost ()
{
 return pd_zero_cost;
}

public static int getResidValue ()
{
 return resid_value;
}

public static int getEconMethod ()
{
 return econ_method;
}

public static int getcut_min ()
{
 return cut_min;
}

public static float getTpaInflection ()
{
 return (float)tpa_inflection;
}

public static int getlook_ahead ()
{
 return look_ahead;
}

public static int getFinal_year ()
{
 return final_year;
}

public static int [] getYr_all ()
{
 return yr_all;
```

```
}

public static String [] getPrice_spc ()
{
 return price_spc;
}

public static int [] [] getPrice_sorts ()
{
 return price_sorts;
}

public static int getMin_ogi ()
{
 return min_ogi;
}

public static float getSd_dbh_young ()
{
 return sd_dbh_young;
}

public static float getSd_dbh_old ()
{
 return sd_dbh_old;
}

public static float getTpa_lg_young ()
{
 return tpa_lg_young;
}

public static float getTpa_lg_old ()
{
 return tpa_lg_old;
}

public static float getDbh_young ()
{
 return dbh_young;
}

public static float getDbh_old ()
{
 return dbh_old;
}

public static float getTpa_all_young ()
{
 return tpa_all_young;
}
```

```
 public static float getTpa_all_old ()
 {
  return tpa_all_old;
 }

 public static int getStandId ()
 {
  return standId;
 }

 public static String getDbName ()
 {
  return dbName;
 }
 // FINISH: 'GETTER' Methods

}
// END: 'INPUTS' Class
```

```
package thesis;

/*****************************************************************************
 * DATABASE Class
 *
 * FUNCTION: Serves as centralized class for initializing database, connection
 * properties, and error handling
 *****************************************************************************/

import JData2_0.sql.*;
import java.sql.*;
import java.io.*;
import javax.swing.JOptionPane;
import java.util.Properties;

public class Database
{

 // CONNECTION and STATEMENT variables declarations
 public static Connection con1; // database connection object
 public static String url = null; // full database location url
 public static String odbc = null; // odbc DSN datasource name, old method
 public static String dbq = null; // actual Full DB if using DBQ property

/*****************************************************************************
 * 'LOAD_DRIVER' Method
 *
 * FUNCTION: Loads database driver
 *****************************************************************************/
 public static void load_driver()
 {
  // START: Load a driver
  try {
   //Class.forName("sun.jdbc.odbc.JdbcOdbcDriver"); // Using the ODBC-BRIDGE
   Class.forName("JData2_0.sql.$Driver" ); // Using the JDATA driver
  }
  catch (ClassNotFoundException cnfe)
  {
   System.out.println("** WARNING ** :Unable to load database driver");
   System.err.println(cnfe);
  }
 }
 //  FINSIH: 'LOAD_DRIVER' Method

/*****************************************************************************
 * 'LOAD_DATABASE' Method
 *
 * FUNCTION: Loads database name and location for connection.
 * 1.  1st checks to see if user specified a DB Name in GUI ('dbq' variable).
 *     If they didnt, it looks to the FPS 'database.ini' file and determines
 *      if it is the old or new format.  Reads the ODBC DSN name if old and
 *      stores in 'odbc'.  If new method, stores full path/name in 'dbq'.
```

```
* 2.  If the user did specify a 'dbName', adds path and sets as 'dbq'.
*
*****************************************************************************/
public static void load_database()
{

 System.out.println("The Load DB Method has been called");
 // START: Find FPS Directory and define URL
 try
 {
  String user_dir = System.getProperty("user.dir");
  File f1 = new File(user_dir);
  String FPSdata_dir = (String)f1.getParent() + "\\Data";
  String name = Inputs.getDbName();

  // *** Temporarily hardwire FPS Path ***
  //FPSdata_dir = "C:\\FPS\\Data";

  File database = new File(FPSdata_dir + "\\database.ini");


 System.out.println("First try block finished, going to search...");

  // -- If the user didn't select a DB name AND 'database.ini' exists
  if (name == null & database.exists())
  {
   System.out.println("DBName is Null but DB.INI exists...");
   String value;
   FileReader databaseReader = new FileReader(database);
   BufferedReader reader = new BufferedReader(databaseReader);
   value = reader.readLine();

   // --- Old style of ODBC 'database.ini' output
   if (value.startsWith("FPS="))
   {
    System.out.println("No DB Name, but old FPS style ODBC");
    int index = 0;
    index = value.indexOf("   ");
    odbc = value.substring(4,index);
    System.out.println("the ODBC datasource name: " + odbc);
    reader.close();
    databaseReader.close();
   }

   // --- Arney's New style of using DBQ in 'database.ini' output
   else if (value.startsWith("-"))
   {
    System.out.println("NO DBName, but new DBQ FPS style");
    reader.readLine(); // move to 2nd line in file
    reader.readLine(); // move to 3rd line in file
    value = reader.readLine(); // move to 4th line in file and store
    dbq = value.substring(6, (value.lastIndexOf(".mdb")) + 4);
```

```
  reader.close();
  databaseReader.close();
 }

 else
 {
  JOptionPane.showMessageDialog(null, "Database.ini file not found " ,
    " " ,JOptionPane.INFORMATION_MESSAGE);
  reader.close();
  databaseReader.close();
  System.exit(0);
 }

}// END: Main 'if' for database.ini validate routine

// --- The user GUI gave a DB name, so use DBQ method
else if (name != null)
{
 dbq = FPSdata_dir + "\\" + name;
 System.out.println("The user gave a DBNAME any my DBQ is : " + dbq);
 // --- Send the GUI provided DBQ to 'database.ini' file for FPS use
 PrintWriter outputStream = null;
 try
 {
  outputStream = new PrintWriter( new FileOutputStream(FPSdata_dir +
    "\\database.ini"));
 }
 catch(FileNotFoundException fnf)
 {
  System.out.println("*** ERROR *** : Output File Not Found");
  System.exit(0);
 }
 outputStream.println(
 "-------------------------------------------------------------------------");
 outputStream.println(
 "|                        FPSData                          |");
 outputStream.println(
 "-------------------------------------------------------------------------");
 int whitespace = 72 - 6 - dbq.length();
 String tempFormat =
 "                                                        |";
 String tempDbq = "| DBQ=" + dbq + tempFormat.substring((6 + dbq.length()));
 outputStream.println(tempDbq);
 outputStream.println(
 "-------------------------------------------------------------------------");
 outputStream.close();


}

// --- Notify that nothing found at all referencing DB!
else
```

```
  {
   JOptionPane.showMessageDialog(null, "Database.ini file not found " ,
     "and No GUI DB name found " ,JOptionPane.INFORMATION_MESSAGE);
   System.exit(0);
  }
 }// END: try  block

 catch (Exception noURL)
 {
  System.out.println("** Warning ** :Unable to find URL");
 }
 // FINISH: Find FPS Directory and URL

}
// FINISH: LOAD_DATABASE Method

/*****************************************************************************
* 'CON_GO' Method
*
* FUNCTION: Method to establish connection to database
*****************************************************************************/
public static void con_go()
{

 // --- THE Old Method for specifying DB connection to the driver
 if (dbq == null)
 {
 //url = "jdbc:odbc:" + odbc; // For use with ODBC Bridge
 url = "jdbc:JDataConnect://localhost/" + odbc; // For use with JDATA driver
 }

 // --- THE New Method for specifying DB connection using DBQ variant
 Properties props = new Properties();
 props.put("connectString", "DRIVER={Microsoft Access Driver (*.mdb)}; DBQ=" + dbq);
 url = "jdbc:JDataConnect://localhost/";

 //System.out.println("Con_Go has just been called...");
 //System.out.println("URL: " + url);
 //System.out.println("ODBC: " + odbc);

 try
 {
  // Use ODBC DSN 'odbc' variable
  if (dbq == null)
  {
   con1 = DriverManager.getConnection(url);
   //System.out.println("Opening Database Connection...");
  }
  // Use Newer direct DBQ method with 'dbq' variable
  else
  {
   con1 = DriverManager.getConnection(url, props);
```

```
    }
   }
   catch (SQLException nocon)
   {
    System.out.println("** Warning ** : Unable to connect to database. ");
    String sqlMessage = nocon.getMessage();
    String sqlState = nocon.getSQLState();
    int vendorCode = nocon.getErrorCode();
    System.err.println("Message: " + sqlMessage);
    System.err.println("SQL State: " + sqlState);
    System.err.println("Vendor code: " + vendorCode);
    System.out.println("---------------------------");
   }
  }
  // FINISH: CON_GO Method

  /*****************************************************************************
  * 'CON_NO' Method
  *
  * FUNCTION: Method to terminate connection to database
  *****************************************************************************/
  public static void con_no() {
   try {
    con1.close();
    //System.out.println("Closing Database connection...");
   }
   catch (SQLException noclose) {
    System.out.println("** Warning ** : Unable to close database. ");
    String sqlMessage = noclose.getMessage();
    String sqlState = noclose.getSQLState();
    int vendorCode = noclose.getErrorCode();
    System.err.println("Message: " + sqlMessage);
    System.err.println("SQL State: " + sqlState);
    System.err.println("Vendor code: " + vendorCode);
    System.out.println("---------------------------");
   }
  }
  // FINISH: CON_NO Method
}
// FINISH: Public Class DATABASE
```

```
package thesis;

/*****************************************************************************
 * 'FPS' Public Class
 *
 * FUNCTION: Serves to hold methods for accessing runtime environment and
 * manipulating FPS software directly.
 *
 *****************************************************************************/

import java.lang.Runtime;
import java.lang.Process;
import java.io.IOException;
import java.sql.*;
import java.lang.System;

public class Fps
{
 // --- Declare runtime and process variables
 private static Runtime r;
 private static Process p;


//============================================================================
=====
 // 'FPS' Constructor
 //
 // FUNCTION:

//============================================================================
=====
 public Fps()
 {

 }
 // FINISH: 'FPS' Constructor


//============================================================================
=====
 // 'FPS_grow' Method
 //
 // FUNCTION: Executes the FPS GROW module using 'goplan.ini' startup file

