SANDIA REPORT

SAND99-0397

Unlimited Release

Printed February 1999

~~MS-0619 Review & Approval Desk, 15102~~
~~For DOE/OSTI~~

Weapon Systems Requirements Analysis Employing a Hybrid of Analytic Technologies

Michael Senglaub, Ph.D.

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation,
a Lockheed Martin Company, for the United States Department of
Energy under Contract DE-AC04-94AL85000.

Approved for public release; further dissemination unlimited.

Sandia National Laboratories

RECEIVED
MAR 12 1999
OSTI

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

NOTICE: This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government, nor any agency thereof, nor any of their employees, nor any of their contractors, subcontractors, or their employees, make any warranty, express or implied, or assume any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represent that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government, any agency thereof, or any of their contractors or subcontractors. The views and opinions expressed herein do not necessarily state or reflect those of the United States Government, any agency thereof, or any of their contractors.

Printed in the United States of America. This report has been reproduced directly from the best available copy.

Available to DOE and DOE contractors from
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Prices available from (703) 605-6000
Web site: <http://www.ntis.gov/ordering.htm>

Available to the public from
National Technical Information Service
U.S. Department of Commerce
5285 Port Royal Rd
Springfield, VA 22161

NTIS price codes
Printed copy: A03
Microfiche copy: A01

DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.

SAND99-0397
Unlimited Release
Printed February 1999

Weapon Systems Requirements Analysis Employing a Hybrid of Analytic Technologies

Michael Senglaub, Ph.D.
Distributed Systems Assurance
Sandia National Laboratories
PO Box 5800
Albuquerque, NM 87185-0455
mesengl@sandia.gov

Abstract

A rigorous effort was made to apply professional systems engineering standards to the design of a modified Mk4 fuze. This required that, as systems engineers, we assess the system requirements from a force structure context in which the system is a single component of a strategic capability. This high level requirements analysis objective required the development of analytical capabilities that had previously not existed in a form suitable for design development efforts. The analysis that was conducted necessitated the development of a series of mission projections as well as a limited number of scenarios for which a strategic response might be required. These missions and stockpile projections provided a basis for assessing a subset of the total system requirements. The analyses also provide metrics in the mission dimension which could be used for concept development and system trade studies. The data permits designers and decision makers to identify the most cost effective solution to this design problem.

THIS PAGE INTENTIONALLY LEFT BLANK.

Table of Contents

Executive Summary	5	START III Environments (Series B)	22
Abstract	5	SRF Force Confirmatory	23
Introduction	6	Exploratory Analysis	24
Systems Engineering Analysis	6	Decision Issues	24
Projections / Scenario Development	7	Follow-on Activities	25
Weapon System Optimization	7	Recommendations for further Development	25
Stochastic Covariance Transformations	7	Trend Analysis of Missions	25
Concept Development	7	Targeting Rule Base Issues	25
Analytical Scope	8	C4ISR Projections	25
Taguchi Analyses	8	References	26
Sensitivity Analysis	8	<i>APPENDIX A</i>	27
Confirmatory Calculation	8	Transformation between category codes	
Scenario / Projection Development	9	and descriptive parameters	27
Projections	9	<i>APPENDIX B</i>	29
Scenarios	9	Experimental configurations for validation	
Weapon System Inventories	9	calculations	29
Projected Missions	9	<i>APPENDIX C</i>	30
Analytical Architecture / Technologies	9	Inventories Used in current, START II,	
Fuzzy Logic (Targeting)	10	and START III scenarios	30
Fuzzification	10	<i>APPENDIX D</i>	31
De-Fuzzification	10	Experiment A, current and START II	
Fuzzy Targeting	11	environments	31
Hedges	12	<i>APPENDIX E</i>	32
Genetic Optimization (Weapon System Allocation)	13	Experiment B, START III environments	32
Multi-criteria Decision Analysis (Fitness Functions)	14	<i>APPENDIX F</i>	33
Mission Fitness Correlation	15	SRF experimental setup	33
Objective Probability of Damage	15	<i>APPENDIX G</i>	34
Minimum Yield	15	Sample input data set	34
HOB objectives	15	<i>APPENDIX H</i>	35
Strategic Time Line	16	Table of contents for code listings	35
Validation	16	<i>APPENDIX I</i>	40
Allocation Sensitivity Assessment	17	Code listing for weapon system allocation models	40
Evolutionary Strategies (Weapon System Opt.)	18	<i>APPENDIX J</i>	60
Weapon System Capability Optimization	18	Code listing for weapon system optimization	
Mission Analysis Results	18	algorithms	60
Sensitivity Space	19	<i>APPENDIX K</i>	151
Current / START II (Series A)	19	Code listing for fuzzy logic algorithms	151
START III Environments (Series B)	20	DISTRIBUTION:	199
SRF Force Sensitivity	20		
Confirmatory Analyses and Results	20		
Current / START II (Series A)	20		

List of Figures

Mission analysis sub-process.	6
Generic application of robust design.	8
Basic algorithm structure.	10
Graphic representation of rule processing.	11
Membership functions for Mission, Function, and Counter_Lethality.	11
Target timing as a function of category code.	13
Target timing as a function of target Counter_Lethality.	13
Target ranking by category code.	13
Target ranking by function.	13
Response surface characteristic of a weapon system allocation problem.	14
Objective probability of damage fitness function.	15
Fitness function for minimum yield constraints.	15
Fitness function for height of burst (HOB) preferences.	16
Delivery time characteristics for IC's, SL's and air delivered weapon systems using sigmoid functions.	16
Target distribution for base mission.	16
Allocations of weapons to target class 871 with different response times.	16
Allocation differences for a second target class.	17
Timing allocation information for base prompt mission.	17
Mission fitness sensitivity.	17
Mission Pd sensitivity results.	18
Sensitivity for experiment series A.	19
Sensitivity for experiment series B.	20
Sensitivity results for the SRF series of experiments.	20
Weapon radius range of targets assigned to weapon systems.	21
Performance range of system against targets allocated.	21
Deployed stockpile allocation for mission scenario A.	21
Required effective CEPs and associated distributions.	22
Convergence information for experiment series A.	22
Weapon radius range of targets assigned to weapon systems.	22
Performance range of system against targets allocated.	22
Deployed stockpile allocation for mission scenario B.	23
Required effective CEPs and associated distributions.	23
Convergence information for experiment series B.	23
SRF effective CEP requirements and associated distribution.	23
Stockpile optimization sensitivity assessment.	24
Yield distribution for an optimized stockpile.	24
Effective CEP requirements for each system in the optimized inventory.	24

Weapon Systems Requirements Analysis Employing a Hybrid of Analytic Technologies

Michael Senglaub, Ph.D.
Sandia National Laboratories
PO Box 5800 MS0455
Albuquerque, NM 87185
mesengl@sandia.gov

Executive Summary

The Mk4/W76 is rapidly approaching its original design service life. The result is an effort to bring the system in line with current technologies and define system requirements based on information that is not 20-30 years out of date. System requirements include, safety, cost, reliability, certifiability, dismantlement, as well as performance. The design problem is multi-criteria in nature, is not single issue, and requires information that can be used by decision makers to make the best concept selection for inclusion in the nation's strategic inventory.

The effort described in this document explores the subset of requirements, specifically the performance of the stockpile and the requirements of the Mk4/W76 component of the deployed stockpile. The effort required the development of an analytic capability to explore multiple strategic configurations based on missions, targeting heuristics, operational scenarios, inventories, and system concepts. The capability developed is based on artificial life (A-life) concepts which proved to possess well matched capabilities. These analytic technologies included, fuzzy logic, evolutionary strategies, and an approximation to simulated annealing.

These analytic technologies had to be integrated to a number of classical techniques including, Taguchi analysis, multi-criteria decision making, scenario development, and target vulnerability calculations. Target vulnerabilities reflected overpressure, dynamic pressure, cratering and g-type targeting. The algorithms were based on Brode-Speicher correlations and on correlations found in PDCALC.

The requirements development focused on three basic sub-missions, START II projections (Series A in Table 1&4), START III scenarios (Series B in Table 1&4), and a number of strategic reserve force (SRF) missions. Within each of these three sub-missions variations in the independent variables were considered to capture uncertainties in projections. The independent variables included, operational status, mission, deployed inventories and mix, strategic reserve size, reliabilites, mission success criteria, and new system configurations. Taguchi analysis techniques were used to iden-

tify the most stressing conditions for use in three confirmatory calculations used as the basis of the requirements assessments. The table which follows defines the results of the three confirmatory calculations. The information includes percentage of the mission assigned to the replacement SL system, the "normalized effective CEP" and the fraction of the mission employing a particular evolved fuze option.

Table 1: Confirmatory analysis results.

Series A 58%		Series B 39%		SRF 100%	
1.0	11%	1.0	6%	1.0	4%
1.8	4%	1.03	3%	2.5	96%
2.13	2%	1.79	4%		
2.23	1%	2.18	2%		
2.5	40%	2.5	24%		

Abstract. A rigorous effort was made to apply professional systems engineering standards to the design of a replacement fuze. This required that, as systems engineers, we assess the system requirements from a force structure context in which the system is a single component of a strategic capability. This high level requirements analysis objective required the development of analytical capabilities that had previously not existed in a form suitable for design development efforts. The analysis that was conducted necessitated the development of a series of mission projections as well as a limited number of scenarios for which a strategic response might be required. These missions and stockpile projections provided a basis for assessing a subset of the total system requirements. The analyses also provide metrics in the mission dimension which could be used for concept development and system trade studies. The data permits designers and decision makers to identify the most cost effective solution to this design problem.

The analytical hybrid developed employed a number of arti-

Figure 1. Mission analysis sub-process.

ficial life technologies as well as a convolution with fuzzy logic and multi-criteria decision making constructs. The problem associated with force structure analyses is the multi-dimensional character of the problem, as well as qualitative constraints which may be a product of the politics of strategic development. The mechanics of force structure analyses is a constrained multi-criteria optimization problem in which we are maximizing system performance on a strategic mission. The tools provide an ability to perform massive searches of concepts across a broad spectrum of missions in order to identify the most robust, effective solution to the constrained strategic force design problem.

The study explored a number of projections and explored scenarios associated with strategic reserve forces in the effort to identify a robust set of capabilities and avoid over specification which can result from system assessments in the absence of force structure considerations. The analyses were performed within a Taguchi framework in order to improve the efficiency of the calculations and provide insights into system sensitivities. Taguchi analysis is a design of experiment technique that has found favor in the commercial sectors. Precursor sensitivity calculations were performed to identify the most stressing conditions to be imposed on the replacement SL weapon system. These conditions were then used in sets of confirmatory calculations to identify the system requirements in terms of weapon system circular error probable (CEP) and associated weightings or the importance of an option requirement given mission parameters.

Introduction.

The suggested and actual service life of nuclear weapons ranges from 20 to 50 years depending on the system. The Mk4/W76 is reaching the end of design service life and as a result requires the attention and re-evaluation by the DoD and DOE design communities. The principle areas of emphasis are life extension issues and an increased nuclear safety margins comparable to the newer systems within the deployed stockpile. The measures of effectiveness (MOE's) for the system include cost, safety, mission, reliability, certification, retirement objectives, and the other "ilities" such as availability, maintainability, producibility, vulnerability, testability, etc. Mission effectiveness requirements pose a unique problem because of the necessity of identifying missions 30-50 years in the future, which represents the potential service life of the system. Projections of strategic systems missions, and operational environments have typically been less than 10 years. Design based on these projections could result in a system design that is obsolete at the time of system deployment. Uncertainties associated with future missions must be addressed by either, (1) flexibility of design, which can result in significant cost impacts to the customer or (2) as systems analysts we work to mitigate uncertainty of future missions.

Systems Engineering Analysis.

The analysis effort to support the development effort explores the transformation of mission needs to hardware concepts. Mission needs can be defined as distributions of

targets in which the dimensions of the distribution include, geographic, clusterings, hardness or damage potential, areal extent, terrain features, and position uncertainty. Hardware concepts represent algorithms implemented in suites of sensors and processors which transform reentry uncertainties into pre-burst covariances which will result in suitable kills of mission targets. A simplified graphic of the development sub-process is depicted in figure 1. The force structure analysis provides the foundations for identification of range weighting distributions, target allocation distributions and associated target vulnerabilities. The range weightings allow us to identify sets of pre-reentry covariances to be used in developing optimal system transformations. These covariances represent the initial conditions for fuzing sub-system design. The target distributions, associated with a particular weapon systems mission, provide the final conditions. The system concept is the black box that transforms the initial conditions to the final pre-burst conditions.

Projections / Scenario Development. Projections and scenario development represent the initial activities needed to begin the systems analysis effort. Projections and scenarios generate information concerning strategic missions, inventories, and operational states for use in the development efforts. Projections, while accurate, lack the ability to anticipate discontinuous effects. These might be advances in technology, major political shifts, or common mode system failures. Projections can be characterized as extrapolations of trends observed in historical data. They can be extremely accurate for short term predictions. Scenarios, provide capabilities necessary for mission analysis efforts associated with systems requiring service lives on the order of decades. Scenarios explore predictions from fundamental dynamic context. Predictions based on this approach might address global states from which problem specific questions are answered. For example, given a multi-polar high technology future what might a strategic mission against a central american adversary look like?

Weapon System Optimization. The mission profiles and weapon system inventories which result for the projections and scenario development effort are used as the basis for the weapon system optimization analyses. The objective is to identify one or more system configurations which when integrated into the existing deployed stockpile will compliment the inventories and not duplicate the other systems. Complementing and not duplicating capability has positive and negative connotations, a cost effective stockpile is one that mitigates duplication, while a reliable stockpile is one that may possess limited amounts of redundancy to mitigate effects resulting from common mode failures or delivery system survivability issues. The weapon to target allocation algorithms provide the basic fitness functions for use in the optimization efforts. Massive searches can be conducted in

which system concepts are assessed within a force structure context to identify the most suitable solution to the design problem. The solution which best compliments the force structure during this phase of the studies leads to robust sets of system requirements.

Stochastic Covariance Transformations. The analysis effort identifies sets of pre-reentry conditions and covariances that characterize the initial conditions a weapon system must operate from. We also see that the final conditions are also represented by sets of conditions / covariances. The system design effort involves finding the transformation which can optimally transform the initial condition set into the final condition set. The constraints that are imposed in seeking this optimal transformation are based on the flexibility of the weapon system (sensor suites, aero-ballistic capability, etc.), and on the operational constraints (including information, preset accuracies, word lengths, delivery system accuracy, etc.) of the weapon and delivery system.

The search for this transformation is based on conventional design paradigms. A tool that could prove highly useful for this activity is the genetic programming approach being developed and used in a number of design arenas. This technology is founded in the evolutionary analytical sciences and is being used to design circuits, bridge trusses, and orbital control systems. The effort did not possess the resources or time to develop the unique operators for this type of genetic programming problem.

Concept Development. A basic conventional approach to design is being employed in the concept development activities. The optimization tools provide sets of effective CEPs which must be generated by the system hardware. The relationship between an effective CEP and probability of target kill is represented in the next equations.

$$Pk = 1.0 - 0.5 \left(\frac{Wr}{CEP} \right)^2 \quad \text{Eqn. 1}$$

$$CEP_{eff} = \frac{Wr}{\sqrt{\frac{\ln(1.0-Pk)}{\ln(0.5)}}} \quad \text{Eqn. 2}$$

Concept development is achieved through a process of concept specification and transformation functionality. Comparisons between capability and objective identify suitable choices of concept architecture. Monte-Carlo techniques are employed to convolve burst covariances with target damage contours to identify system level probabilities of kill. This result is then compared to the system requirement (effective CEP) through the translations defined in equations 1 and 2. The process while not elegant has proven suitable for many

generations of strategic system development.

Analytical Scope. A broad set of mission projections were proposed as well as scenario excursions developed in an effort to capture the mission requirements for this system. The system capabilities were assessed based on a system-of-systems context in which the Mk4 was considered to be part of a larger deployed nuclear stockpile. The delineation of the mission projections and the scenario projections provided a foundation from which weapon system optimizations could be conducted to identify the most suitable Mk4/W76 weapon system configuration. This approach differed from efforts in the past because it required analytical capabilities that have not existed within the DOE community. While the DoD community possesses tools approaching the needed capabilities, the DoD algorithms are designed to provide detailed lay-downs and are ill suited for the studies needed to identify system requirements. The effort, therefore, required the development of tools which could be used to search for optimal configurations of the Mk4/W76 system under a broad spectrum of conditions. The tools needed to operate within a Taguchi framework, capture targeting heuristics, and facilitate multi-criteria decision problems. A brief description of these tools and the analytical framework follows in the next sections.

Taguchi Analyses.

Taguchi design techniques are employed in environments associated with non-linear complex systems and where engineers are concerned with achieving robust designs. The objective of robust design is to set design control parameters to target values that minimize response variability. Classical system response can be defined by the following relation.

$$y = f(M, x, z) \quad \text{Eqn. 3}$$

M is the signal factor and defines external control of the system by an operator in order to attain some intended response values. x captures the noise characteristic of the system and environment which cannot be controlled by the system designer. Finally, z represents the control factors, the parameters that are under control of the designer to satisfy system requirements.

Taguchi techniques recognize that parameter design and tolerance design are two distinct approaches to robust design and there can be significant cost advantages associated with parameter design in non-linear systems. Robust or Taguchi design is founded on the premise that any deviation from a target value has a cost associated with it. The classic design margin approach assumes that all solutions within the design margins are equally acceptable. The next figure attempts to characterize this principle. When system cost is a function of response variability, a designer can work to reduce the

uncertainties associated with the control factor, ie. tolerance design, or find a control factor target value that corresponds to a "flat" part of the response curve (shift of control from 75 to 175 in Figure 2).

Figure 2. Generic application of robust design.

The techniques associated with these methodologies become highly useful for system analysis problems. The design problem being explored possesses a number of factors that must be addressed from a multi-criteria perspective in order to define system requirements. The sensitivity studies that are basic to the technique allow us to identify combinations of conditions that lead to the best design solution.

Sensitivity Analysis

A principle of Taguchi analysis is the identification of important design or analysis factors, usually in noisy environments. The analysis or experimentation that is typical of Taguchi methodologies provides information that can be used to optimize "level of effort" in design or analysis efforts. The process involves a parameter study to identify factors associated with the design problem and a noise analysis to define the characteristics of the operational environment. The control factor levels are set based on engineering judgement and constraints being imposed on the design problem. Once the parameter analysis is complete the technique requires a selection of an orthogonal matrix, the assignment of variables or parameters to the columns of the orthogonal matrix, and the execution of analytical experiments. Subsequent to the experiments a statistical analysis is conducted to determine sensitivities, non-linearities, confounding effects etc. The experimental layout for a number of the analyses are included in the appendices.

Confirmatory Calculation. The preliminary sensitivity analyses used in these studies aided in the identification of the most stressing conditions being imposed on a replacement SL weapon system. For the purposes of the weapon system optimizations, we are attempting to identify the most system stressing conditions, and options that can provide the best performance under these conditions. The confirma-

tory calculation selects these conditions for use in a final calculation to determine the force configuration and system characteristics that best meets the needs of the customer.

Scenario / Projection Development.

There are two basic concepts for attempting to define environments a system may experience, the operational conditions that may be imposed and the functionality that may be assigned to a system in the future. The first, more basic approach is that defined as "projections". Projections are based on trend analyses. The second approach is "scenario development". In this case attempts are made to define a future environment and the system is then designed for functionality requirements in this postulated future.

Projections.

As indicated projections are based on trend analyses in which historical information is used to define conditions which will exist in the near future. Projections can possess high degrees of fidelity for these near term conditions but lack the ability to anticipate the unknown. Unknowns might be major political shifts and alliances, or major developments in technology.

A number of mission spectrums and stockpile inventories were identified and assessed in order to identify the effective CEP requirements for a replacement SL weapon system intended to compliment the deployed stockpile. The series A set of calculations, current and START II environments, involved pure projections based on treaty developments, operational trends, and target vulnerability trends.

Scenarios.

The series B set of calculations, START III, and the SRF missions were based on a limited scenario development approach. Environments were defined to capture inventories and target sets that might be evident in a future in which START III treaties were verified. The target sets used were based on projections of START II type mission spectrums. A more comprehensive scenario development effort would have included effort to define the composition of future target characteristics.

A great deal of consideration should be expended in an effort to capture future scenarios into which the replacement SL weapon system might be placed. The effort expended in this area will ultimately save the nation money since the system

will possess the flexibility and robustness to deal with a broad spectrum of missions. Some of the considerations that may affect future missions include:

- Hardness trends
- Defensive weapon trends
- C4I projections
- Geo-political trend analysis
- New delivery platforms. Fast bomber fleet may impact requirements.

Weapon System Inventories. The inventories of weapon systems for the three analysis subsets are listed in table 4 of Appendix C. The inventories vary due to projections by different organizations and reflect the differing assumptions concerning possible treaty limits. The SRF inventories were assumed to consist solely of W76 weapons with varying quantities. This assumption was made to generate the most significant system performance demands.

Projected Missions. The current and START II experiments, Series A, explored three mission profiles. These are Scenarios A, B, and C in the experimental layout defined in the classified addendum. Scenarios D through G were estimates of possible mission profiles in a START III, or Series B, type environment. The target sets were generated statistically from extensive databases. The intent was to capture system needs and not to define or speculate on potential adversaries and missions. Each experiment consisted of 1000 targets generated from statistical mission profiles. The the weapon system inventories were scaled from full target sets to the 1000 number to reflect or capture the implications of target rich and weapon rich environments.

The last four scenarios were used in the strategic reserve force (SRF) analyses and were designed to capture requirements which may arise due to third party belligerents. Again conditions were sought to develop requirements rather than plan an SRF mission. These cases were run against 500 targets per SRF scenario.

Analytical Architecture / Technologies.

The optimization algorithms are based on three major analytical elements, the system optimization routines; the target allocation routines, and a set of fuzzy logic routines.

Algorithm Structure

Figure 3. Basic algorithm structure.

The two major sub-blocks can be executed independent of the weapon optimization functions in stand-alone mode. This provides us with a higher degree of modularity and reuse as well as enhancing the validation and debug of the various algorithms. The targeting heuristics modeled in the fuzzy logic sub-component represents our best attempt to mimic the perceived targeting philosophy at strategic headquarters in an unclassified setting. The resultant rules could be significantly improved by obtaining a current allocation and using genetic programming principles to find an optimal set of rules reflecting current targeting philosophies.

Fuzzy Logic (Targeting).

Fuzzy set theory is an extension of classic set theory. In classic set theory component membership is defined by a Dirac Delta function, either 1 or 0. Fuzzy set theory allows set members to possess membership values which range continuously over the domain between 0.0 and 1.0. These membership functions provide us with an ability to characterize the degree to which an element might belong to a specified set. For example, given a set of "accurate weapons", a system possessing a CEP of "X" feet might possess a membership value of 0.5, while a system with a "10X" foot CEP has a membership value in the set of "accurate weapons" of 1.0e-9, reflecting a fact that in some scenarios even a "10X" CEP might be considered accurate. Unlike statistical concepts which consider the uncertainties associated with

assessing an observable, fuzzy deals with the uncertainty associated with the underlying physics or functionality.

Fuzzy logic extends this unique set theory by defining the calculus that can be used to capture behavior in this problem space. This treatment of fundamental functional uncertainties provides us with another tool for modeling complex dynamics, and non-linear systems. Implementation of the calculus of fuzzy logic provides us with the ability to perform approximate reasoning. The dynamics of the system is captured by semantic rule sets operating on semantic variables. The multi-faceted nature of targeting, both deterministic and heuristic, requires a hybrid approach for which fuzzy logic is a good match.

Fuzzy provides us with the ability to model / approximate the highly complex problem of weapon system allocation in which many of the phenomena cannot be modeled deterministically. The politics, and psychology of strategic targeting can be easily modeled in linguistic rules but is impossible in closed form analytic form. Fuzzy also permits us to consider conflicting guidance in the allocation process. As directives evolve, allocation strategies may emerge which are in internal conflict. Unlike expert rule based systems which typically fail under these conditions, fuzzy logic can operate under these conditions flawlessly.