//============================================================================
=====
 public static int fps_grow ()
 {
  // --- Declare return variable reporting on status of execution
  int status = 1;

  // --- Pause the program before running FPS tools
```

```
try
{
 long time = System.currentTimeMillis();
 long waitTill = time + 1500;
 while (time < waitTill)
 {
  time = System.currentTimeMillis();
 }
}
catch (Exception e)
{
}

// --- Verify that DB Connection is closed first
try
{
 if (Database.con1.isClosed() == false)
 {
  //System.out.println("MESSAGE: DB Open - Closing DB Connection now...");
  Database.con_no();
 }
 else
 {
  //System.out.println("MESSAGE: DB Verified as Closed...");
 }
}
catch (Exception ogiSql)
{
 System.err.println(ogiSql.getMessage());
 ogiSql.printStackTrace();
}

// --- Instantiate Runtime and Process variables
r = Runtime.getRuntime();
p = null;

// --- Display Memory prior to running FPS
//System.out.println("MESSAGE: Free Memory, pre-FPS = " + r.freeMemory() +
//  "\t Available Memory = " + r.totalMemory());

// --- Sleep for 1 second


// --- Start FPS Growth Module
try
{
 // --- Execute FPS Growth Module
 p = r.exec("c:/FPS/FPSGrow -GoOpti.ini -all");

 try
 {
 // --- Wait for FPS Growth Module to finish
```

```
    status = p.waitFor();
    //System.out.println("MESSAGE: FPS GROW Exit Status => " + status);
    //status = p.exitValue();
    //System.out.println("MESSAGE: Stand Grown to All Years.");
    }
    catch (InterruptedException wait)
    {
     System.err.println("*** ERROR *** :" + wait);
    }
  }
  catch(IOException fps)
  {
   System.err.println("*** ERROR *** : " + fps);
  }
  // --- Attempt to finalize and garbage collect
  r.runFinalization();
  r.gc();

  // --- Display Memory prior to running FPS
  //System.out.println("MESSAGE: Free Memory, pst-FPS = " + r.freeMemory() +
  //  "\t Available Memory = " + r.totalMemory());

  // --- Pause the program before running FPS tools
  // --- Pause the program before running FPS tools
  try
  {
   long time = System.currentTimeMillis();
   long waitTill = time + 2000;
   while (time < waitTill)
   {
    time = System.currentTimeMillis();
   }
  }
  catch (Exception e)
  {
  }

  // --- Return status
  return status;
 }
 // FINISH: 'FPS_GROW' Method


//=================================================================================
=====
 // 'FPS_growBase' Method
 //
 // FUNCTION: Executes the FPS GROW and grows stand to the BASE year from
 // the Inputs class (passed in)

//=================================================================================
=====
```

```
public static int fps_growBase (short base)
{
 // --- Declare return variable reporting on execution status
 int status = 1;

 // --- Verify that DB Connection is closed first
 try
 {
  if (Database.con1.isClosed() == false)
  {
   //System.out.println("MESSAGE: DB Open - Closing DB Connection now...");
   Database.con_no();
  }
  else
  {
   //System.out.println("MESSAGE: DB Verified as Closed...");
  }
 }
 catch (Exception ogiSql)
 {
  System.err.println(ogiSql.getMessage());
  ogiSql.printStackTrace();
 }

 // --- Instantiate Runtime and Process variables
 r = Runtime.getRuntime();
 p = null;

 // --- Pause the program before running FPS tools
 try
 {
  long time = System.currentTimeMillis();
  long waitTill = time + 2000;
  while (time < waitTill)
  {
   time = System.currentTimeMillis();
  }
 }
 catch (Exception e)
 {
 }

 // --- Start FPS Growth Module
 try
 {
  // --- Execute FPS Growth Module
  p = r.exec("c:/FPS/FPSGrow " + base + " -all");

  try
  {
  // --- Wait for FPS Growth Module to finish
  p.waitFor();
```

```
    status = p.exitValue();
    //System.out.println("MESSAGE: Stand Grown to Year " + base);
    }
    catch (InterruptedException wait)
    {
     System.err.println("*** ERROR *** :" + wait);
    }
   }
   catch(IOException fps)
   {
    System.err.println("*** ERROR *** : " + fps);
   }
   // --- Attempt to finalize and garbage collect
   r.runFinalization();
   r.gc();

   // --- Pause the program before running FPS tools
   try
   {
    long time = System.currentTimeMillis();
    long waitTill = time + 1500;
    while (time < waitTill)
    {
     time = System.currentTimeMillis();
    }
   }
   catch (Exception e)
   {
   }

   // --- Return status
   return status;
  }
  // FINISH: 'FPS_GROWBASE' Method


//=============================================================================
=====
  // 'FPS_CRUS' Method
  //
  // FUNCTION: Executes the FPS CRUS module.
  //

//=============================================================================
=====
  public static int fps_Crus ()
  {
   // --- Declare return variable reporting on status
   int status = 1;

   // --- Verify that DB Connection is closed first
   try
```

```
{
 if (Database.con1.isClosed() == false)
 {
  //System.out.println("MESSAGE: DB Open - Closing DB Connection now...");
  Database.con_no();
 }
 else
 {
  //System.out.println("MESSAGE: DB Verified Closed.");
 }
}
catch (Exception ogiSql)
{
 System.err.println(ogiSql.getMessage());
 ogiSql.printStackTrace();
}

// --- Instantiate Runtime and Process variables
r = Runtime.getRuntime();
p = null;

// --- Pause the program before running FPS tools
try
{
 long time = System.currentTimeMillis();
 long waitTill = time + 2000;
 while (time < waitTill)
 {
  time = System.currentTimeMillis();
 }
}
catch (Exception e)
{
}

// --- Start FPS Crus Module
try
{
 // --- Execute FPS Crus Module
 p = r.exec("c:/FPS/FPSCrus -go");

 try
 {
 // --- Wait for FPS Crus to finish
 p.waitFor();
 status = p.exitValue();
 //System.out.println("MESSAGE: Stand Compiled");
 }
 catch (InterruptedException wait)
 {
  System.err.println("*** ERROR *** :" + wait);
 }
```

```
    }
    catch(IOException fps)
    {
     System.err.println("*** ERROR *** : " + fps);
    }
    // --- Attempt to finalize and garbage collect
    r.runFinalization();
    r.gc();

    // --- Pause the program before running FPS tools
    try
    {
     long time = System.currentTimeMillis();
     long waitTill = time + 2000;
     while (time < waitTill)
     {
      time = System.currentTimeMillis();
     }
    }
    catch (Exception e)
    {
    }

    // --- Return status
    return status;
   }
   // FINISH: 'FPS_CRUS' Method

}
// FINISH: 'FPS' Pubic Class
```

```
package thesis;

/*****************************************************************************
 * CALCULATIONS Class
 *
 * FUNCTION: Contains two large methods, OGI and PNV, to be called for any
 * given 'STAND' number and 'YEAR', then return a value to calling class
 *****************************************************************************/

import java.sql.*;
import java.util.*;
import java.lang.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.text.NumberFormat;
import java.io.*;

public class Calculations {

 //---------------------------------------------------------------------
 // Declare Prepared Statements and Resultsets for OGI and PNV queries
 //---------------------------------------------------------------------

 // --- ECOLOGY Method statements and resultsets
 private PreparedStatement psq4;  private ResultSet rsq4;
 private PreparedStatement psq5;  private ResultSet rsq5;
 private PreparedStatement psq6;  private ResultSet rsq6;
 private PreparedStatement psq7;  private ResultSet rsq7;
 private PreparedStatement psq9;  private ResultSet rsq9;

 // --- APPRAISE Method statements and resultsets
 private PreparedStatement psq1;   private ResultSet rsq1;
 private PreparedStatement psq8;   private ResultSet rsq8;
 private PreparedStatement psq10;  private ResultSet rsq10;
 private PreparedStatement getTpa;  private ResultSet standTpa;
 private PreparedStatement getCutTpa;  private ResultSet standCutTpa;
 //FINISH: Statement and Resultset declarations

 //---------------------------------------------------------------------
 // Declare Query Strings
 //---------------------------------------------------------------------

 private String getStand =
  "SELECT * FROM STAND WHERE (STAND.RPT_YR = ?) AND " +
  "(NOT STAND.VEG_LBL = 'Dead') AND (NOT STAND.VEG_LBL = '.Cut')";

 // --- ECOLOGY Method Query Strings
 private String q4 =
  "SELECT SUM(TREES) AS SumTrees FROM DBHCLS WHERE (GROUP = ? OR GROUP =
?) " +
  "AND RPT_YR = ? AND DBH >= ? AND STD_ID = ? GROUP BY RPT_YR ";
```

```
 private String q5 =
  "SELECT SUM(TREES) AS SumTrees FROM DBHCLS WHERE (GROUP = ? OR GROUP =
?) " +
  "AND RPT_YR = ? AND DBH > ? AND SPECIES = ? AND STD_ID = ? GROUP BY RPT_YR
";
 private String q6 =
  "SELECT COUNT(TREES) AS qty FROM DBHCLS WHERE (GROUP = ? OR GROUP = ?) "
+
  "AND RPT_YR = ? AND DBH > ? AND STD_ID = ?";
 private String q7 =
  "SELECT * FROM DBHCLS WHERE (GROUP = ? OR GROUP = ?) AND RPT_YR = ? AND "
+
  "DBH > ? AND STD_ID = ?";
 private String q9  =
  "SELECT SUM((DBH * TREES)/ ?) AS dbh_all FROM DBHCLS WHERE (GROUP = ? " +
  "OR GROUP = ?) AND RPT_YR = ? AND DBH >= ? AND STD_ID = ? GROUP BY
RPT_YR";