Fuzzification. Fuzzy modeling requires transformations between crisp and fuzzy domains in order to capture the dynamics of a system. The objectives of fuzzy models is to develop rule sets which operate on linguistic variables which in turn capture the dynamics of the process or system. Fuzzification is a process in which the concepts of numbers are re-represented in terms of fuzzy linguistic entities. Once the model representations are captured in this fuzzy linguistic space the operations or transformations defined by the rule set can be performed. The result of the fuzzy transformations is another set of linguistic fuzzy variables which must be transformed back into the real domain for evaluation. This process is de-fuzzification.

De-Fuzzification. After a rule set has been processed the resulting consequent set must be quantified. This process is called "de-fuzzification" in the literature (Cox, 1994). There are three methods that have been considered in these development efforts. They include: the centroid method, the average maximum method and the maximum method. The next

figures provide a simple graphic of rule processing.

Figure 4. Graphic representation of rule processing.

Figure 4 graphically demonstrates the processing of a simple set of rules as defined below.

- if COUNTER_LETHALITY is MODERATE then PRIORITY is HIGH
- if MISSION is STRATEGIC_OFFENSE then PRIORITY is LOW

Assuming the value for COUNTER_LETHALITY was 57 and MISSION was 16.4. The lines between the charts reflect the degree of truth of the antecedents and the resulting impact on the consequent parameter, PRIORITY. The membership distribution for PRIORITY is represented by the two clipped sigmoid distributions. The average maximum defuzzification method would produce a priority result in the neighborhood of 0.9. The centroid method would produce a priority in the range of 0.6 to 0.8. The maximum method would result in either 0.8 or 1.0 depending on the version of the de-fuzzification method.

Fuzzy Targeting. In order to develop rules associated with weapon system targeting we were forced to redefine a coding system for the targets based on a three parameter model. The category code structure associated with strategic target sets represent a book keeping scheme, as opposed to a characteristic scale similar to a member of a fuzzy set. The raw category codes do not allow us to take advantage of the

inherent strengths of fuzzy logic.

Figure 5. Membership functions for Mission, Function, and Counter_Lethality.

The independent variables selected include the following three parameters: Mission, Function, and Counter_Lethality. Associated with each variable is a set of levels. Levels act as units associated with the variable similar to the numbers which belong to the positive integer set. The levels associated with Mission include: strategic offense and defense, tactical offense and defense, and multi-missions. Counter_Lethality is comprised of low, moderate and high. Counter_Lethality in this context is the damage a surviving target could inflict on us. Finally, function consists of: fabrication, support, command & control, combat, storage, attack, and "weapons of mass destruction" (WMD) storage. The next figure depicts the variables and the membership functions for each level used in the studies.

The objective of the fuzzy targeting within this analytical architecture involves the assessment of target weighting, rec-

ommended detonation altitudes and strategic time line considerations associated with delivery system response times and survivability. The consequent fuzzy variables include: priority, timing and height of burst (HOB). The levels defined for priority include: High, moderate, and low. The levels of timing include: days, hours, and minutes. Finally, the levels of HOB include: high, moderate, low, and contact.

Hedges. Hedges represent mathematical adjectives in the arena of fuzzy logic. They are used as modifiers to semantic variables and act to change the distribution associated with the membership function. The membership function for "high" of the semantic variable counter_lethality can be modified by the hedge "very". The effect of that operation is a change in the membership function by squaring each point on the membership function. This results in a sharpening of the distribution function for "high".

The first cut at sets of fuzzy targeting rules are provided in

Table 2: Sample rule set used in the first strategic targeting studies.

Rule No.	Rule
a1	< if FUNCTION is FABRICATION then HOB is HIGH >
a2	< if COUNTER_LETHALITY is HIGH and MISSION is STRAT_OFF then HOB is CONTACT >
a3	< if FUNCTION is SUPPORT then HOB is HIGH >
a4	< if FUNCTION is CCC then HOB is LOW >
a5	< then HOB is MODERATE >
b1	< if MISSION is STRAT_OFF then PRIORITY is VERY HIGH >
b2	< if MISSION is STRAT_DEF then PRIORITY is HIGH >
b3	< if MISSION is TAC_OFF then PRIORITY is SOMEWHAT HIGH >
b4	< if MISSION is TAC_DEF then PRIORITY is MODERATE >
b5	< if MISSION is MULTI_MIS then PRIORITY is VERY MODERATE >
b6	< if MISSION is TAC_DEF and FUNCTION is FABRICATION then PRIORITY is VERY LOW >
b7	< if MISSION is TAC_DEF and FUNCTION is STORAGE then PRIORITY is LOW >
b8	< if MISSION is TAC_OFF and FUNCTION is SUPPORT then PRIORITY is MODERATE >
b9	< if COUNTER_LETHALITY is VERY HIGH then PRIORITY is VERY HIGH >
b10	< if FUNCTION is SUPPORT then PRIORITY is LOW >
b11	< if FUNCTION is CCC then PRIORITY is MODERATE >
b12	< if FUNCTION is FABRICATION then PRIORITY is LOW >

Table 2: Sample rule set used in the first strategic targeting studies.

Rule No.	Rule
b15	< if FUNCTION is ATTACK then PRIORITY is VERY HIGH >
b16	< if FUNCTION is WMD_STORAGE then PRIORITY is HIGH >
c1	< if COUNTER_LETHALITY is HIGH then TIMING is MINUTES >
c2	< if COUNTER_LETHALITY is MODERATE then TIMING is HOURS >
c3	< if COUNTER_LETHALITY is LOW then TIMING is DAYS >
c6	< if MISSION is STRAT_OFF then TIMING is MINUTES >
c7	< if MISSION is STRAT_DEF then TIMING is FEW MINUTES >
c8	< if MISSION is TAC_OFF then TIMING is HOURS >
c9	< if FUNCTION is FABRICATION then TIMING is DAYS >
c10	< if FUNCTION is SUPPORT then TIMING is HOURS >
c11	< if FUNCTION is CCC then TIMING is MINUTES >
c12	< if FUNCTION is COMBAT then TIMING is HOURS >
c13	< if FUNCTION is STORAGE then TIMING is DAYS >
c14	< if FUNCTION is ATTACK then TIMING is MINUTES >
c15	< if FUNCTION is WMD_STORAGE then TIMING is HOURS >

the table 2.

Appendix A contains tables defining the transformation between category codes, and the metrics used in the weapon system allocation code. As mentioned earlier, category codes are unsuitable for use in the fuzzy targeting models and had to be transformed into semantic variables that could be used by the rule set defined. The approach was to select roughly 80 plus category codes, transform them into the semantic space discussed and execute the rule base on these 80 plus target types to generate values for the HOB, priority, and timing variables which were used in the weapon system allocation portion of the analytic models.

The de-fuzzification method used to generate the table is the "centroid" de-fuzzification method. The average maximum and the maximum methods were explored, however, the centroid method appears to produce the best results for this application. The next couple of displays provide results of the rule set defined in table 2. Figure 6 provides a subset of information useful for assessing the suitability of the targeting rule base. The axes represent target accounting codes

vesus the time urgency parameter.

Figure 6. Target timing as a function of category code.

Figure 7 provides information which correlates target Counter_Lethality with the time urgency fuzzy variable. The trend identified is consistent with expectations, i.e. lethal targets should be attacked early in a strategic operation.

Figure 7. Target timing as a function of target Counter_Lethality.

Figures 8 and 9 provide information concerning the prioritization of targets. This dimension of the problem becomes important in target rich environments. In target rich environments the process involves an allocation of weapons to the targets starting with the highest priority (rank 1 in the charts) to the lowest priority. The "b" series of rules, the rules associated with priority, in table 1 were used to generate the fol-

lowing plots. The series of plots were used as part of a validation process. The objective was to capture, within the time frame of the analyses, a set of targeting heuristics that reflected actual targeting mechanics. Additional effort would be needed to add fidelity to the rule base as well as provide a foundation from which policy issues associated with strategic targeting could be explored.

Figure 8. Target ranking by category code.

Figure 9. Target ranking by function.

Genetic Optimization (Weapon System Allocation).

The objective of the allocation algorithms is to select the best weapon for use against a target consistent with a set of criteria or objectives. A characteristic of this optimization problem is the multiple combinations of acceptable solutions. A graphical depiction of the problem is the generic representation of the optimization complexity provided in Figure 10 below. The axes are indicative of combinations of alloca-

tions and the resulting fitness of the mission. The point to be drawn from the figure is the complex response surface associated with weapon system allocation optimizations. The initial effort to find a solution algorithm was to use genetic algorithms with cyclic permutation operators. What became evident, and it should have been obvious on my part, was the inordinate amounts of computational effort required to find a solution. Genetic algorithm can be characterized as directed random searches for an optimal solution. The random aspects of the search on a complex response surface is mechanistically inefficient. The complexity associated with the optimization is in part due to the numerous combinations of optimal solutions and to fitness functions that are at a mission level. The importance of unit allocation changes using these fitness functions is averaged over the number of targets in the strategic mission. The result is minimal impact to mission fitness due to small changes in the allocation scheme.

Figure 10. Response surface characteristic of a weapon system allocation problem.

The allocation solution methodology which proved to be the most efficient was based on a relaxation principle. The mechanics of this optimization technique involves a pre-screen for sub-optimal weapon allocations in which suitability is defined by the user. Each sub-optimal solution is compared randomly to other sub-optimal solutions in this distribution, if mission fitness is improved by exchanging weapon allocations, the exchange proceeds otherwise the exchange of allocation does not occur. The pseudo code for

the optimization is presented in the following table.

Table 3: Optimization Pseudo code.

```

Loop on all allocations
{
  Compare allocation fitness to user defined acceptance
  If ( fitness < acceptance )
    Add to "unacceptable allocation distribution (UAD)"
  }
Loop on UAD distribution
{
  randomly select another allocation from UAD
  exchange weapon allocations for the two targets
  if( fitness new > fitness old )
    make exchange permanent
  }

```

The fitness of an allocation is defined by sets of criteria to be discussed in the next section.

Multi-criteria Decision Analysis (Fitness Functions).

Strategic targeting represents a multi-dimensional optimization problem in which many of the criteria may possess qualitative foundations. Target priority is a clear example of a qualitative criteria. Policy directives, military objectives, and geo-political opinions all contribute to the dynamic of target prioritization. The following set of criteria was used in the force structure analysis documented in this note. The criteria consisted of the following 7 factors.

- Minimum Yield
- Objective probability of damage (Pd)
- Stockpile constraints
- Target importance
- Strategic time lines
- HOB objectives
- Control sub-system Reliability

The factors listed can be grouped into two basic types, those exhibiting threshold type behavior and those more appropriately characterized as goal functions. Threshold criteria drive the suitability of a weapon to target allocation to zero if the fitness threshold of the criteria is not reached. This is used to capture stockpile limits, if there are insufficient numbers of systems to cover targets, the fitness contribution for an uncovered target must be zero. With goal type criteria, such as objective Pd, credit is given for an allocation which produces a probability of target kill different from the mission goals. This characterization, while rather simple, captures the functionality of interest. Second order effects might drive certain targets to transition into threshold type criteria. Allocations failing to inflict a minimum level of damage, would be equivalent to an allocation which inflicts no target damage. The mission fitness correlation is defined

in the next section and characterizations of a number of the fitness criteria are discussed in subsequent sections.

Mission Fitness Correlation. The functional form of the mission fitness function is presented in equation 4. The functionality captures threshold type criteria as well as goal type criteria.

$$F_{total} = F_{stk} \cdot F_{wt} \cdot F_{tm} \cdot F_{re} \cdot (F_{Pd} + F_h + F_y) \quad \text{Eqn. 4}$$

In this expression F_{tm} is the fitness associated with strategic timing issues, F_{re} addresses system reliability, F_{Pd} captures objective Pd issues, F_h covers to height of burst considerations, and F_y represents minimum yield considerations. F_{stk} captures the stockpile fitness and is either 0 or 1 depending on the ability to allocate a weapon system to a given target. F_{wt} captures target weighting concerns. When there are sufficient numbers of weapons in the inventory the term takes on the priority of the target, the problem becomes one of maximizing the sub-fitness' for high priority targets in order to maximize mission fitness. The weighting parameters also act as a threshold in target rich environments. Under these conditions, a lower priority cutoff is established that corresponds to the number of higher priority targets that can be attacked.

The maximum mission fitness that is attainable is 3.0 and the minimum fitness is 0.0. Each sub-criteria of the mission fitness function can range in value from 0 to 1 with the summed terms in parentheses driving the maximum value.

Objective Probability of Damage. Figure 11 defines the functional form of the probability of damage fitness correlation. The peak of the curve occurs at a value defined as the objective Pd. The degradation in fitness for probabilities of damage below and above that value is founded on the logic that too much capability is as bad as too little damage capability. This effect shows up in the requirements for the weapon system. Fitness correlations that indicate "bigger is better" will drive the system requirements up and potentially

impact cost.

Figure 11. Objective probability of damage fitness function.

Minimum Yield. The functional form of this fitness criteria, Figure 12, had to capture the targeting objective of imposing the minimum yield available on the target while still satisfying the other criteria. This criteria has foundations in the desire to mitigate collateral damage in particular, and to preserve our most capable systems for allocations for more stressing problems.

Figure 12. Fitness function for minimum yield constraints.

HOB objectives. The criteria associated with heights of burst (HOB) captures issues associated with optimal or recommended HOBs. Physics of weapons, targets, terrain, or trajectory can impose constraints on targeting HOB. This fitness function, Figure 13, captures the effect of this prob-

lem mathematically.

Figure 13. Fitness function for height of burst (HOB) preferences.

Strategic Time Line. Strategic time line is a decision metric associated with target attack urgency, system delivery times, and indirectly delivery system survivability. Target time urgency takes target counter lethality into consideration when allocating weapon systems to the target. For example, strategic offense targets are a class of target that would be targeted early in a mission while strategic storage facilities, under most scenarios, could be targeted late in a mission. Survivability of the delivery system becomes a timing issue if there is some question of first strike survivability. These systems might be best suited for allocation to time urgent targets. The forms assumed for weapon system timing considerations are presented in Figure 14. These figures represent ICBMs, SLBMs, and air delivered systems. In conjunction with the fuzzy targeting model, which defines timing urgency, these generic delivery time curves provide a decision metric for proper assignment of system to target.

Figure 14. Delivery time characteristics for IC's, SL's and air delivered weapon systems using sigmoid functions.

Validation. Validation of the allocation model was conducted on a best guess of current targeting conditions and associated allocation rules.

Figure 15. Target distribution for base mission.

Two sets of base mission allocations were performed. The first used response times for the air delivered assets such that 50% of the weapons could be delivered in 750 minutes. The second-set of allocations set the mean delivery time to 1200 minutes. Figures 16 and 17 provides comparisons of a couple of target classes for the two response times.

Figure 16. Allocations of weapons to target class 871 with different response times.

Figure 17. Allocation differences for a second target class.

Figure 18 provides information related to the timing aspects of the strategic timing problem. For reasons of survivability and delivery delays, we expect the allocation algorithms to preferentially select IC systems to attack time urgent targets and air delivered assets to be used against targets that are not time critical. From the figure we see that for a prompt response scenario, these allocation patterns are evident.

Figure 18. Timing allocation information for base prompt mission.

Other aspects of the targeting validation considerations were demonstrated in the fuzzy logic section presented earlier (Figures 4-7). We found that the generic characteristics of

the targeting allocation algorithms demonstrated trends that were expected for the missions assessed.

Allocation Sensitivity Assessment. A sensitivity set of calculations were performed to assess the significance of model and system parameters in the allocation process. The exercise considered operational scenarios, mission suitability, system reliabilities, code convergence parameters, defuzzification methods, and the category code transformation matrix. The sensitivities were based on the L_{18} orthogonal matrix in which the 8 columns were fully assigned to analysis parameters. The table of experiments and settings are listed in the table of appendix B. A detailed description of the Taguchi analysis methodology follows in later sections.

The result of these 18 calculations are shown in the next figure.

Figure 19. Mission fitness sensitivity.

The metric displayed in the Figure 19 is the fitness function defined by Equation 4. The expression provides numerical foundations for making comparisons for the different allocations of weapons to targets. The next figure demonstrates the sensitivity of mission Pd to the same set of parameters.

The vertical extent of a parameter in these plots indicates sensitivity to a design or analysis parameter. We also need to remember that each data bar represents the effect of that parameter averaged over all the remaining parameters. What we observe in these displays is a dominant system performance sensitivity to the operational scenario. The probability of damage (Pd) sensitivity plots really do not add a great deal of extra information but simply reinforce the sensitivity

observations from Figure 20.

Figure 20. Mission Pd sensitivity results.

The fitness sensitivity plots indicate, that along with operational scenario, objective Pd or mission suitability is significant. The plot also indicates that submarine launched (SL) reliability and the de-fuzzification method needs to be examined closely when performing allocation studies. The implication of these results are a sensitivity of the design to perceptions of mission success and operational scenario.

Analysis recommendations include the use of the "centroid" de-fuzzification and a high user defined convergence acceptance level. The acceptance level acts as a threshold for improvement considerations in the allocation process.

Evolutionary Strategies (Weapon System Opt.).

The system optimization methodology is based on evolutionary strategies (ES). Basically, ES is a genetic algorithm which operates in a real solution space as opposed to a binary representation of the solution space employed in genetic algorithm optimizations. The problem is to solve equation 5 which is a representation of a vector function in x . The response surface defined by f may be of any order or complexity. The objective of the ES search algorithms is to identify an optimal solution to equation 4. It must be recognized that for any complex surface, the probability of finding the optimum is less than 1.0. What has been shown is that optimization techniques founded in the biological sciences do a better job of finding the optimal solution than classic linear programming methodologies.

$$y = f(\bar{x}) \quad \text{Eqn. 5}$$

The "ES-chromosome" represents the vector solution to the equation and is represented in equation 6. The two vector

components of the solution c_i are the nominal value associated with the solution vector and a search strategy parameter. Evolutionary strategies possess a characteristic that permits searches to be conducted in preferential directions of an n -dimensional space. These directions evolve during the course of the optimization iterations.

$$c_i = (\overline{op}, \overline{sp}) \quad \text{Eqn. 6}$$

The following equations define the operators which are characteristic of an evolutionary strategies optimization methodology. Parametric or nominal value mutation is defined as:

$$\overline{op}_{mut} = \overline{op} + N_0(\overline{sp}) \quad \text{Eqn. 7}$$

In this expression N_0 represents a normal distribution about $\overline{op}$ with a standard deviation of $\overline{sp}$. The mutation process involves updating the nominal solution with random draws from the normal distribution defined by N_0 .

Strategy mutation is defined by:

$$\overline{sp}_{mut} = \bar{s} \cdot \bar{A} \quad \text{Eqn. 8}$$

$$A_i = \begin{cases} \alpha & E < 0.5 \\ 1/\alpha & E \geq 0.5 \end{cases} \quad \text{Eqn. 9}$$

E is a random number: $E \in (0, 1)$. $\bar{s}$ is the vector of standard deviations used with N_0 , and α has been recommended to be set to 1.3.

The equations delineated represent the optimization algorithm that was implemented for the weapon system optimization section of the analysis algorithms.

Weapon System Capability Optimization. The objectives of the weapon system optimization analysis is the identification of weapon system characteristics that best represent the solution to system needs which compliment the deployed stockpile. The algorithms also possess a capability to explore the broader question of stockpile composition based on delivery system characteristics, treaties and future mission scenarios. The algorithm, based on evolutionary strategies, has the ability to identify yields, inventories and option performance requirements to satisfy a mission.

Mission Analysis Results.

The analysis was executed on subsets of the scenarios and projections defined for this upgrade design effort. The basic subsets of missions consisted of START II scenarios, START III scenarios and sets of strategic reserve force (SRF) mis-

sions. The objectives were to identify potential mission sets for a strategic force in the near future as well as develop scenarios for system utilizations beyond a ten year time frame. Most projections of mission requirements do not explore conditions beyond this short time horizon. The scenarios that were developed were biased toward START III type projections. These scenarios defined inventories and targets that possessed characteristics similar to START II projections but at levels consistent with trends emerging in strategic treaties. The SRF missions did the best job of developing scenarios and provided some of the more stressing system requirements.

The first set of calculations performed for each of the three strategic environments consisted of sets of sensitivity calculations designed to identify the stressing conditions for mission performance of a replacement SL weapon system. The metric chosen to represent system suitability is defined in equation 9. The metric is defined as the standard deviation of the weapon radii assigned to the Mk4 divided by the standard deviation of the mission probability of damage.

$$S = \frac{\sigma_{Wr}}{\sigma_{Pd}} \quad \text{Eqn. 10}$$

This equation can be interpreted as characterizing a system which is robust, in terms of target coverage, and flexible, in terms of an ability to achieve a Pd close to the mission success parameter. The greater the value of S the better the system solution. The search for stressing conditions are parameter settings that drive the metric S to minimal values.

The first part of the effort is to identify combinations of parameters in each of the three subsets of mission which minimize the metric S. These parameters and options become the basis for defining system requirements of the modified system through confirmatory sets of calculations.

Sensitivity Space.

The independent variables identified consisted of

- Operational scenario (Prompt / Delayed).
A prompt scenario reflects "launch on warning" scenarios, while the delayed operational scenario reflects the situation in which we succumb to a first strike before launching a retaliatory strike.
- Inventory.
Inventory consists of the mix and numbers of ICBMs, SLBMs and air delivered assets in the deployed stockpile.
- Strategic reserve force size.
- Mission.
Mission defines the target sets, the inventories, the policy, and the operational scenario.
- Objective probability of damage (Pd)
This parameter is a reflection of the targeteers criteria for defining mission success.
- IC reliability.
- SL reliability.
- Number of replacement SL weapon system options.

Current / START II (Series A). The series A mission captures variations in the START II projections database. The target inventories are the largest in these projections and the weapon system inventories exhibit the broadest spectrum of capabilities. As a result we expect these scenarios to be the least stressing of the three subsets of strategic missions. The sensitivity calculations are structured to provide information indicating the most stressing set of conditions for this mission series. The results of these calculations provide the conditions which will be used in the confirmatory calculations which follow in later sections.

Figure 21. Sensitivity for experiment series A.

The performance metric employed in the assessment is the metric defined by equation 9. Figure 20 is the results of the series A set of analytical experiments. In these figures the vertical extent of each factor represents the sensitivity of that factor on the average of over all the other factors. The conditions which stress the system include: delayed response, inventory 2, mission C (definitions of inventories and missions are discussed in the classified addendum), a 2 boat SRF, "objective probability of damage" of 0.6, IC reliabilities of 0.85 and SL reliabilites of 0.9. The last three parameters including the number of new options are somewhat counter intuitive but may be attributed to confounding effects of other parameters in the analysis. This is also indicated by the change in slope of the parameters. The response surface indicates a degree of non-linearity in the perfor-

mance metric. Each factor in the analysis was monotonically changed and if the dependence of the performance metric were linear we would expect monotonic changes in the metric. Of significance is the fact that all of the parameters exhibit consistently large sensitivities.

START III Environments (Series B). The series B mission set consisted of scenarios and projections that explore conditions associated with START III environments. The retirement of weapon systems and reduction of inventories would tend to drive the requirements for a replacement SL weapon system up. The series B sensitivity analysis utilized an L_{32} orthogonal matrix which allowed for 10 parameters to be studied.

The results of these experiments, Figure 21, indicate that there exists factors which dominate the problem. Mission and objective Pd are the dominant stressing conditions while the number of options provides a significant capability for countering these effects. As with the series A analysis the stressing conditions were identified based on this set of calculations and used in the confirmatory calculations.

Figure 22. Sensitivity for experiment series B.

SRF Force Sensitivity. The SRF sensitivity analysis explored a subset of factors explored in the series A and B sets of conditions. The analysis was limited to four scenarios and situations in which the surviving SRF consisted solely of Mk4/W76 systems. This scenario was selected based on delivery system survivability considerations and on considerations for identifying the most robust set of system design requirements. Assuming a solely Mk4/W76 SRF force approximates a more stressing design environment. The four SRF missions consisted of two former Soviet Union satellite states, a North Korean mission, and a Chinese mission. The sensitivity was limited to mission, objective Pd

and the number of new options. This set of analytical experiments was based on a modified L_8 orthogonal matrix. The modification consisted of combining the first two factors in order to create a four level factor to capture the mission variability.