 // --- APPRAISE Method Query Strings
 private String q1 =
  "SELECT RPT_YR, SPECIES, GROUP, SUM(BRD1_DN) AS SUM_BRD1_DN,
SUM(BRD2_DN) " +
  "AS SUM_BRD2_DN, SUM(BRD3_DN) AS SUM_BRD3_DN FROM DBHCLS WHERE
(RPT_YR = ? "+
  "AND GROUP = ? AND STD_ID = ?) GROUP BY RPT_YR, SPECIES, GROUP ";
 private String q8 =
  "SELECT COUNT(SPECIES) AS CNT FROM DBHCLS WHERE (RPT_YR = ? AND GROUP
= ? "
  + "AND STD_ID = ?) GROUP BY SPECIES ";
 private String queryTpa = "SELECT * FROM STAND WHERE (STAND.RPT_YR = ?)" +
   " AND (STAND.STD_ID = ?) AND (NOT STAND.VEG_LBL = 'Dead')  AND" +
   " (NOT STAND.VEG_LBL = '.Cut')";
 private String queryCutTpa = "SELECT * FROM STAND WHERE (STAND.RPT_YR = ?)" +
   " AND (STAND.STD_ID = ?) AND (STAND.VEG_LBL = '.Cut')";
 // --- FINISH: Query String Declarations

 //-----------------------------------------------------------------------
 // Calculated Variables for ECOLOGY and PNV methods
 //-----------------------------------------------------------------------
 private float sd;
 private float calc_sd;
 private float tpa_all;
 private float calc_tpa_all;
 private float tpa_lg;
 private float calc_tpa_lg;
 private float dbh_all;
 private float calc_dbh_all;
 private float ogi1;
 private float ogi2;
 private float ogi3;
 private float ogi4;
 private float ogi;
```

```
private float cut_vol;
private float resid_vol;
private double d_cut_vol;
private double sd_cut_vol;
// FINISH: Calculated Variables declarations

/*****************************************************************************
 * 'CALCULATIONS' Constructor
 *
 * FUNCTION:  Nothing for now
 *****************************************************************************/
public Calculations()
{

}

/*****************************************************************************
 * 'ECOLOGY' Method
 *
 * FUNCTION: Receives a 'Year' and 'Stand', calculates statistics for the
 * given year/stand combination, forumlates an OGI value, and returns this
 * value to the caller.
 *****************************************************************************/
public float ecology(int yr, int stand)
{
 // --- Check Status of DB Connection and open if needed
 try
 {
  if (Database.con1.isClosed() == true)
  {
   //System.out.println("Opening DB Connection...");
   Database.con_go();
  }
  else
  {
   //System.out.println("DB Connection Open - Using Active Port...");
  }
 }
 catch (SQLException ogiSql)
 {
  System.err.println(ogiSql.getMessage());
  System.err.println(ogiSql.getSQLState());
  ogiSql.printStackTrace();
 }

 // --- Setup Prepared Statements
 try
 {
  psq4 = Database.con1.prepareStatement(q4);
  psq5 = Database.con1.prepareStatement(q5);
  psq6 = Database.con1.prepareStatement(q6);
  psq7 = Database.con1.prepareStatement(q7);
```

```
 psq9 = Database.con1.prepareStatement(q9);
}
catch (SQLException ogiSql)
{
 System.err.println(ogiSql.getMessage());
 System.err.println(ogiSql.getSQLState());
 ogiSql.printStackTrace();
}

// --- Initialize instance variables and instantiate for OGI calculations
String grp = ".."; // Specify 'live trees' Group
String grp2 = ".C"; // Specify 'cut' trees group
float dbh_limit = 0; // ** Important: Minimum DBH to limit queries
float SumTrees = 0;
sd = 0;
tpa_all = 0;
tpa_lg = 0;
dbh_all = 0;
int rowcount7 = 0; // row count variable for query 7

// --- Gather OGI variables for Stand/Year passed to method

// -(1)- Calculate density of all trees greater than specified dbh minimum
dbh_limit = (float)(5/2.54); // **Note: The user may want to change this

try
{
 psq4.setString(1, grp);  psq4.setString(2, grp2);
 psq4.setInt(3, yr);  psq4.setFloat(4, dbh_limit); psq4.setInt(5, stand);
}
catch (SQLException ogiSql)
{
 System.err.println(ogiSql.getMessage());
 System.err.println(ogiSql.getSQLState());
 ogiSql.printStackTrace();
}

try
{
 rsq4 = psq4.executeQuery();
 if(rsq4.next())
   tpa_all = rsq4.getFloat("SumTrees"); //TOTAL trees above minimum dbh
 else
   tpa_all = 0;
 rsq4.close();
}
catch (SQLException ogiSql)
{
 System.err.println(ogiSql.getMessage());
 System.err.println(ogiSql.getSQLState());
 ogiSql.printStackTrace();
}
```

```
// -(2)- Calculate density of all DF trees greater than specified dbh limit
dbh_limit = (float)(100/2.54); // **Note: user may want to change this
String species = "DF"; // **Note: user may want to change primary species

try
{
 psq5.setString(1, grp); psq5.setString(2, grp2); psq5.setInt(3, yr);
 psq5.setFloat(4, dbh_limit); psq5.setString(5, species);
 psq5.setInt(6, stand);
}
catch (SQLException ogiSql)
{
 System.err.println(ogiSql.getMessage());
 System.err.println(ogiSql.getSQLState());
 ogiSql.printStackTrace();
}

try
{
 rsq5 = psq5.executeQuery();
 if (rsq5.next() == false)
  tpa_lg = 0; //catches stands with 0 large trees
 else
   tpa_lg = rsq5.getFloat("SumTrees"); //TOTAL large trees species and dbh
 rsq5.close();
}
catch (SQLException ogiSql)
{
 System.err.println(ogiSql.getMessage());
 System.err.println(ogiSql.getSQLState());
 ogiSql.printStackTrace();
}

// -(3)- Calculate mean tree DBH
dbh_limit = 0; // Reset dbh minimum to zero...mean calculated for ALL trees

try
{
 psq4.setString(1, grp); psq4.setString(2, grp2);
 psq4.setInt(3, yr); psq4.setFloat(4, dbh_limit); psq5.setInt(5, stand);
}
catch (SQLException ogiSql)
{
 System.err.println(ogiSql.getMessage());
 System.err.println(ogiSql.getSQLState());
 ogiSql.printStackTrace();
}

try
{
 rsq4 = psq4.executeQuery();
```

```
 if( rsq4.next() )
  SumTrees = rsq4.getFloat("SumTrees"); //TOTAL trees of all sizes
 else
  SumTrees = 0;
 rsq4.close();
}
catch (SQLException ogiSql)
{
 System.err.println(ogiSql.getMessage());
 System.err.println(ogiSql.getSQLState());
 ogiSql.printStackTrace();
}

try
{
 psq9.setFloat(1, SumTrees); psq9.setString(2, grp);
 psq9.setString(3, grp2); psq9.setInt(4, yr);
 psq9.setFloat(5, dbh_limit); psq9.setInt(6, stand);
}
catch (SQLException ogiSql)
{
 System.err.println(ogiSql.getMessage());
 System.err.println(ogiSql.getSQLState());
 ogiSql.printStackTrace();
}

try
{
 rsq9 = psq9.executeQuery();
 if( rsq9.next() )
  dbh_all = (rsq9.getFloat("dbh_all"));// MEAN DBH of ALL trees
 else
  dbh_all = 0;
 rsq9.close();
}
catch (SQLException ogiSql)
{
 System.err.println(ogiSql.getMessage());
 System.err.println(ogiSql.getSQLState());
 ogiSql.printStackTrace();
}

// -(4)- Newer Version of SD calculation
try
{
 psq6.setString(1, grp); psq6.setString(2, grp2);
 psq6.setInt(3, yr); psq6.setFloat(4, dbh_limit); psq6.setInt(5, stand);
}
catch (SQLException ogiSql)
{
 System.err.println(ogiSql.getMessage());
 System.err.println(ogiSql.getSQLState());
```

```
 ogiSql.printStackTrace();
}

try
{
 psq7.setString(1, grp); psq7.setString(2, grp2);
 psq7.setInt(3, yr); psq7.setFloat(4, dbh_limit); psq7.setInt(5, stand);
}
catch (SQLException ogiSql)
{
 System.err.println(ogiSql.getMessage());
 System.err.println(ogiSql.getSQLState());
 ogiSql.printStackTrace();
}

try
{
 rsq6 = psq6.executeQuery();
 if( rsq6.next() )
  rowcount7 = rsq6.getInt("qty"); // Number of dbh classes total...rows
 else rowcount7 = 0;
 rsq6.close();
}
catch (SQLException ogiSql)
{
 System.err.println(ogiSql.getMessage());
 System.err.println(ogiSql.getSQLState());
 ogiSql.printStackTrace();
}

try
{
 rsq7 = psq7.executeQuery(); rsq7.next();
}
catch (SQLException ogiSql)
{
 System.err.println(ogiSql.getMessage());
 System.err.println(ogiSql.getSQLState());
 ogiSql.printStackTrace();
}

// Loop to process new version of SD calculation
int start7 = 1;
float runsum = 0; float sumdiffx = 0;
float diffx = 0; float sqrdiffx = 0;
float xbar = dbh_all;

while (start7 <= rowcount7)
{
 try
 {
  diffx = (rsq7.getFloat("DBH") - xbar);
```

```
   sqrdiffx = (float)((Math.pow(diffx, 2))*((rsq7.getFloat("TREES"))*100));
   runsum = runsum + sqrdiffx;
   start7++;
   rsq7.next();
  }
  catch (SQLException ogiSql)
  {
   System.err.println(ogiSql.getMessage());
   System.err.println(ogiSql.getSQLState());
   ogiSql.printStackTrace();
  }
}// End new SD calculation loop

sd = (float)(Math.sqrt(runsum/((SumTrees*100)-1))); // NEW SD for TOTAL

// Set Calculated variable values before modifying due to OGI constraints
calc_sd = sd;
calc_tpa_all = tpa_all;
calc_tpa_lg = tpa_lg;
calc_dbh_all = dbh_all;

// Check to see if 4 variables are within constraints of published values
if (sd < Inputs.getSd_dbh_young()) sd = Inputs.getSd_dbh_young();
if (tpa_lg < Inputs.getTpa_lg_young()) tpa_lg = Inputs.getTpa_lg_young();
if (dbh_all < Inputs.getDbh_young()) dbh_all = Inputs.getDbh_young();
if (tpa_all > Inputs.getTpa_all_young()) tpa_all = Inputs.getTpa_all_young();
if (sd > Inputs.getSd_dbh_old()) sd = Inputs.getSd_dbh_old();
if (tpa_lg > Inputs.getTpa_lg_old()) tpa_lg = Inputs.getTpa_lg_old();
if (dbh_all > Inputs.getDbh_old()) dbh_all = Inputs.getDbh_old();
if (tpa_all < Inputs.getTpa_all_old()) tpa_all = Inputs.getTpa_all_old();