Figure 23. Sensitivity results for the SRF series of experiments.

The results, Figure 22, indicate that performance of the system is almost solely dependent on the SRF mission.

Confirmatory Analyses and Results.

The final phase of a Taguchi analysis is the confirmatory calculations which bases factor settings on the results of the sensitivity studies. The object of the analysis is to define the system level requirements of a replacement SL weapon system within the context of a strategic force. The sensitivity studies were structured to identify conditions which would lead to the most constrained set of system requirements for the three mission subsets. Again these mission sub-sets consisted of near term projections, moderate range scenarios / projections, and the SRF mission subset.

Current / START II (Series A). The conditions of the confirmatory analysis were delineated in the series A sensitivity section of this note. What follows is a series of plots providing detailed allocation information, stockpile allocation information, convergence data, and the distributions associated with the effective CEP requirements developed for the system. The decision metric is defined by equation 9.

Figures 23 and 24 are representations of the raw data captured in equation 9. The first figure shows the coverage of each weapon system in the inventory in terms of target weapon radius. Information contained in these plots includes the mean, the dot inside the boxes. The inter-quar-

tile range is defined by the box, while the whiskers define the limits of the standard span, i.e. $1.5 \times$ inter-quartile range. Points delineated beyond the whiskers are the outliers. These figures provide regimes of weapon system responsibility given specific system characteristics. The indices on the vertical axis correlate to the weapon system within the strategic inventory. Systems 1 and 2 are the SLs, 3,4 and 5 would represent the ICs and the remaining the air delivered assets. Systems 3 and 4 are unallocated in the cases depicted in the figures.

Figure 24. Weapon radius range of targets assigned to weapon systems.

Figure 23 delineates the assigned area of responsibility, the next figure characterizes the capability of the system on the assigned area of responsibility. A infinitely capable system would result in a simple vertical line at a level equivalent to the objective Pd value. As we can see from the figure none of the systems (those with statistically significant allocation)

tions) exhibit this characteristic.

Confirmatory (EXP A)

Figure 25. Performance range of system against targets allocated.

Figure 25 captures the allocation statistics for all systems within the strategic inventory, including the newly optimized Mk4/W76 system. From the figure we see that for the series A mission subset 65-70 percent of the mission is being covered by the SL branch of the triad.

Stockpile Allocation (Exp. A)

Figure 26. Deployed stockpile allocation for mission scenario A.

The mission series A is a weapon rich scenario. Given the optimized capabilities of the replacement SL weapon system, this system is an extremely import component in a strategic weapons mix. The effective CEPs for the system are defined in the lower left panel of Figure 26. Five options

evolved which had targets assigned to them.

Figure 27. Required effective CEPs and associated distributions.

Based on the two previous figures we see that nearly 50% of the mission could be successfully attacked by a Mk4 system capability of 2.5. Similarly, less than 14% would require a Mk4 system with a 1.0 normalized capability.

A concern when using biologically inspired optimization algorithms is the probability of convergence to a maximum in the response space. Figure 27 shows that we indeed found an optimum early in the calculational series. We can also see that for this series in the mission subsets, we approached a relatively high mission fitness, greater than 1.9 of a theoretical maximum of 3.0.

Figure 28. Convergence information for experiment series A.

START III Environments (Series B). The series B mission subset is a target rich scenario. In this case some targets will survive uncovered by a one-to-one targeting philosophy. The figures which follow provide identical information to the information generated for the series A mission subset. The first point to notice with these calculations are the diminished responsibility assigned to the replacement SL weapon system. This is potentially due to a number of factors including the uncovering of soft targets because of priority considerations in a target rich environment, and a different weapons mix resulting from treaty considerations. A significant number of second order factors come into play some of which are captured in the allocation and optimization algorithms.

Confirmatory (EXP B)

Figure 29. Weapon radius range of targets assigned to weapon systems.

Confirmatory (EXP B)

Figure 30. Performance range of system against targets allocated.

The figures which capture the essence of the roles and requirements are shown in Figures 30 and 31. We see that

given the strategic inventories associated with this mission subset, we still see over half the mission being covered by the SL branch of the triad. There is however a diminished role of the Mk4/W76 in the allocations, less than 40%.

Figure 31. Deployed stockpile allocation for mission scenario B.

As in the case of the series A mission subset, the recommended set of effective CEPs range from 1.0 to 2.5 normalized feet, the limits explored by the optimization algorithms. We find in this case however that only 25% of the mission requires the largest effective CEP and approximately only 8% requires the tightest effective CEP.

Figure 32. Required effective CEPs and associated distributions.

As with series A, the convergence to a solution occurred early in the calculational series. The solution converged to a

level lower than that of the first series in large part to the uncovered targets which are considered to contribute a value of zero (a penalty) to the fitness functions.

Figure 33. Convergence information for experiment series B.

SRF Force Confirmatory. The SRF mission subset consisted of 4 missions generated from target databases on regional missions. SRF missions are assumed to exist under conditions of limited weapon system inventories, in this case only M4/W76 weapon systems, and limited target sets. The sensitivity analysis conducted was used to identify the most stressing environment for the Mk4 system, the conditions identified were used in the SRF confirmatory calculations. The results of these calculations are defined in the next plot.

Figure 34. SRF effective CEP requirements and associated distribution.

We find under these conditions that a system exhibiting max-

imum capabilities will be employed 95% of the time while the lowest capability would be used only 5% of the time. The implications are that the SRF mission, based on the 4 case subset, will not be a system requirements driver in this design effort.

Exploratory Analysis.

The tool possesses the capability to optimize on yields, inventories as well as accuracy constraints. With this capability we also have the ability to explore the optimal configuration of the strategic stockpiles. An exploratory set of calculations were performed to find the optimal stockpile given a mission with a distribution of targets comparable to a current mission, "Scenario A" of our mission set. The sensitivity study explored the effects of objective Pd, SRF force size, reliabilities of the delivery systems, as well as the effects of the number of potential yields and number of options(effective CEPs) of the new systems. The results of those analyses are provided in figure 34.

Figure 35. Stockpile optimization sensitivity assessment.

The number of options per system is the most important design parameter with yields and SRF force size being the next most important design and operational parameters. The

confirmatory calculation produced the following results.

Figure 36. Yield distribution for an optimized stockpile.

The yield distribution requirements are presented in Figure 35. The optimal stockpile would contain five basic systems with yields ranging from 25 to 500 Kt. The corresponding option requirements are provided in Figure 36. The systems are numbered from lower left to upper right, 1-6. The two 25 kt systems correspond to 2 and 6 in the figure. Of the remaining systems, system 1 is a 431 kt system, system 3 is 167 kt, system 4 is at 500 kt and system 5 is at 302kt.

Figure 37. Effective CEP requirements for each system in the optimized inventory.

Decision Issues.

The following table summarizes the system level requirements and the percent of target coverage associated with the

option for the three mission subsets. The percentages in the headers is the percent of targets assigned to the Mk4 weapon system.

Table 4: Confirmatory analysis results.

Series A 58%		Series B 39%		SRF 100%	
1.0	11%	1.0	6%	1.0	4%
1.8	4%	1.03	3%	2.5	96%
2.13	2%	1.79	4%		
2.23	1%	2.18	2%		
2.5	40%	2.5	24%		

At the smallest normalized effective CEPs, series A and B, the option is used on approximately 10% of the targets, while for the largest effective CEP, the system is used on 24 to 40% of the series A or B targets.

Follow-on Activities

The results delineated in earlier sections provide information for use in the system development effort. The remainder of the section provides the context in which the data is used. One of the responsibilities of the systems engineer is to provide transformations between system level requirements and the sub-system level metrics used to assess the performance of the sub-system. In the case of this effort the sub-system design engineers are using Monte-Carlo techniques to determine the probability of kill (P_k) of a concept against a target set. The transformation between effective CEP requirements and concept P_k is defined by equation 2.

A re-statement of the development activity and the information generated follows: **Scenario development** provides mission details including weapon system inventories, target distributions, range distributions, and mixes of delivery platforms. **Range distributions** provide weightings of pre-reentry covariances. **Target distributions and locations** provide target clustering distributions which enable the analyst to assess foot printing capabilities as functions of warhead weight and yield. **Weapon system optimization** provides information delineating the subset of mission targets allocated to the new / modified weapon system concept. This effort also generates the effective CEP requirements for the system and fuzing options.

Recommendations for further Development.

The next couple of points capture considerations for further work or efforts that should be considered in future development efforts. The two basic issues concern the mission

developments, and the technology for completing the requirements development process in a manner that is 1) technically feasible, and 2) more attuned to classic systems engineering development approaches.

Trend Analysis of Missions.

The scenario development considered a static target database for use in defining future missions. This approach is a reasonable first order approach to mission analysis. The problem with this approach is it lacks consideration of trends in asset protection. We have seen from historical evidence that targets are not static but under go design modifications that in some cases result in diminished vulnerabilities, and / or greater location uncertainties. Targets are deployed in deeper bunkers, the bunkers are better engineered, or take better advantage of the natural terrain and geology. Taking these types of dynamics into consideration might have resulted in modified target distributions from which the missions were developed.

Targeting Rule Base Issues.

Some effort should be expended to explore the use of genetic programming techniques to find the rule set which most closely approximates the current targeting philosophy of strategic targeteers. The rule set used in this study is based on a best guess, based on generic targeting considerations. The fuzzy approach was employed to provide a degree of flexibility for the exploration of policy issues associated with strategic targeting. In order to explore these excursions, a rule set needs to be identified that exhibits a high degree of fidelity with current policy.

There are two basic methodologies useful for developing target allocation rules. The first is to follow an evolutionary approach similar to the process that resulted in the current targeting heuristics. It is suggested that this evolutionary approach be pursued in order to make the process of capturing complex non-linear dependencies tractable. The second more sophisticated approach is to write a wrapper for the allocation algorithms in which a genetic algorithm (GA) employing cyclic permutations searches for combinations of fuzzy operators and terminals to identify an optimal target rule set. This approach requires a known solution to act as the fundamental training set.

C4ISR Projections

A replacement SL weapon system will remain in the deployed stockpile for 30 plus years. Command and control development efforts are likely to result in capabilities that could permit re-targeting of a strategic weapon system in real time. This requires that target recognition assets have evolved capabilities sufficient to assess the status of a silo or the location of re-locatable assets and could securely com-

municate this information to the missile / bus system. With stockpile size trends resulting from treaties, the value of the assets continues to increase.

The technology which may make real time targeting possible revolves around the concept of cyber-agents. The possibility exists for the agents to orchestrate the identification, location and projected location of strategic targets at times consistent with the speed of a SRT and the time from release to impact and provide that information to targeting agents which could correct preset conditions within the constraints of the bus energy. What might the implications to system performance be under these conditions. We no longer would have to shoot at potentially hard empty holes, but could engage targets with a high degree of location uncertainty.

References.

Cox, E., The Fuzzy Systems Handbook, AP Professional, Cambridge MA, 1994, ISBN 0-12-194270-8 .

Dr. Christian Jacob; "Evolutionary Algorithms with Mathematica", Tutorial at Genetic Programming Conf.

1997, Stanford U.

Defense Intelligence Agency, "Physical Vulnerability Handbook for Nuclear Weapons", OGA-2800-23-92

Klir, G.J., Folger, T.A., Fuzzy Sets, Uncertainty & Information, Prentice Hall, 1988, ISBN 0-13-345984-5

R.R. Yager; "Multiple objective decision-making using fuzzy sets", Iona College, Int J Man-Machine Studies, 1977 9,375-382.

M.E. Senglaub, "Information Addendum To Weapon System Requirements Analysis", to be published.

This work was supported by the United States Department of Energy under Contract

DE-AC04-94AL85000.

APPENDIX A Transformation between category codes and descriptive parameters.