//Calculate Total OGI
ogi1 = Math.abs((sd - Inputs.getSd_dbh_young())/
  (Inputs.getSd_dbh_old() - Inputs.getSd_dbh_young()));
ogi2 = Math.abs((tpa_lg - Inputs.getTpa_lg_young())/
  (Inputs.getTpa_lg_old() - Inputs.getTpa_lg_young()));
ogi3 = Math.abs((dbh_all - Inputs.getDbh_young())/
  (Inputs.getDbh_old() - Inputs.getDbh_young()));
// --- Handle reverse tpa_all due to unusually low tpa
if (calc_tpa_all >= Inputs.getTpaInflection())
{
  ogi4 = Math.abs((tpa_all - Inputs.getTpa_all_young())/
    (Inputs.getTpa_all_old() - Inputs.getTpa_all_young()));
}
else
{
  ogi4 = Math.abs((calc_tpa_all)/
    (Inputs.getTpaInflection()));
}
ogi = 25 * (ogi1 + ogi2 + ogi3 + ogi4);

// Close Statements and Database Connection
```

```
try
{
 psq4.close();
 psq5.close();
 psq6.close();
 rsq7.close();
 psq7.close();
 psq9.close();
 //Database.con_no();
}
catch (SQLException close)
{
 System.out.println(close.getMessage());
 close.printStackTrace();
}
return ogi;
}
//FINISH: 'ECOLOGY' Method


/*****************************************************************************
 * 'PRINTECOLOGY' Method
 *
 * FUNCTION: Prints out all calculated variables from an Ecology method
 * call.  Could further be extended to send the information to an ASCII
 * file.
 *****************************************************************************/

public void printEcology()
{
 // --- Obtain and format a numberFormat for output
 NumberFormat nf = NumberFormat.getNumberInstance();
 nf.setMaximumFractionDigits(2);
 nf.setMinimumFractionDigits(2);

 // --- Print 4 calculated structural variables for stand
 System.out.println("CALCULATED OGI Variables for this Stand/Year:");
 System.out.println("SD : " + nf.format(calc_sd) + " TPA ALL : " +
  nf.format(calc_tpa_all) + " TPA LG : " + nf.format(calc_tpa_lg) +
  " AVG DBH : " + nf.format(calc_dbh_all));

 // --- Print 4 constrained structural variables for stand
 System.out.println("CONSTRAINED OGI Variables for this Stand/Year:");
 System.out.println("SD : " + nf.format(sd) + " TPA ALL : " +
  nf.format(tpa_all) + " TPA LG : " + nf.format(tpa_lg) +
  " AVG DBH : " + nf.format(dbh_all));
}
// FINISH: 'PrintEcology' Method

public float getSD ()
{
 return calc_sd;
```

```
}

public float getTpaAll ()
{
 return calc_tpa_all;
}

public float getTpaLg ()
{
 return calc_tpa_lg;
}

public float getDbh ()
{
 return calc_dbh_all;
}

//#############################################################################

/*****************************************************************************
 * 'APPRAISE' Method
 *
 * FUNCTION: Method that accepts a STAND and YEAR and returns a value
 * for the PNV of that STAND/YEAR combination
 *
 * NOTE: PNV is calculated assuming value of final harvest at passed year
 * plus value of all previous thinnings, discounted to the base year.
 *****************************************************************************/

public float appraise(int yr, int standpnv)
{

 int rows1 = 0; // Variable to tally number species found in RS#8

 //--- Load array to hold the FPS report YEARS array from INPUTS class
 int [] yr_all = new int [20];
 yr_all = Inputs.getYr_all(); // Obtain global array of report years

 //--- Load array to hold the Species Codes for log prices from INPUTS class
 String [] price_spc = new String [10];
 price_spc = Inputs.getPrice_spc();

 //--- Load array (2-D) to hold log prices for 3 sorts per species
 int [] [] price_sorts = new int [10] [3];
 price_sorts = Inputs.getPrice_sorts();

 // --- Check Status of DB Connection and open if needed
 try
 {
  if (Database.con1.isClosed() == true)
  {
```

```
    //System.out.println("Opening Database Connection...");
    Database.con_go();
   }
   else
   {
    //System.out.println("DB Connection Open - Using Active Port...");
   }
  }
  catch (SQLException ogiSql)
  {
   System.err.println(ogiSql.getMessage());
   System.err.println(ogiSql.getSQLState());
   ogiSql.printStackTrace();
  }

  //--- Load Prepared SQL Statements
  try
  {
   psq1 = Database.con1.prepareStatement(q1);
   psq8 = Database.con1.prepareStatement(q8);
  }
  catch (SQLException ps_go)
  {
   System.out.println("*** ERROR *** : 'Appraise' Prepared Statements");
   System.err.println(ps_go.getMessage());
   System.err.println(ps_go.getSQLState());
   ps_go.printStackTrace();
  }

//=============================================================================
===
  // Set the year passed to method as the year to C.C. all stems and then
  // iterate to find all previous thinning years

//=============================================================================
===

  int yr_cnt = 0; // Counter for the point in the 'yr' array
  boolean cc = false; // Variable to identify if clear-cut year or not
  int lenyr_all = yr_all.length; // Length of 'yr_all' array
  float compval = 0; // Value of PNV for year passed to be returned

  /*
  PrintWriter outputStream1 = null;
  try
  {
   outputStream1 = new PrintWriter( new FileOutputStream("Opti_cost.txt", true));
  }
  catch(FileNotFoundException fnf)
  {
   System.out.println("*** ERROR *** : Output File Not Found");
```

```
  System.exit(0);
 }
 */


//=============================================================================
===
 // Start to analyze the PNV for all previous years and the year passed to
 // the 'Appraise' method.

//=============================================================================
===

 for (int d = 0; d <= lenyr_all; d++)
 {
  if (yr_all[yr_cnt] == yr) cc = true;
   else cc = false; // Identifies if a clear-cut year or not

  String [] grp = {".C", ".."}; // Two FPS 'Groups' for sorting on
  int a = 0; // Initialize counter for 'grp' array
  float totalval = 0; // Initialize PNV accumulator for THIS year's run
  int len_grp = grp.length; //length of 'grp' array - 2 elements (3 if '.D')
  cut_vol = 0; // Initialize cut volume accumulator for this year's run
  resid_vol = 0; // Initialize residual volume accumulator
  float tpa_cut = 0; // Initialize cut TPA for given period
  float tpa_tot = 0; // Initialize residual TPA for given period
  float loggingcost = 0; // Initialize logging cost for given period
  float annualcost = 0; // Initialize compounded annual cost for period
  float time = 0; // Initialize years for discount formula



  //-------------------------------------------------------------------
  // Start loop for EACH FPS 'Group' at given year
  //-------------------------------------------------------------------
  while (a < len_grp)
  {
   //-----------------------------------------------------------------
   // Set parameters for PS#8 for specific stand, year, and group
   //-----------------------------------------------------------------
   try
   {
    psq8.setInt(1, yr_all[yr_cnt]); // Set 'Year' of report
    psq8.setString(2, grp[a]); // Set 'Group' code
    psq8.setInt(3, standpnv); // Set 'Stand ID'
    //System.out.println("Data: " +yr_all[yr_cnt] + " " + grp[a] + " " + standpnv);
   }
   catch (SQLException ogiSql)
   {
    System.err.println(ogiSql.getMessage());
    System.err.println(ogiSql.getSQLState());
    ogiSql.printStackTrace();
```

```
}

//---------------------------------------------------------------------
// Execute PS#8 on FPS DBHCLS to get the Count of species found
// for given stand, year, and group.
//
// RS#8 should return a resultset containing a single row for each
// species found, listing SPECIES code.
//---------------------------------------------------------------------
try
{
 rsq8 = psq8.executeQuery();
 rows1 = 0;
 while (rsq8.next())
 {
  rows1 = rows1 + 1;
 }
 //System.out.println("There are rows :" + rows1);
}
catch (SQLException ogiSql)
{
 System.err.println(ogiSql.getMessage());
 System.err.println(ogiSql.getSQLState());
 ogiSql.printStackTrace();
}

//---------------------------------------------------------------------
// Define arrays to hold Species/Volume summary for this run
//---------------------------------------------------------------------
String [] species = new String [rows1];
float [] [] volumes = new float [rows1] [3];

//---------------------------------------------------------------------
// Set parameters for PS#1 for specific stand, year, and group
//---------------------------------------------------------------------
try
{
 psq1.setInt(1, yr_all[yr_cnt]); // Set 'Year' of report
 psq1.setString(2, grp[a]); // Set 'Group' code
 psq1.setInt(3, standpnv); // Set 'Stand ID'
}
catch (SQLException ogiSql)
{
 System.err.println(ogiSql.getMessage());
 System.err.println(ogiSql.getSQLState());
 ogiSql.printStackTrace();
}

//---------------------------------------------------------------------
// Execute PS#1 on FPS DBHCLS to get volume by species summary
// for given stand, year, and group.
//
```

```
// RS#1 returns a resultset containing several rows.  Each row is
// a summary of the three volume sorts for each SPECIES of the
// year and group specified.  Use a for loop to populate the 'species'
// and 'volumes' array with their members for this year and group
// using 'rows1' as the number of times to loop.
//-----------------------------------------------------------------
try
{
 rsq1 = psq1.executeQuery();
 int z = 0;
 while(rsq1.next())
 {
  species [z] = rsq1.getString("SPECIES");
  volumes [z] [0] = (float)rsq1.getDouble("SUM_BRD1_DN");
  volumes [z] [1] = (float)rsq1.getDouble("SUM_BRD2_DN");
  volumes [z] [2] = (float)rsq1.getDouble("SUM_BRD3_DN");
  z ++;
 }
}
catch (SQLException ogiSql)
{
 System.err.println(ogiSql.getMessage());
 System.err.println(ogiSql.getSQLState());
 ogiSql.printStackTrace();
}

//-----------------------------------------------------------------
// Start a loop that sorts volumes and obtains log prices
//-----------------------------------------------------------------
for (int b = 0; b < rows1; b++)
{
 float vol_1 = 0;
 float vol_2 = 0;
 float vol_3 = 0;
 String spc = null;
 float prc_1 = 0;
 float prc_2 = 0;
 float prc_3 = 0;

 //--- For each Species (i.e. row1), get SPECIES code then do volumes
 spc = species[b];

 //--- Assign total bdft volume for all three sorts for this species
 vol_1 = volumes[b][0];
 vol_2 = volumes[b][1];
 vol_3 = volumes[b][2];

 //-----------------------------------------------------------------
 // SET LOG PRICES for this species
 //
 // Loop thru the 'price_spc' array incrementaly until queried species
 // match is found and assign the three log sort prices from the
```

```
// 'price_sorts' array.  If no match, defaults to last set of prices
// in the array.
//-----------------------------------------------------------------
try
{
 for(int x = 0; x < price_spc.length; x ++)
 {
  if(spc.equalsIgnoreCase(price_spc[x]))
  {
   prc_1 = (float)price_sorts[x][0];
   prc_2 = (float)price_sorts[x][1];
   prc_3 = (float)price_sorts[x][2];
   break;
  }
  else
  {
   prc_1 = (float)price_sorts[price_spc.length - 1][0];
   prc_2 = (float)price_sorts[price_spc.length - 1][1];
   prc_3 = (float)price_sorts[price_spc.length - 1][2];
  }
 }
}
catch(Exception prices)
{
 System.err.println(prices.getMessage());
 prices.printStackTrace();
}
// --- End species/log price matching

//--- Assign Logging Costs - TBA!!!!

//--- Calculate Total Net Log Return
float netval = ((vol_1 * prc_1) + (vol_2 * prc_2) +
 (vol_3 * prc_3))/(float)1000;

float npv_t = 0;

//--- Assess time period for discounting
time = (float)yr_all[yr_cnt] - (float)(Inputs.getBase());

//--- Calculate Discounted Net Value
if (time < 1)
{
 npv_t = netval;
}
else
{
 npv_t = netval/(float)Math.pow((1 + Inputs.getRate()), time);
}
totalval = totalval + npv_t; // Markup total value

if(grp[a] == "..") resid_vol = resid_vol + vol_1 + vol_2 + vol_3;
```

```
        else cut_vol = cut_vol + vol_1 + vol_2 + vol_3;

        //System.out.println("group: " + grp[a] + "  species " + spc + "  year " + yr_all[yr_cnt]);
        //System.out.println("vol sort1: " + vol_1 + " vol sort3 " + vol_3 + " price 1 " + prc_1 + "
price 3 " + prc_3 );
        //System.out.println("npv for that species and group " + npv_t);
        //System.out.println("running totalval for that year " + totalval);
       }
       //End for loop for species in a group

       //--- Increment 'group' counter
       a++;

       // Break loop if only a thinning period
       if (cc == false)
       {
        break;
       }
      }
      // End while statement for all 'groups'

      //---------------------------------------------------------------------
      // Subtract logging costs for this period.  Apply only to '.C' volume
      // for a non-'CC' year, or all volume if a 'CC' year.
      //---------------------------------------------------------------------

      // --- 1st get total TPA for given period in case a 'CC' year
      try
      {
       getTpa = Database.con1.prepareStatement(queryTpa);
       getTpa.setShort(1, (short)yr_all[yr_cnt]);
       getTpa.setInt(2, standpnv);
       standTpa = getTpa.executeQuery();
       if(standTpa.next())
        tpa_tot = (float)standTpa.getDouble("STEMS");
       else tpa_tot = (float)0.00;
      }
      catch (SQLException ogiSql)
      {
       System.err.println(ogiSql.getMessage());
       System.err.println(ogiSql.getSQLState());
       ogiSql.printStackTrace();
      }

      // --- Close Statement and Resultset
      try
      {
       getTpa.close();
       standTpa.close();
      }
      catch (SQLException ogiSql)
      {
```

```
 System.err.println(ogiSql.getMessage());
 System.err.println(ogiSql.getSQLState());
 ogiSql.printStackTrace();
}

//--- 2nd, get TPA of trees cut this period in case a thinning only
try
{
 getCutTpa = Database.con1.prepareStatement(queryCutTpa);
 getCutTpa.setShort(1, (short)yr_all[yr_cnt]);
 getCutTpa.setInt(2, standpnv);
 standCutTpa = getCutTpa.executeQuery();
 if(standCutTpa.next())
   tpa_cut = (float)standCutTpa.getDouble("STEMS");
 else tpa_cut = (float)0.00;
}
catch (SQLException ogiSql)
{
 System.err.println(ogiSql.getMessage());
 System.err.println(ogiSql.getSQLState());
 ogiSql.printStackTrace();
}

// --- Close Statement and Resultset
try
{
 getCutTpa.close();
 standCutTpa.close();
 //Database.con1.close();
 //System.out.println("MESSAGE: Obtained current CUT TPA.");
}
catch (SQLException ogiSql)
{
 System.err.println(ogiSql.getMessage());
 System.err.println(ogiSql.getSQLState());
 ogiSql.printStackTrace();
}



// --- 3rd, calculate logging costs (check for 'CC' or 'thin only')
if (cc == false)
{
 // --- Process cost if a commercially viable operation
 if (cut_vol > Inputs.getBdftMin() )
 {
  d_cut_vol = (double)cut_vol;
  sd_cut_vol = Inputs.getCoeffB() * Math.sqrt(d_cut_vol);
  // --- Calculate $/BF logging cost
  loggingcost = (float)(Inputs.getIntercept() + (Inputs.getCoeffA() *
    tpa_cut) + sd_cut_vol + Inputs.getHaulCost());
  // --- Multiply by volume harvested
```

```
    loggingcost = (loggingcost * (cut_vol / 1000));
    // --- Discount to base year
    if (time < 1)
    {
     loggingcost = loggingcost;
    }
    else
    {
     loggingcost = loggingcost/(float)Math.pow((1 + Inputs.getRate()), time);
    }
    // --- Subtract from totalVal
    totalval = totalval - loggingcost;
   }

   // --- Process cost if a pre-commercial operation
   else
   {
    // --- Zero out Revenue in 'totalval' due to non-commercial limit
    totalval = 0;
    // --- Calculate $/Acre logging cost
    loggingcost = (float)(Inputs.getIntercept_p() + (Inputs.getCoeffA_p() *
     tpa_cut));
    if (tpa_cut < 1)
     loggingcost = 0;
    // --- Discount to base year
    if (time < 1)
    {
     loggingcost = loggingcost;
    }
    else
    {
     loggingcost = loggingcost/(float)Math.pow((1 + Inputs.getRate()), time);
    }
    // --- Subtract from totalVal
    totalval = totalval - loggingcost;
   }
  }// --- END 'if' clause for non-'CC' period logging cost calculation

  // --- Calculate costs as if a 'CC' period (use both cut and resid vol)
  else
  {
   // *** Get cost based on cut_vol plus resid_vol at this period, discount,
   // and subtract from totalval.  Don't forget hauling cost.
   // --- Process cost if a commercially viable operation
   if ((cut_vol + resid_vol) > Inputs.getBdftMin() )
   {
    d_cut_vol = (double)(cut_vol + resid_vol);
    sd_cut_vol = Inputs.getCoeffB() * Math.sqrt(d_cut_vol);
    // --- Calculate $/BF logging cost
    loggingcost = (float)(Inputs.getIntercept() + (Inputs.getCoeffA() *
     (tpa_tot)) + sd_cut_vol + Inputs.getHaulCost());
    // --- Multiply by volume harvested
```

```
        loggingcost = loggingcost * ((cut_vol + resid_vol) / 1000);
        // --- Discount to base year
        if (time < 1)
        {
         loggingcost = loggingcost;
        }
        else
        {
         loggingcost = loggingcost/(float)Math.pow((1 + Inputs.getRate()), time);
        }
        // --- Subtract from totalVal
        totalval = totalval - loggingcost;
       }

       // --- Process cost if a pre-commercial operation
       else
       {
        // --- Zero out Revenue in 'totalval' due to non-commercial limit
        totalval = 0;
        // --- Calculate $/Acre logging cost
        loggingcost = (float)(Inputs.getIntercept_p() + (Inputs.getCoeffA_p() *
         (tpa_tot)));
        // --- Discount to base year
        if (time < 1)
        {
         loggingcost = loggingcost;
        }
        else
        {
         loggingcost = loggingcost/(float)Math.pow((1 + Inputs.getRate()), time);
        }
        // --- Subtract from totalVal
        totalval = totalval - loggingcost;
       }
      }// --- End'else' clause for 'CC' operation

      //--- Add this years total to next years up to passed year
      compval = compval + totalval;

      //outputStream1.println("the total value for year " + yr_all[yr_cnt] + " is " + totalval + "
logging= " + loggingcost);
      //System.out.println("the running compval so far is " + compval);

      //--- Stop global loop if "cc" year and return final value
      if (cc == true)
      {
       break;
      }


      //--- Increment to next year otherwise
      yr_cnt ++;
```

```
}
// End of global for loop

//System.out.println("The Year: " + yr + " The PNV if C.C. now: " + compval);

//--- Close Statements and Database
try
{
 rsq1.close();
 rsq8.close();
 psq1.close();
 psq8.close();
 //Database.con_no();
}
catch (SQLException ogiSql)
{
 System.err.println(ogiSql.getMessage());
 System.err.println(ogiSql.getSQLState());
 ogiSql.printStackTrace();
}

//-----------------------------------------------------------------------
// *** This is also where you would evaluate PNV vs LEV in terms of a
// start-up cost, fixed treatment costs along the way, and most
// importantly, perpetual subsequent rotations or an opportunity cost.
//-----------------------------------------------------------------------