Table 5: Category code and variable transformation with basic rule set. (Descriptions can be found in the classified addendum)

CatCode	Description	Counter_lethality	Mission	Function	HOB	Timing	Weight
21100		20	50	85	0	1774.77	0.496094
21300		10	50	85	0	1785.28	0.496094
21500		10	50	85	0	1785.28	0.496094
41100		85	10	45	162.906	64.9121	0.59863
41400		80	10	45	164.151	123.194	0.59863
42100		10	50	90	0	1785.3	0.496094
42200		10	50	90	0	1785.3	0.496094
42600		10	50	90	0	1785.3	0.496094
42700		10	50	90	0	1785.3	0.496094
45100		20	50	90	0	1776.07	0.496094
45300		20	50	90	0	1776.07	0.496094
45500		10	40	85	0	1687.43	0.566804
45900		30	40	35	1979.33	1388.18	0.339355
60300		80	5	135	0.0310793	350.962	0.851451
60400		85	5	130	0.0260747	358.633	0.850451
60500		80	5	35	1926.85	349.161	0.543991
61100		20	15	10	2056.13	1748.24	0.344194
63200		15	15	15	1995	1746.39	0.403487
63400		10	15	15	1995	1747.89	0.403487
63500		10	15	15	1995	1747.89	0.403487
64400		10	15	15	1995	1747.89	0.403487
66100		75	30	135	0	365.458	0.818808
66200		75	35	130	0	364.996	0.807953
66500		75	30	135	0	365.458	0.818808
66600		75	35	130	0	364.996	0.807953
67200		90	15	30	2034.09	359.038	0.451828
68000		30	50	10	2056.13	1684.75	0.354194
68100		20	45	10	2056.13	1774.25	0.333109
68200		20	45	10	2056.13	1774.25	0.333109
68300		15	45	10	2056.13	1774.25	0.333109
68400		20	45	10	2056.13	1774.25	0.333109
68500		15	50	15	1995	1783.65	0.39039
68600		20	50	15	1995	1774.77	0.39039
68700		15	50	15	1995	1783.65	0.39039
74100		90	50	50	142.745	44.7661	0.580098
76000		90	50	50	142.745	44.7661	0.580098
80000		95	15	110	0.059728	126.551	0.881386
80050		90	30	110	0	359.217	0.818808
80100		70	40	65	0	363.039	0.566804
81100		90	10	55	160.759	44.7661	0.602328
81200		85	10	55	160.759	64.9121	0.593928

Table 5: Category code and variable transformation with basic rule set. (Descriptions can be found in the classified addendum)

CatCode	Description	Counter lethality	Mission	Function	HOB	Timing	Weight
81300		80	25	55	164.699	356.11	0.588974
81400		60	25	55	164.699	360.824	0.588974
82200		85	70	110	0	64.9121	0.851482
82400		85	70	110	0	64.9121	0.851482
82600		85	70	110	0	64.9121	0.851482
83200		85	5	50	142.745	64.9121	0.584779
84100		85	10	55	160.759	64.9121	0.593928
84500		90	70	45	168.072	44.7661	0.624703
85100		90	70	110	0	44.7661	0.851482
85300		95	70	110	0	41.6943	0.851482
86100		70	45	90	0	1661.97	0.516997
86200		70	55	90	0	1661.97	0.516997
86900		70	55	90	0	1661.97	0.516997
87100		95	5	110	0.022495	41.6943	0.88307
87200		90	70	110	0	44.7661	0.851482
87400		90	5	50	142.654	44.7661	0.584779
87600		80	10	35	1926.85	349.576	0.522452
87800		95	5	50	142.407	41.6943	0.585192
87900		70	45	80	0	331.556	0.516997
88100		95	5	110	0.022495	41.6943	0.88307
89100		95	50	50	142.745	44.4562	0.584356
89200		90	50	50	142.745	44.7661	0.580098
89500		85	50	50	142.745	64.9121	0.566483
89700		80	50	55	164.699	127.29	0.548832
91000		55	30	55	164.699	365.186	0.627476
92000		50	55	90	0	1457.79	0.496992
95100		50	50	75	0	387.007	0.496094
95200		90	20	110	0.204469	293.47	0.858072
96100		50	30	55	164.699	365.186	0.627476
96200		40	30	55	164.699	867.286	0.627476
96300		30	30	55	164.699	1254.39	0.627476
97200		40	30	85	0	1330.21	0.818808

APPENDIX B Experimental configurations for validation calculations.

The prompt operational scenario reflects a launch on warning philosophy of suffering a first strike and then retaliating with the remaining philosophy, while the delayed response reflects a philosophy of suffering a first strike and then retaliating with the remaining

Table 6: Sensitivity analysis setups.

Exp. No.	Plan	Objec Pd	Iter.	Accept	IC Relia.	SL Relia.	Defuzz	Cat Code
1	Prompt	0.6	100	1.0	0.7	0.8	centroid	brad
2	Prompt	0.6	175	1.5	0.85	0.9	ave max	mike
3	Prompt	0.6	250	2.0	1.0	1.0	max	brad
4	Prompt	0.75	100	1.0	0.85	0.9	max	brad
5	Prompt	0.75	175	1.5	1.0	1.0	centroid	brad
6	Prompt	0.75	250	2.0	0.7	0.8	ave max	mike
7	Prompt	0.9	100	1.5	0.7	1.0	ave max	brad
8	Prompt	0.9	175	2.0	0.85	0.8	max	brad
9	Prompt	0.9	250	1.0	1.0	0.9	centroid	mike
10	Delay	0.6	100	2.0	1.0	0.9	ave max	brad
11	Delay	0.6	175	1.0	0.7	1.0	max	mike
12	Delay	0.6	250	1.5	0.85	0.8	centroid	brad
13	Delay	0.75	100	1.5	1.0	0.8	max	mike
14	Delay	0.75	175	2.0	0.7	0.9	centroid	brad
15	Delay	0.75	250	1.0	0.85	1.0	ave max	brad
16	Delay	0.9	100	2.0	0.85	1.0	centroid	mike
17	Delay	0.9	175	1.0	1.0	0.8	ave max	brad
18	Delay	0.9	250	1.5	0.7	0.9	max	brad

APPENDIX C **Inventories Used in current, START II, and START III scenarios.**

See Classified Addendum for table of weapon system inventories. "Information Addendum To Weapon System Requirements Analysis", to be published.

APPENDIX D Experiment A, current and START II environments.

Table 7: Taguchi setup for Current and Start II scenarios, L_{18} matrix.

Exp No.	Ops	Inventory	SRF (Boats)	Mission	Obj Pd	Reliability IC	Reliability SL	No. New Options
1	Prompt	1	1	A	0.6	0.7	0.8	2
2	Prompt	1	2	B	0.75	0.85	0.9	4
3	Prompt	1	3	C	0.9	1.0	1.0	6
4	Prompt	2	1	A	0.75	0.85	1.0	6
5	Prompt	2	2	B	0.9	1.0	0.8	2
6	Prompt	2	3	C	0.6	0.7	0.9	4
7	Prompt	3	1	B	0.6	1.0	0.9	6
8	Prompt	3	2	C	0.75	0.7	1.0	2
9	Prompt	3	3	A	0.9	0.85	0.8	4
10	Delayed	1	1	C	0.9	0.85	0.9	2
11	Delayed	1	2	A	0.6	1.0	1.0	4
12	Delayed	1	3	B	0.75	0.7	0.8	6
13	Delayed	2	1	B	0.9	0.7	1.0	4
14	Delayed	2	2	C	0.6	0.85	0.8	6
15	Delayed	2	3	A	0.75	1.0	0.9	2
16	Delayed	3	1	C	0.75	1.0	0.8	4
17	Delayed	3	2	A	0.9	0.7	0.9	6
18	Delayed	3	3	B	0.6	0.85	1.0	2

APPENDIX E Experiment B, START III environments.

Table 8: Taguchi setup for START III scenarios, L'_{32} matrix.

Exp No.	1 Ops	2 Inventory	3 SRF (boats)	4 Mission	5 Obj Pd	6 Reliability IC	7 Reliability SL	8 No. New Options	9 Acc. of IC (Un-Modeled)
1	Prompt	11	0	D	0.6	0.7	0.7	2	0.7
2	Prompt	11	1	E	0.7	0.8	0.8	4	0.8
3	Prompt	11	0	F	0.8	0.9	0.9	6	0.9
4	Prompt	11	1	G	0.9	1.0	1.0	8	1.0
5	Prompt	12	0	D	0.7	0.8	0.9	6	1.0
6	Prompt	12	1	E	0.6	0.7	1.0	8	0.9
7	Prompt	12	0	F	0.9	1.0	0.7	2	0.8
8	Prompt	12	1	G	0.8	0.9	0.8	4	0.7
9	Prompt	13	0	E	0.8	1.0	0.7	4	0.9
10	Prompt	13	1	D	0.9	0.9	0.8	2	1.0
11	Prompt	13	0	G	0.6	0.8	0.9	8	0.7
12	Prompt	13	1	F	0.7	0.7	1.0	6	0.8
13	Prompt	14	0	E	0.9	0.9	0.9	8	0.8
14	Prompt	14	1	D	0.8	1.0	1.0	6	0.7
15	Prompt	14	0	G	0.7	0.7	0.7	4	1.0
16	Prompt	14	1	F	0.6	0.8	0.8	2	0.9
17	Delayed	11	0	G	0.6	1.0	0.8	6	0.8
18	Delayed	11	1	F	0.7	0.9	0.7	8	0.7
19	Delayed	11	0	E	0.8	0.8	1.0	2	1.0
20	Delayed	11	1	D	0.9	0.7	0.9	4	0.9
21	Delayed	12	0	G	0.7	0.9	1.0	2	0.9
22	Delayed	12	1	F	0.6	1.0	0.9	4	1.0
23	Delayed	12	0	E	0.9	0.7	0.8	6	0.7
24	Delayed	12	1	D	0.8	0.8	0.7	8	0.8
25	Delayed	13	0	F	0.8	0.7	0.8	8	1.0
26	Delayed	13	1	G	0.9	0.8	0.7	6	0.9
27	Delayed	13	0	D	0.6	0.9	1.0	4	0.8
28	Delayed	13	1	E	0.7	1.0	0.9	2	0.7
29	Delayed	14	0	F	0.9	0.8	1.0	4	0.7
30	Delayed	14	1	G	0.8	0.7	0.9	2	0.8
31	Delayed	14	0	D	0.7	1.0	0.8	8	0.9
32	Delayed	14	1	E	0.6	0.9	0.7	6	1.0

Inventory Scaling

- D - 0.345
- E - 0.388
- F - 0.364

• G - 0.364

APPENDIX F SRF experimental setup.

Table 9: Taguchi setup for optimized stockpile scenario, L₁₆ matrix.

Exp No.	1 Obj Pd	2 Air Relia	3 SRF	4	5	6	7	8 No. CEPs	9	10 IC Relia	11	12 SL Relia	13	14	15 No. Yields
1	0.7	0.7	0					3		0.7		0.7			3
2	0.7	0.7	0					6		0.9		0.9			6
3	0.7	0.7	0					3		0.7		0.9			6
4	0.7	0.7	0					6		0.9		0.7			3
5	0.7	0.9	15%					3		0.9		0.7			6
6	0.7	0.9	15%					6		0.7		0.9			3
7	0.7	0.9	15%					3		0.9		0.9			3
8	0.7	0.9	15%					6		0.7		0.7			6
9	0.9	0.7	15%					3		0.7		0.7			6
10	0.9	0.7	15%					6		0.9		0.9			3
11	0.9	0.7	15%					3		0.7		0.9			3
12	0.9	0.7	15%					6		0.9		0.7			6
13	0.9	0.9	0					3		0.9		0.7			3
14	0.9	0.9	0					6		0.7		0.9			6
15	0.9	0.9	0					3		0.9		0.9			6
16	0.9	0.9	0					6		0.7		0.7			3

APPENDIX G Sample input data set.