// --- Apply annual cost formula and subtract from 'compval'
float pv_annualCost = Inputs.getAnnualCost() *
 (((float)Math.pow((1 + (double)Inputs.getRate()),
 (double)(yr - Inputs.getBase())) - 1) /
 ((float)Math.pow((1 + (double)Inputs.getRate()),
  (double)(yr - Inputs.getBase())) * Inputs.getRate()));
//outputStream1.println("Annual cost " + pv_annualCost);

// --- Subtract discounted annual cost and startup cost
compval = compval - pv_annualCost - (float)Inputs.getPdZeroCost();
//outputStream1.println("dRev-dcost-dannualcost" + compval);

//-----------------------------------------------------------------------
// Determine which Econ Method was selected by user. Econ 1 implies the
// stand is just planted and the current year minus the base year is
// a perpetual rotation age. Compound 'compval' to end of 1st rotation
// and use perpetual period series formula.
// Econ 2 implies an existing stand. The user should hopefully have
// supplied a 'resid_value' which represents either the resale value
// or LEV of bare land or estimated LEV of subsequent rotations (all the
// same thing). This value is used as a revenue at the end year and
// discounted back with the final revenues.
//-----------------------------------------------------------------------
if (Inputs.getEconMethod() == 2)
{
```

```
    compval = compval + ((float)Inputs.getResidValue()/(float)Math.pow((1 + Inputs.getRate()),
      (yr - Inputs.getBase())));
   }

   // --- Assume method 1
   else
   {
    compval = (compval * (float)Math.pow((1 + Inputs.getRate()), (yr - Inputs.getBase()))) /
      ((float)Math.pow((1 + Inputs.getRate()), (yr - Inputs.getBase())) - 1);
   }

   // --- Return the total pnv value for the year requested
   //outputStream1.close();
   return compval;


  }
  // FINISH: 'APPRAISE' Method
//=============================================================================
=======

  public float getCut_vol ()
  {
   return cut_vol;
  }

  public float getResid_vol ()
  {
   return resid_vol;
  }

}
// FINISH: 'Calculations' Class
```

```
package thesis;

import java.text.NumberFormat;
import java.io.*;

/*****************************************************************************
 * 'Scenario' Class
 *
 * FUNCTION:  This class is an object that will be instantiated numerous times
 * and held in an array of 'Scenario' objects.  Each 'Scenario' represents a
 * single trajectory of prescriptions along a path.  It has variables to hold
 * critical statistics regarding the character of the stand and resulting
 * OGI and PNV values at each Report Period.
 *****************************************************************************/

public class Scenario
{

  private int [] years; // Report years
  private int [] rxNumber; // The Scenario number for this period
  private boolean [] harvest; // True or false for harvest activity in period
  private float [][] ogi_variables; // The calculated 4 OGI statistics by period
  private int [][] rx_variables; // The FPS Silvics table parameters by period
  private float [] bdft_cut; // BDFT cut, if any, by period
  private float [] bdft_resid; // BDFT residual (or total) by period
  private float [] tpa_Cut; // TPA cut (all dbh's)
  private float [] tpa_postCut; // TPA just after cutting (all dbh's, live)
  private float [] ogi; // The OGI value by period
  private float [] pnv; // The PNV value by period


//=============================================================================
=====
  // 'Scenario' Constructor
  //
  // FUNCTION: Instantiates all variables to zero values when object created.
  // Creates a static sized array able to hold up to 20 periods of data each.

//=============================================================================
=====
  public Scenario()
  {
    years = new int [20];
    rxNumber = new int [20];
    harvest = new boolean [20];
    ogi_variables = new float [20][4];
    rx_variables = new int [20][3];
    bdft_cut = new float [20];
    bdft_resid = new float [20];
    tpa_Cut = new float [20];
    tpa_postCut = new float [20];
    ogi = new float [20];
```

```
 pnv = new float [20];
}
// FINISH: 'Scenario' Constructor

public void setYears (int passedPeriod, int passedYear)
{
 years [passedPeriod] = passedYear;
}

public void setRXnumber (int passedPeriod, int passedScenarioNumber)
{
 rxNumber [passedPeriod] = passedScenarioNumber;
}

public void setHarvest (int passedPeriod, boolean passedHarvest)
{
 harvest [passedPeriod] = passedHarvest;
}

public void setOGIvariables (int passedPeriod, float sd, float tpa_all,
 float tpa_lg, float dbh_all)
{
 ogi_variables [passedPeriod] [0] = sd;
 ogi_variables [passedPeriod] [1] = tpa_all;
 ogi_variables [passedPeriod] [2] = tpa_lg;
 ogi_variables [passedPeriod] [3] = dbh_all;
}

public void setRXvariables (int passedPeriod, int method, int mv, int level)
{
 rx_variables [passedPeriod] [0] = method;
 rx_variables [passedPeriod] [1] = mv;
 rx_variables [passedPeriod] [2] = level;
}

public void setBDFT_cut (int passedPeriod, float passedBDFT_cut)
{
 bdft_cut [passedPeriod] = passedBDFT_cut;
}

public void setBDFT_resid (int passedPeriod, float passedBDFT_resid)
{
 bdft_resid [passedPeriod] = passedBDFT_resid;
}

public void setTPA_Cut (int passedPeriod, float passedTPA_Cut)
{
 tpa_Cut [passedPeriod] = passedTPA_Cut;
}

public void setTPA_postCut (int passedPeriod, float passedTPA_postCut)
{
```

```
 tpa_postCut [passedPeriod] = passedTPA_postCut;
}

public void setOGI (int passedPeriod, float passedOGI)
{
 ogi [passedPeriod] = passedOGI;
}

public void setPNV (int passedPeriod, float passedPNV)
{
 pnv [passedPeriod] = passedPNV;
}

public int getRxNumber (int passedPeriod)
{
 return rxNumber [passedPeriod];
}

public float getOgi (int passedPeriod)
{
 return ogi[passedPeriod];
}

public float getPnv (int passedPeriod)
{
 return pnv[passedPeriod];
}

public int [][] getRx_variables ()
{
 return rx_variables;
}

public boolean [] getHarvest ()
{
 return harvest;
}

public float getTpa_postCut (int passedPeriod)
{
 return tpa_postCut[passedPeriod];
}

public void printTo (int passedPeriod)
{
 // --- Obtain and format a numberFormat for output
 NumberFormat nf = NumberFormat.getNumberInstance();
 nf.setMaximumFractionDigits(2);
 nf.setMinimumFractionDigits(2);

 System.out.println(
 "-----------------------------------------------------------------");
```

```
System.out.println(
"SCENARIO: " + rxNumber[passedPeriod]);
System.out.println(
"PERIOD: " + passedPeriod + "\tYEAR: " + years[passedPeriod] +
"\t\tHARVEST? : " + harvest[passedPeriod] +
"\tTPA CUT: " + nf.format((tpa_Cut[passedPeriod])));
System.out.println(
"PRESCRIPTION => " + "\tMETHOD: " + rx_variables[passedPeriod][0] +
"\tMV: " + rx_variables[passedPeriod][1] + "\tLEVEL: " +
rx_variables[passedPeriod][2]);
System.out.println(
"TPA pre-CUT: \t" + nf.format(tpa_Cut[passedPeriod] +
tpa_postCut[passedPeriod]) + "\tTPA post-cut: \t" +
nf.format(tpa_postCut[passedPeriod]));
System.out.println(
"BDFT CUT: \t" + nf.format(bdft_cut[passedPeriod]) + "\tBDFT RESID.: \t" +
nf.format(bdft_resid[passedPeriod]));
System.out.println(
"OGI VALUES => " + "SD: " + nf.format(ogi_variables[passedPeriod][0]) +
 "\tTPA_ALL: " + nf.format(ogi_variables[passedPeriod][1]) + "\tTPA_LG: " +
nf.format(ogi_variables[passedPeriod][2]) + "\tDBH: " +
nf.format(ogi_variables[passedPeriod][3]));
System.out.println(
"OGI :" + nf.format(ogi[passedPeriod]) + "\tPNV: $" +
nf.format(pnv[passedPeriod]));
System.out.println(
"-------------------------------------------------------------------------");
}
// FINISH: 'PrintTo' Method

public void printFile (int passedPeriod)
{
 // --- Obtain and format a numberFormat for output
 NumberFormat nf = NumberFormat.getNumberInstance();
 nf.setMaximumFractionDigits(2);
 nf.setMinimumFractionDigits(2);

 PrintWriter outputStream = null;

 try
 {
  //outputStream = new PrintWriter( new FileOutputStream("OGI_" +
  //  Inputs.getMin_ogi() + "_ENDYEAR_" + Inputs.getFinal_year() + ".txt" , true));
  outputStream = new PrintWriter( new FileOutputStream("Opti_out.txt" , true));
 }
 catch(FileNotFoundException fnf)
 {
  System.out.println("*** ERROR *** : Output File Not Found");
  System.exit(0);
 }
 outputStream.println("MESSAGE: The Detailed Winning Rx is ===> " );
```

```
    outputStream.println(
    "------------------------------------------------------------------------");
    outputStream.println(
    "PERIOD: " + passedPeriod + "\tYEAR: " + years[passedPeriod] +
    "\t\tHARVEST? : " + harvest[passedPeriod] +
    "\tTPA CUT: " + nf.format((tpa_Cut[passedPeriod])));
    outputStream.println(
    "PRESCRIPTION => " + "\tMETHOD: " + rx_variables[passedPeriod][0] +
    "\tMV: " + rx_variables[passedPeriod][1] + "\tLEVEL: " +
    rx_variables[passedPeriod][2]);
    outputStream.println(
    "TPA pre-CUT: \t" + nf.format(tpa_Cut[passedPeriod] +
    tpa_postCut[passedPeriod]) + "\tTPA post-cut: \t" +
    nf.format(tpa_postCut[passedPeriod]));
    outputStream.println(
    "BDFT CUT: \t" + nf.format(bdft_cut[passedPeriod]) + "\tBDFT RESID.: \t" +
    nf.format(bdft_resid[passedPeriod]));
    outputStream.println(
    "OGI VALUES => " + "SD: " + nf.format(ogi_variables[passedPeriod][0]) +
     "\tTPA_ALL: " + nf.format(ogi_variables[passedPeriod][1]) + "\tTPA_LG: " +
    nf.format(ogi_variables[passedPeriod][2]) + "\tDBH: " +
    nf.format(ogi_variables[passedPeriod][3]));
    outputStream.println(
    "OGI :" + nf.format(ogi[passedPeriod]) + "\tPNV: $" +
    nf.format(pnv[passedPeriod]));
    outputStream.println(
    "------------------------------------------------------------------------");

    outputStream.close();
   }
   // FINISH: 'PrintTo' Method

}
// FINSIH: 'Scenario' public class
```

```
package thesis;

/*****************************************************************************
 * 'SILVICS' Class
 *
 *  FUNCTION: This class serves as a container class for methods to
 *  interact with the FPS database tables.
 *****************************************************************************/

import java.sql.*;

public class Silvics
{

 //---------------------------------------------------------------------
 // Declare Prepared Statements and Resultsets for queries
 //---------------------------------------------------------------------

 private PreparedStatement clrSilvics;
 private PreparedStatement clrStand;
 private PreparedStatement clrDbhcls;
 private PreparedStatement setFlag;
 private PreparedStatement getTpa;  private ResultSet standTpa;
 private PreparedStatement getCutTpa;  private ResultSet standCutTpa;
 //---------------------------------------------------------------------
 // Declare Query Strings
 //---------------------------------------------------------------------

 private String delSilvics = "DELETE SILVICS.* FROM SILVICS";
 private String delStand = "DELETE STAND.* FROM STAND";
 private String delDbhcls = "DELETE DBHCLS.* FROM DBHCLS";
 private String queryTpa = "SELECT * FROM STAND WHERE (STAND.RPT_YR = ?)" +
  " AND (STAND.STD_ID = ?) AND (NOT STAND.VEG_LBL = 'Dead')  AND" +
  " (NOT STAND.VEG_LBL = '.Cut')";
 private String queryCutTpa = "SELECT * FROM STAND WHERE (STAND.RPT_YR = ?)" +
  " AND (STAND.STD_ID = ?) AND (STAND.VEG_LBL = '.Cut')";
 private String flagAdmin1 = "UPDATE ADMIN SET FLAG = 0";
 private String flagAdmin2 = "UPDATE ADMIN SET FLAG = 1 WHERE (ADMIN.STD_ID =
?)";


//=============================================================================
=====
 // 'SILVICS' Constructor
 //
 // FUNCTION:

//=============================================================================
=====
 public Silvics()
 {
```

```
}
// FINISH: 'SILVICS' Contstructor

//=============================================================================
=====
// 'clrSilvics' Method
//
// FUNCTION: Clears the FPS Silvics table of all data

//=============================================================================
=====
public int clrSilvics ()
{
 int status = 1; // Dummy variable to return to calling class

 // --- Check Status of DB Connection and open if needed
 try
 {
  if (Database.con1.isClosed() == true)
  {
   //System.out.println("MESSAGE: Opening DB Connection...");
   Database.con_go();
  }
  else
  {
   //System.out.println("MESSAGE: DB Connection Open - Using Active Port...");
  }
 }
 catch (SQLException ogiSql)
 {
  System.err.println(ogiSql.getMessage());
  System.err.println(ogiSql.getSQLState());
  ogiSql.printStackTrace();
 }

 // --- Set Query to clear FPS Silvics table and execute
 try
 {
  clrSilvics = Database.con1.prepareStatement(delSilvics);
  clrSilvics.execute();
 }
 catch (SQLException ogiSql)
 {
  System.err.println(ogiSql.getMessage());
  System.err.println(ogiSql.getSQLState());
  ogiSql.printStackTrace();
 }

 // --- Close Statement and DB Connection
 try
 {
```

```
    clrSilvics.close();
    //Database.con1.close();
    //System.out.println("MESSAGE: SILVICS Table Cleared.");
   }
   catch (SQLException ogiSql)
   {
    System.err.println(ogiSql.getMessage());
    System.err.println(ogiSql.getSQLState());
    ogiSql.printStackTrace();
   }

   return status;
  }
  // FINISH: 'clrSilvics' Method


//=============================================================================
=====
  // 'setFlags' Method
  //
  // FUNCTION: Sets all stands in ADMIN to '0' flag and flags '1' for subject.

//=============================================================================
=====
  public int setFlags ()
  {
   int status = 1; // Dummy variable to return to calling class

   // --- Check Status of DB Connection and open if needed
   try
   {
    if (Database.con1.isClosed() == true)
    {
     //System.out.println("MESSAGE: Opening DB Connection...");
     Database.con_go();
    }
    else
    {
     //System.out.println("MESSAGE: DB Connection Open - Using Active Port...");
    }
   }
   catch (SQLException ogiSql)
   {
    System.err.println(ogiSql.getMessage());
    System.err.println(ogiSql.getSQLState());
    ogiSql.printStackTrace();
   }

   // --- Set Query to clear FPS Silvics table and execute
   try
   {
```

```
    setFlag = Database.con1.prepareStatement(flagAdmin1);
    setFlag.execute();
    setFlag = Database.con1.prepareStatement(flagAdmin2);
    setFlag.setInt(1, Inputs.getStandId());
    setFlag.execute();
   }
   catch (SQLException ogiSql)
   {
    System.err.println(ogiSql.getMessage());
    System.err.println(ogiSql.getSQLState());
    ogiSql.printStackTrace();
   }

   // --- Close Statement and DB Connection
   try
   {
    setFlag.close();
    //Database.con1.close();
    //System.out.println("MESSAGE: SILVICS Table Cleared.");
   }
   catch (SQLException ogiSql)
   {
    System.err.println(ogiSql.getMessage());
    System.err.println(ogiSql.getSQLState());
    ogiSql.printStackTrace();
   }

   return status;
  }
  // FINISH: 'setFlags' Method


//=============================================================================
=====
  // 'clrStand' Method
  //
  // FUNCTION: Clears the FPS Stand table of all data

//=============================================================================
=====
  public void clrStand ()
  {


   // --- Check Status of DB Connection and open if needed
   try
   {
    if (Database.con1.isClosed() == true)
    {
     //System.out.println("MESSAGE: Opening DB Connection...");
     Database.con_go();
    }
```

```
    else
    {
     //System.out.println("MESSAGE: DB Connection Open - Using Active Port...");
    }
   }
   catch (SQLException ogiSql)
   {
    System.err.println(ogiSql.getMessage());
    System.err.println(ogiSql.getSQLState());
    ogiSql.printStackTrace();
   }

   // --- Set Query to clear FPS Stand table and execute
   try
   {
    clrStand = Database.con1.prepareStatement(delStand);
    clrStand.execute();
   }
   catch (SQLException ogiSql)
   {
    System.err.println(ogiSql.getMessage());
    System.err.println(ogiSql.getSQLState());
    ogiSql.printStackTrace();
   }

   // --- Close Statement and DB Connection
   try
   {
    clrStand.close();
    //Database.con1.close();
    //System.out.println("MESSAGE: STAND Table cleared.");
   }
   catch (SQLException ogiSql)
   {
    System.err.println(ogiSql.getMessage());
    System.err.println(ogiSql.getSQLState());
    ogiSql.printStackTrace();
   }

  }
  // FINISH: 'clrStand' Method


//=============================================================================
=====
  // 'clrDbhcls' Method
  //
  // FUNCTION: Clears the FPS Dbhcls table of all data

//=============================================================================
=====
  public void clrDbhcls ()
```

```
{
 // --- Check Status of DB Connection and open if needed
 try
 {
  if (Database.con1.isClosed() == true)
  {
   //System.out.println("MESSAGE: Opening DB Connection...");
   Database.con_go();
  }
  else
  {
   //System.out.println("MESSAGE: DB Connection Open - Using Active Port...");
  }
 }
 catch (SQLException ogiSql)
 {
  System.err.println(ogiSql.getMessage());
  System.err.println(ogiSql.getSQLState());
  ogiSql.printStackTrace();
 }

 // --- Send Query to clear FPS Dbhcls table and execute
 try
 {
  clrDbhcls = Database.con1.prepareStatement(delDbhcls);
  clrDbhcls.execute();
 }
 catch (SQLException ogiSql)
 {
  System.err.println(ogiSql.getMessage());
  System.err.println(ogiSql.getSQLState());
  ogiSql.printStackTrace();
 }

 // --- Close Statement and DB Connection
 try
 {
  clrDbhcls.close();
  //Database.con1.close();
  //System.out.println("MESSAGE: DBHCLS Table cleared.");
 }
 catch (SQLException ogiSql)
 {
  System.err.println(ogiSql.getMessage());
  System.err.println(ogiSql.getSQLState());
  ogiSql.printStackTrace();
 }
}
// FINISH: 'clrDbhcls' Method
```

```
//=============================================================================
=====
 // 'queryTpa' Method
 //
 // FUNCTION: Selects rows from FPS Stand table to get number of live TPA
 // For a single stand at a single year passed.

//=============================================================================
=====
 public float queryTpa (int year, int stand)
 {
  // --- Instantiate variable to return TPA for given Stand and Year
  float tpa = (float)0.00;

  // --- Check Status of DB Connection and open if needed
  try
  {
   if (Database.con1.isClosed() == true)
   {
    //System.out.println("MESSAGE: Opening DB Connection...");
    Database.con_go();
   }
   else
   {
    //System.out.println("MESSAGE: DB Connection Open - Using Active Port...");
   }
  }
  catch (SQLException ogiSql)
  {
   System.err.println(ogiSql.getMessage());
   System.err.println(ogiSql.getSQLState());
   ogiSql.printStackTrace();
  }

  // --- Set query and parameters to Query FPS Stand table for TPA, execute
  try
  {
   getTpa = Database.con1.prepareStatement(queryTpa);
   getTpa.setShort(1, (short)year);
   getTpa.setInt(2, stand);
   standTpa = getTpa.executeQuery();
   if(standTpa.next())
    tpa = (float)standTpa.getDouble("STEMS");
   else tpa = (float)0.00;
  }
  catch (SQLException ogiSql)
  {
   System.err.println(ogiSql.getMessage());
   System.err.println(ogiSql.getSQLState());
   ogiSql.printStackTrace();
  }
```

```
  // --- Close Statement and DB Connection
  try
  {
   getTpa.close();
   standTpa.close();
   //Database.con1.close();
   //System.out.println("MESSAGE: Obtained current LIVE TPA.");
  }
  catch (SQLException ogiSql)
  {
   System.err.println(ogiSql.getMessage());
   System.err.println(ogiSql.getSQLState());
   ogiSql.printStackTrace();
  }
  return tpa;
 }
 // FINISH: 'queryTpa Method


//=============================================================================
=====
 // 'queryCutTpa' Method
 //
 // FUNCTION: Selects rows from FPS Stand table to get number of cut TPA
 // For a single stand at a single year passed.