Sk_Confirm	<u>Name of the file</u>		
ScaledInv 0.357	<u>Scaling of inventories to</u>	25 500	
	<u>match the ratio of targets to</u>	100 850	
	<u>weapons</u>	50 500	
Weapons 0 6	<u>Old and new weapons</u>	50 500	
		50 500	
Options 0 6	<u>Old and new options</u>	50 500	
SysRelia 0.9	<u>System reliability</u>	50 500	
Response 120 10 2400	<u>Delivery system</u>	50 500	
	<u>characteristic timing</u>		
eoi		25 500	
		250 1275	
Options 0 6 SysRelia 0.8 Response 45 -3 60 eoi		100 500	
Options 0 6 SysRelia 0.7 Response 1200 75 3000 eoi		100 500	
		100 500	
Options 0 6 SysRelia 0.9 Response 120 10 2400 eoi		100 500	
Options 0 6 SysRelia 0.8 Response 45 -3 60 eoi		100 500	
Options 0 6 SysRelia 0.7 Response 1200 75 3000 eoi		100 500	
Population 25	<u>ES population size</u>	25 500	
Convergence 400	<u>Convergence iterations</u>	0 425	
Mutation 0.125	<u>Mutation probability</u>	200 500	
ObjectivePD 0.75	<u>Mission success paramter</u>	200 500	
Conv_PPM 10000	<u>Convergence criteria in PPM</u>	200 500	
Debug	<u>Generates extra output data</u>	200 500	
Constraint 48	<u>Total number of parameters</u>	200 500	
	<u>being optimized; (6 systems</u>	200 500	
	<u>with 6 yields, 6 inventories,</u>		
	<u>and 6 options. 6*(1+1+6)</u>	25 500	
0.5	<u>Scale factor for evolutionary</u>	100 850	
	<u>strategy search parameter</u>	50 500	
25 500	<u>Range for yield search</u>	50 500	
250 1275	<u>Range of inventory levels</u>	50 500	
100 500	<u>Range of option capabilities</u>	50 500	
100 500	...	50 500	
100 500		50 500	
100 500			
100 500		Scenarios 1	<u>Number of scenarios</u>
100 500			<u>in the optimization</u>
25 500		6 1 2 3 4 5 6	<u>Number and which</u>
0 425			<u>warheads are in each</u>
200 500			<u>scenario</u>
200 500		Scenario.A	<u>File name of the target</u>
200 500			<u>database used in</u>
200 500			<u>optimization</u>
200 500		eoi	
200 500			

APPENDIX H Table of contents for code listings.

+++++ Main Weapon System Allocation +++++	40
int main(int argc, char *argv[])	40
+++++ Weapon System Optimization (WeaponOpt) +++++	41
class WeaponOpt	44
WeaponOpt::WeaponOpt()	45
ostream& operator<<(ostream& out, WeaponOpt& A)	46
void WeaponOpt::Dump_WeaponOpt(ostream& Dump_out)	47
void WeaponOpt::Load_Data(istream& in)	48
void WeaponOpt::Set_Allocation(float* Means)	52
float WeaponOpt::WS_Fitness(int iter, int popu, ostream& Dump_out)	52
float WeaponOpt::WS_Fitness_Validate(int iter)	53
void WeaponOpt::WS_Opt()	54
void WeaponOpt::File_Adj(fstream& FL_b, streampos f_strt, streampos f_end)	58
+++++ Main Weapon System Optimization +++++	60
int main(int argc, char *argv[])	60
+++++ Weapon System Allocation (Targeting) +++++	61
class Targeting	64
Targeting::Targeting()	65
istream& operator>>(istream& in, Targeting& adum)	66
ostream& operator<<(ostream& out, Targeting& A)	68
float Targeting::Dist_Gamma(float alp, float bet, float x)	68
float Targeting::Dist_Sigmoid(float alp, float bet, float gam, float x)	68
void Targeting::Dump_suitability()	69
void Targeting::Dump_Targeting(ostream& out)	70
void Targeting::IC_Alloc()	71
float Targeting::Fitness(int targ_indx, int sys_indx, float *cep,	73
float yld, int fz_opts)	73
float Targeting::Next_Iteration()	74
float Targeting::Obj_HOB(int targ_indx, int sys_indx, int opt_indx)	76
float Targeting::Obj_Pk(int targ_indx, float cep, float yld)	77
float Targeting::Obj_Time(int targ_indx, int sys_indx)	77
float Targeting::Obj_Yield(float yld)	79
void Targeting::Perf_Results()	79
float Targeting::Pssk(int targ_indx, float cep, float yld)	82
void Targeting::Storage_Setup()	82
void Targeting::Suitability_Setup()	83
float Targeting::Targeting_Opt()	84
+++++ Genetic Algorithym Optimization (GA_Alloc) +++++	88
class GA_Alloc	88
GA_Alloc::GA_Alloc()	89
istream& operator>>(istream& in, GA_Alloc& adum)	89
ostream& operator<<(ostream& out, GA_Alloc& A)	90
float GA_Alloc::GA_Alloc_Fitness(float *data)	90
int GA_Alloc::GA_Xchange(int jj, int *suit_set, int indx)	92
int GA_Alloc::GA_Mutate(int jj)	92

int GA_Alloc::GA_Reproduce(int jj)	93
void GA_Alloc::Init_GA_Alloc()	93
float GA_Alloc::Next_Gen_GA_Alloc(float **suit, float acceptance)	94
void GA_Alloc::Setup_GA_Alloc(long seed_val)	95
++++ Evolutionary Strategy Opt (ES_Chromosome) ++++	96
class ES_Chromosome	96
ES_Chromosome::ES_Chromosome()	97
istream& operator>>(istream& in, ES_Chromosome& adum)	97
ostream& operator<<(ostream& out, ES_Chromosome& A)	98
void ES_Chromosome::Setup_ES_Chromo(float** ml, float** sl)	99
float ES_Chromosome::ES_Chromo_Fitness(float *data)	100
float ES_Chromosome::Next_Gen_ES_Chromo()	102
float ES_Chromosome::Mutate()	103
float ES_Chromosome::Xover()	104
class Allocation	106
Allocation::Allocation()	107
istream& operator>>(istream& in, Allocation& adum)	108
ostream& operator<<(ostream& out, Allocation& A)	109
float Allocation::Allocation_Opt()	110
float Allocation::Allo_Fitness(int *ga_soln, int *cep_index, float *suit)	111
float Allocation::Fitness_PD(int targ_indx, float *cep, float yld, int fz_opts)	113
float Allocation::Fitness_Stk_Lmt(int inv_num)	113
float Allocation::Fitness_Time(int targ_indx)	114
float Allocation::Fitness_Wt(int targ_indx)	114
float Allocation::Fitness_Yield(float yld)	114
void Allocation::Genetic_Setup()	115
void Allocation::IC_Alloc()	115
void Allocation::Perf_Results(int soln_index)	117
float Allocation::Pssk(int targ_indx, float cep, float yld)	118
float Allocation::Obj_Pk(int targ_indx, float cep, float yld)	119
void Allocation::Gen_Dump(ostream& out)	119
void Allocation::Gen_Restart(istream& in)	120
+++++ Mission Definition (Mission_) +++++	122
class Mission_	122
Mission_::Mission_()	125
istream& operator>>(istream& in, Mission_& t)	125
ostream& operator<<(ostream& out, Mission_& t)	126
int Mission_::Adj_vn(int t_vn, int t_k, float yld)	127
float Mission_::Brode_84(float xt, float yt)	127
float Mission_::Brode_DP(float xt, float yt)	129
void Mission_::CatCode_Trans()	129
float Mission_::CR_calc(int t_vn, int t_k, float yld)	130
float Mission_::DP_calc(int t_vn, int t_k, float yld)	130
void Mission_::Dump_Mission_(ostream& out)	131
float Mission_::Gr_coupling(float hob, float yld)	131
float Mission_::Relative_dist(int i, int j)	132

float Mission_::OP_calc(int t_vn, int t_k, float yld)	132
void Mission_::OP_contour(float yld)	133
void Mission_::Vul_Setup(int targ_indx, float yld)	134
float Mission_::WR_cr(float yld, float sig, float HOB)	136
float Mission_::WR_dp(float yld, float sig, float HOB)	136
float Mission_::WR_gvn(int targ_indx, float yld, float HOB)	138
float Mission_::WR_op(float yld, float sig, float HOB)	139
+++++ Random Number operations (randint) +++++ -----	141
class randint	141
void getx()	141
randint(long s = 1000)	141
void seed(long s)	141
long get_seed()	141
int draw()	141
float fdraw()	142
int exp_dist()	142
void exp_dist_Rep(int tmp)	142
void Setup_exp_dist(int* hist, int siz)	142
void CleanUp_dist()	143
void Swap(float* a, float* b)	143
void Sort(float* A, int sz)	143
float gauss()	143
class WEAPON_	145
WEAPON_::WEAPON_()	145
istream& operator>>(istream& in, WEAPON_& t)	146
ostream& operator<<(ostream& out, WEAPON_& t)	146
void WEAPON_::WEAPON_Load(istream& in)	147
void WEAPON_::WEAPON_Setup(int num_opts)	149
WEAPON_& WEAPON_::operator=(WEAPON_& aWeapon)	149
+++++ Main Fuzzy Analysis +++++ -----	151
int main(int argc, char *argv[])	151
+++++ Fuzzy Decision Algorithms (FzDecision) +++++ -----	152
class FzDecision	152
FzDecision::FzDecision()	155
void FzDecision::Assess_Issues(float* info_in, float* info_out)	156
void FzDecision::DeFuzzify(int ijk)	158
void FzDecision::Dump_Model(char* O_file)	159
void FzDecision::Dump_Results(char* O_file)	160
FzVariable* FzDecision::Find_FzVariable(char* v_nam)	160
void FzDecision::Issue_ID()	161
void FzDecision::Load_Model(char* I_file)	162
void FzDecision::Parse_Rule()	163
void FzDecision::Reset_Var()	165
void FzDecision::Rule_Aggregation(FzVariable* A, int aggr_op)	166
void FzDecision::Run_Model(char* O_file)	167
void FzDecision::Store_Results(int case_num, ostream& out)	170

void FzDecision::Z_AND()	171
void FzDecision::Z_OR()	171
int FzDecision::Z_Truth()	171
+++++ Fuzzy Hedge Algorithms (Hedge_) +++++ -----	173
class Hedge_	173
Hedge_::Hedge_(Hedge_ *p)	174
void Hedge_::Tran(FzSet& bs)	174
istream& operator>>(istream& in, Hedge_& t)	174
ostream& operator<<(ostream& out, Hedge_& t)	174
+++++ Fuzzy Issue Control (Issue_) +++++ -----	175
class Issue_	175
struct RO_	175
Issue_::Issue_(Issue_ *p)	176
void Issue_::Load_issue(istream& in)	176
void Issue_::Dump_issue(ostream& out)	176
+++++ Fuzzy Rule Operations (Rules_) +++++ -----	177
class Rules_	177
struct RA_	177
Rules_::Rules_(Rules_ *p)	178
void Rules_::Load_rule(istream& in)	178
void Rules_::Dump_rule(ostream& out)	179
+++++ Fuzzy Sets Operations (FzSet) +++++ -----	180
class FzSet	180
FzSet::FzSet(FzSet *p)	182
void FzSet::Help()	182
istream& operator>>(istream& in, FzSet& adum)	182
ostream& operator<<(ostream& out, FzSet& adum)	184
void FzSet::Alfacut_Above(double cut_val)	184
double FzSet::Centroid()	184
double FzSet::Ave_Max()	185
double FzSet::Max_Edge()	186
double FzSet::Degree_of_Truth(double scaler)	186
void FzSet::Fz_Initialize()	187
double FzSet::Get_Max()	187
void FzSet::Normalize_FZS()	187
void FzSet::Reset_Wrk()	188
void FzSet::Setup_Beta()	188
void FzSet::Setup_Custom()	188
void FzSet::Setup_Gauss()	188
void FzSet::Setup_Linear()	189
void FzSet::Setup_Pi()	189
void FzSet::Setup_Sigmoid()	190
void FzSet::Setup_Tri()	191
void FzSet::Setup_Uniform()	191
+++++ Fuzzy Variable Operations (FzVariable) +++++ -----	193
class FzVariable	193

FzVariable::FzVariable(FzVariable *p)	195
Hedge_* FzVariable::Find_FzHedge(char* v_nam)	195
FzSet* FzVariable::Find_FzSet(char* v_nam)	196
void FzVariable::Help()	196
void FzVariable::Reset_FS()	196
void FzVariable::Variable_Output(ostream& out)	196
void FzVariable::Variable_Setup(istream& in)	197