//=============================================================================
=====
 public float queryCutTpa (int year, int stand)
 {
  // --- Instantiate variable to return TPA for given Stand and Year
  float tpa = (float)0.00;

  // --- Check Status of DB Connection and open if needed
  try
  {
   if (Database.con1.isClosed() == true)
   {
    //System.out.println("MESSAGE: Opening DB Connection...");
    Database.con_go();
   }
   else
   {
    //System.out.println("MESSAGE: DB Connection Open - Using Active Port...");
   }
  }
  catch (SQLException ogiSql)
  {
   System.err.println(ogiSql.getMessage());
   System.err.println(ogiSql.getSQLState());
   ogiSql.printStackTrace();
```

```
  }

  // --- Set query and parameters to Query FPS Stand table for TPA, execute
  try
  {
   getCutTpa = Database.con1.prepareStatement(queryCutTpa);
   getCutTpa.setShort(1, (short)year);
   getCutTpa.setInt(2, stand);
   standCutTpa = getCutTpa.executeQuery();
   if(standCutTpa.next())
    tpa = (float)standCutTpa.getDouble("STEMS");
   else tpa = (float)0.00;
  }
  catch (SQLException ogiSql)
  {
   System.err.println(ogiSql.getMessage());
   System.err.println(ogiSql.getSQLState());
   ogiSql.printStackTrace();
  }

  // --- Close Statement and DB Connection
  try
  {
   getCutTpa.close();
   standCutTpa.close();
   //Database.con1.close();
   //System.out.println("MESSAGE: Obtained current CUT TPA.");
  }
  catch (SQLException ogiSql)
  {
   System.err.println(ogiSql.getMessage());
   System.err.println(ogiSql.getSQLState());
   ogiSql.printStackTrace();
  }
  return tpa;
 }
 // FINISH: 'queryCutTpa Method


//=============================================================================
=====
 // 'setRx' Method
 //
 // FUNCTION: Receives the current prescription log, harvest log, and years
 // array from the 'Control' class.  Inserts the current RX into the FPS
 // Silvics Table (and the Control class then runs FPS Grow).

//=============================================================================
=====
 public int setRX (int [][] rxLog_passed, boolean [] harvestLog_passed,
  int [] years_passed)
 {
```

```
int status = 1; // Dummy variable to return to calling class
int trt_Nbr = 0; // The Silvics table paramater

// --- Check Status of DB Connection and open if needed
try
{
 if (Database.con1.isClosed() == true)
 {
  //System.out.println("MESSAGE: Opening DB Connection...");
  Database.con_go();
 }
 else
 {
  //System.out.println("MESSAGE: DB Connection Open - Using Active Port...");
 }
}
catch (SQLException ogiSql)
{
 System.err.println(ogiSql.getMessage());
 System.err.println(ogiSql.getSQLState());
 ogiSql.printStackTrace();
}

// --- Declare Prepared Statement
PreparedStatement updateSilvics;

// --- Declare Query String
String silvicsRx = "INSERT INTO Silvics values ('OPTI', '..', ?, " +
 "'YNYY', 1, ?, ?, ?, 1, ?, '..', 0, 0)";

// --- Set the Initial State (trt_nbr = 0) RX for 'Opti' regime
try
{
 updateSilvics = Database.con1.prepareStatement(silvicsRx);
 updateSilvics.setInt(1, (short)trt_Nbr);
 updateSilvics.setInt(2, 0);
 updateSilvics.setShort(3, (short)0);
 updateSilvics.setInt(4, 0);
 updateSilvics.setInt(5, 0);
 updateSilvics.executeUpdate();
 updateSilvics.close();
}
catch (SQLException ogiSql)
{
 System.err.println(ogiSql.getMessage());
 System.err.println(ogiSql.getSQLState());
 ogiSql.printStackTrace();
}

// --- Set Trt_Nbr to 1st actual treatment
trt_Nbr = 1;
```

```
// --- Loop to determine any subsequent harvest, trt_nbr 1 thru finish
for (int counter = 0; counter < harvestLog_passed.length; counter ++)
{
 int rxState = 0;
 if (harvestLog_passed[counter] == false)
  rxState = 0;
 else
  rxState = 1;

 switch (rxState)
 {
  case 0:
   break;

  case 1:
   try
   {
    updateSilvics = Database.con1.prepareStatement(silvicsRx);
    updateSilvics.setInt(1, (short)trt_Nbr);
    updateSilvics.setInt(2, years_passed[counter]);
    updateSilvics.setShort(3, (short)rxLog_passed[counter][0]);
    updateSilvics.setInt(4, rxLog_passed[counter][1]);
    updateSilvics.setInt(5, rxLog_passed[counter][2]);
    updateSilvics.executeUpdate();
    updateSilvics.close();
    System.out.println("MESSAGE: Silvics Table Updated.");
    System.out.println(
     "Trt #: " + trt_Nbr + "\tYear: " + years_passed[counter] +
     "\tMethod: " + rxLog_passed[counter][0] + "\tDBH : " +
     rxLog_passed[counter][1] + " \tResid TPA : " +
     rxLog_passed[counter][2] );
   }
   catch (SQLException ogiSql)
   {
    System.err.println(ogiSql.getMessage());
    System.err.println(ogiSql.getSQLState());
    ogiSql.printStackTrace();
   }
   trt_Nbr ++;
   break;
 }
 // FINISH: Switch statement
}
// FINISH: RX For loop assignment

 return status;
}
// FINISH: 'setRX' Method

public int setRegime ()
{
 int status = 0; // Dummy variable to calling class
```

```
    // --- Check Status of DB Connection and open if needed
    try
    {
     if (Database.con1.isClosed() == true)
     {
      //System.out.println("MESSAGE: Opening DB Connection...");
      Database.con_go();
     }
     else
     {
      //System.out.println("MESSAGE: DB Connection Open - Using Active Port...");
     }
    }
    catch (SQLException ogiSql)
    {
     System.err.println(ogiSql.getMessage());
     System.err.println(ogiSql.getSQLState());
     ogiSql.printStackTrace();
    }

    // --- Declare Prepared Statement
    PreparedStatement updateRegime;

    // --- Declare Query String
    String regime_stand = "UPDATE STAND SET REGIME = 'OPTI'";

    try
    {
     updateRegime = Database.con1.prepareStatement(regime_stand);
     updateRegime.executeUpdate();
     updateRegime.close();
     //System.out.println("MESSAGE: Stand Table Regime Updated.");
    }
    catch (SQLException ogiSql)
    {
     System.err.println(ogiSql.getMessage());
     System.err.println(ogiSql.getSQLState());
     ogiSql.printStackTrace();
    }

    return status;
   }

}
// FINISH: 'SILVICS' public class
```

```
package thesis;

/*****************************************************************************
 * 'Optimize' Class
 *
 * FUNCTION: This small class is instantiated by the 'OptiComparator' class
 * and serves a single purpose - to create tiny objects that hold three
 * variables (Scenario #, PNV, OGI) for a global analysis year. The
 * 'Control' class then loads these objects into an array that is sent to
 * the 'OptiComparator' class to be sorted.
 *****************************************************************************/

import java.util.*;
import java.text.NumberFormat;

// --- Define a quick class to hold summary scenario info each period
class Optimize
{
 private int rx = 0;
 private float ogi = (float)0.00;
 private float pnv = (float)0.00;

 // --- 'OptiClass' Constructor
 Optimize(int rx_passed, float ogi_passed, float pnv_passed)
 {
  rx = rx_passed;
  ogi = ogi_passed;
  pnv = pnv_passed;
 }
 // FINISH: 'optiClass' Constructor

 public int getRx()
 {
  return rx;
 }

 public float getOgi()
 {
  return ogi;
 }

 public float getPnv()
 {
  return pnv;
 }

 public String toString()
 {
  // --- Obtain and format a numberFormat for output
  NumberFormat nf = NumberFormat.getNumberInstance();
  nf.setMaximumFractionDigits(2);
  nf.setMinimumFractionDigits(2);
```

```
    return ("Scenario: " + rx + "\tPNV: " + nf.format(pnv) +
      "\tOGI: " + nf.format(ogi));
  }

}
// FINISH: 'optiClass'
```

```
package thesis;

/*****************************************************************************
 * 'OptiComparator' Class
 *
 * FUNCTION: This class implements the Comparator interface and defines the
 * rules for sorting the array of 'Optimize' objects for a given global
 * year analysis.  The rule is to sort the Scenarios in a given year based
 * on PNV (at the 'lookAhead' year set in the 'Inputs' Class) then by OGI.
 *
 *****************************************************************************/

import java.util.*;

// --- Define a Comparator class to determine sorting preference
class OptiComparator implements Comparator
{
 public int compare(Object o1, Object o2)
 {
  Optimize opti_1 = (Optimize)o1;
  Optimize opti_2 = (Optimize)o2;
  if(opti_1.getPnv() == opti_2.getPnv())
  {
   return (int)(opti_2.getOgi() - opti_1.getOgi());
  }
  else
  {
   //return (int)(opti_2.getPnv() - opti_1.getPnv());
   return ( opti_2.getPnv() < opti_1.getPnv() ? -1 :
     ( opti_2.getPnv() == opti_1.getPnv() ? 0 : 1));
  }
 }

 public boolean equals(Object obj)
 {
  return obj.equals(this);
 }
}
// FINISH: 'OptiComparator' Class
```