APPENDIX I Code listing for weapon system allocation models.

```
//     +++++ Main Weapon System Allocation +++++

#include "/home/mesengl/Data_codes/Fuzzy_Targeting/header.h"

#include "/home/mesengl/Data_codes/Fuzzy_Targeting/FzSet.h"
#include "/home/mesengl/Data_codes/Fuzzy_Targeting/FzHedge.h"
#include "/home/mesengl/Data_codes/Fuzzy_Targeting/FzVariable.h"
#include "/home/mesengl/Data_codes/Fuzzy_Targeting/FzRule.h"
#include "/home/mesengl/Data_codes/Fuzzy_Targeting/FzIssue.h"
#include "/home/mesengl/Data_codes/Fuzzy_Targeting/FzDecision.h"

#include "MES_random.h"
#include "MES_ES_opt.h"
#include "MES_Weapon.h"
#include "MES_Mission.h"
#include "MES_Targeting.h"
#include "MES_WS_Opt.h"

//+++++

int main( int argc, char *argv[])

{
cout << "+++++++ Begin Assessment ++++++\n"<<flush;
cout << "\n";

ifstream allo_in( argv[1] );
cout << "Input data file : " << argv[1] << "\n" << flush ;
ofstream d_out("Validate1.dat");

Targeting tst_allo;
allo_in >> tst_allo;

tst_allo.Targeting_Opt( );

d_out << "\n-----\n";
tst_allo.Dump_suitability( d_out );
cout << "\n";
cout << "+++++++ End of Assessment ++++++\n";
};     // Block: 1
```

// ++++++ **Weapon System Optimization (WeaponOpt)** ++++++

// ++++++

//

//

Weapon System Optimization

//

//

+++++

//

//

//

//

//

//

//

//

//

Weapon system optimization involves the search for system configurations, including accuracies, yields, inventories, and the number of fuzing options per configuration which best meets the needs of a strategic mission scenario or sets of scenarios. The weapon system optimization algorithm can be split into three(3) separate computational packages. They include the weapon system optimization code, a weapon system allocation algorithm and a fuzzy logic modeling environment.

//

//

//

//

//

These entities can be operate independent of each other provided a driver routine is written to load relevant data, execute the algorithm, and provide for output/post processing control.

//

//

//

//

//

Tools being brought to bear include, evolutionary strategies, fuzzy logic, and an "annealing" type of allocation optimization technique. The genetic algorithm proved to be unsuitable for optimizing weapon to target allocation.

//

References:

//

//

//

//

//

Dr. Christian Jacob; "Evolutionary Algorithms with Mathematica", Tutorial at Genetic Programming Conf. 1997, Stanford U.

//

//

//

//

//

//

//

//

//

//

//

//

(www.cpsc.ucalgary.ca/~jacob/ or www2.informatik.uni-erlangen.de/~jacob/)

Defense Intelligence Agency, "Physical Vulnerability Handbook for NuclearWeapons", OGA-2800-23-92

Cox, E., *The Fuzzy Systems Handbook*, AP Professional (a div of Harcourt Brace & Co.), 1994, ISBN 0-12-194270-8

Klir, G.J., Folger, T.A., *Fuzzy Sets, Uncertainty & Informtion*, Prentice Hall, 1988, ISBN 0-13-345984-5

R.R. Yager; "Multiple objective decision-making using fuzzy sets", Iona Colledge, Int J Man-Machine Studies, 1977 9, 375-382.

JSTPS TR 91-1, "PDCALC A Computer Program For The Probability Of Damage Calculations", 1991.

H. Brode, S. Speicher, "Air Blast From Nuclear Bursts-analytic Approximations", Pacific-Sierra Research Corp., PSR Report 1419-1.

// S. Speicher, H. Brode, "Extremely High Overpressure Analytic
 // Expression For Burst Height, Range And Time - Over An Ideal
 // Surface", Pacific-Sierra Research Corp., PSR Note 611
 // Capt. G.Martelle, "3DPD HANDBOOK, Formulas and Algorithms for
 // Computing The Probability of Damage of New Generation
 // Nuclear Weapons", AF Operational Test & Evaluation Center.
 //
 // **Code Structure :**
 //

// **ProgramConstruction:**

```
//
//
// The driver for the optimization algorithms is simply a main program
// in which an object is declared, data is loaded, and the optimization
// is performed. An example of a driver for the optimization follows:
```

```
//
//      ifstream weap_in( argv[1] );
//      ofstream d_out("Validate1.dat");
//      WeaponOpt NuSys;
//      NuSys.Load_Data( weap_in );
//      d_out << NuSys << flush ;
//      NuSys.WS_Opt();
//
```

```
//
// The first 2 statements define the input and output files. The third
// line defines the WeaponOpt class object, NuSys.Load_Data loads
// the data which is immediately dumped. NuSys.WS_Opt begins the
// sytem optimization.
```

```
//
// Input Format: (typical dataset)
```

```
//
//      NuMk4
//      Weapons 2 0
//      Yield 333 Inventory 75 Options 0 2 Response 1.0 eoi
//      Yield 111 Inventory 225 Options 4 0 CEPs 450 400 750 350
//      Response 1.0 eoi
//      Constraint 2
//      0.5
//      175 250
//      200 300
//      Population 20
//      Convergence 33
//      ObjectivePD 0.75
//      Mutation 0.05
//      Seed 555
//      Dump
//      Scenarios 1
//      2 1 2
//      target.db
//      eoi
//
```

```
//
// target.db is the data base of mission targets. We use a seperate
// file for this input as it provides greater flexibility when decoupling the
// various system codes. The data in this file looks like:
```

```
//
//      scenario1
//      Dump
//      Convergence 50
```

```

//      Acceptance 1.0
//
//      Targets Targets 250
//      608 41L8 0.7317663603 61.93145 44.1054146 46.25955643
//      465 37M6 0.5177064626 58.98407 101.7777439 179.97161811
//      500 37M6 0.4310704167 51.18029 157.6116195 229.35555875
//      ...
//      eoi
//
//      Input Keywords include:
//      Weapons, Yield, Inventory, Options , CEPs, Response, Constraint,
//      Population, Convergence, ObjectivePD, Mutation, Seed, Dump,
//      Debug, Scenarios, & eoi for termination.
//
//      The Targeting class is the subset of algorithms which perform the
//      weapon to target allocation. Targeting and WeaponOpt must load
//      weapon system characteristics, When we are performing the opti
//      mization function, WeaponOpt loads weapon characteristics and
//      passes this information to the Targeting object. When the alloca
//      tion function is being exercised weapon characteristics are loaded
//      from Targeting.
//
//      WeaponOpt Member Functions:
//
//      Setup, input and output of information:
//      Dump_WeaponOpt, Load_Data, Set_Allocation, File_Adj
//      This last function is used to capture and store the last two iterations
//      of the run.
//
//      Optimization and fitness evaluations are performed in:
//      WS_Fitness, WS_Fitness_Validate, WS_Opt
//
//      ++++++
//      ++++++

```

class WeaponOpt

```

{
public:
//
//      Define addiitonal model parameters:
//
int exists;
int DEBUG, DUMP ;
int PERF_OUT, freq_out;
char* obj_name;

```

```

//
//      An Enumeration Will Be Used To Facilitate Info Transfer To
//      Processors In The MP Implementations
//
enum{ es_pop, max_scenarios, es_iter, total_WH , total_optimize,
      mut_pr, fr, pd, XO_pr, lvec_sz };
int* lvec;
float mutation_pr, Xover_pr;

WEAPON_ *sys;
int** sys2scene;
float** constraint ;
float** constraint_tmp ;
float fraction, pd_objective ;
float Scale_Inv ;

randint random;
long seed_val;
ES_Chromosome soln;
Targeting* scenario;

WeaponOpt( );          // Constructor
//
//      overload the input and output operators
//
friend ostream& operator<<( ostream& out, WeaponOpt& );
//
//      Define addiitonal member functions:
//
void Dump_WeaponOpt( ostream& Dump_out );
void Load_Data( istream& in );
void Set_Allocation( float* );
float WS_Fitness( int, int, ostream& );
float WS_Fitness_Validate( int );
void WS_Opt( );
void File_Adj( fstream&, streampos, streampos );

}; // Block: 2
//      ++++++
//      NOTE: The integers in the constructor must be adjusted to reflect
//      the number of variables and arguments in the model

WeaponOpt::WeaponOpt( )

{
obj_name = new char[10];

```



```

//
//          Define default conditions
//
exists=FALSE;
strcpy( obj_name,"None" );
DEBUG=FALSE;
DUMP=FALSE;
PERF_OUT=FALSE;
freq_out=25;

Scale_Inv = 1.0 ;
lvec = new int[ 10 ];
lvec[ lvec_sz ]=10;
lvec[ es_iter ]=75;
lvec[ es_pop ]=75;
//
// Com 1      Default Is To Search For Fuzing Options
//
lvec[ total_WH ]=0 ;
lvec[ total_optimize ]= 0 ;
lvec[ max_scenarios ]= 0 ;

mutation_pr=0.01 ;
Xover_pr=0.25 ;
pd_objective= 0.75 ;
fraction= 0.25;
lvec[ mut_pr ] = int( 1000*mutation_pr );
lvec[ XO_pr ] = int( 1000*Xover_pr );
lvec[ fr ] = int( 1000*fraction );
lvec[ pd ] = int( 1000*pd_objective );
constraint = new float*[2];
constraint_tmp = new float*[2];
//
//          The Next Set Operations Ensure That Each Processor Will
//          Begin With A Different Random Nuber Seed Value.
//
seed_val=111111;
random.seed( seed_val );
}; // Block: 3
//          ++++++

ostream& operator<<( ostream& out, WeaponOpt& A )

{
//
//          provide coding to represent the ouput desired for this object

```

```

//
out << A.obj_name << "\n";
out << "Convergence " << A.lvec[ A.es_iter ] << "\n";
out << "Population " << A.lvec[ A.es_pop ] << "\n";
out << "\nWeapon Systems \n";
for( int k0=0; k0<=A.lvec[ A.total_WH ]; k0++ )
{
    out << " Master Warhead number: " << k0 << "\n";
    out << A.sys[k0];
}
out << "\nObjectivePD " << A.pd_objective << "\n";
out << "Mutation " << A.mutation_pr << "\n";
out << "\nConstraints: Strategy fraction = " << A.fraction << "\n";
for( int k=0; k< A.lvec[ A.total_optimize ]; k++ )
    out << A.constraint[ 0 ][ k ] << " " << A.constraint[ 1 ][ k ] << " \n";
out << "\nScenarios " << A.lvec[ A.max_scenarios ] << "\n";
for( int k1=0; k1<A.lvec[ A.max_scenarios ]; k1++ )
{
    out << " Scenario number: " << k1 << "\n";
    for( int k2=0; k2<=A.scenario[k1].total_WH ; k2++ )
    {
        out << " Warhead number: " << k2 << "\n";
        out << A.scenario[k1].sys[k2];
    }
}
out << "\n" << flush;
return out;
}; // Block: 4
//

```

void WeaponOpt::Dump_WeaponOpt(ostream& Dump_out)

```

{
//
//      Routine To Dump Information Associated With Weaponopt For
//      Debug And Restart(?).
//
Dump_out << "---- :: WeaponOpt Dump (bgn) :: ---- " << obj_name << "\n";
Dump_out << "---- :: WeaponOpt Dump (---) :: ---- "
    << "Storage Parameters : ( lvec[10] ) \n";
Dump_out << "---- :: WeaponOpt Dump (---) :: ---- ";
    for( int k=0; k<lvec[lvec_sz]; k++ )
        Dump_out << lvec[k] << "\t";
Dump_out << "\n---- :: WeaponOpt Dump (---) :: ---- "
    << "Random Seed : ( seed_val ) \n";
Dump_out << "---- :: WeaponOpt Dump (---) :: ---- " << seed_val << "\n";
}

```

```

Dump_out << "---- :: WeaponOpt Dump (---) :: ----"
    << "Control Parameters : (DEBUG, DUMP, PERF_OUT, freq_out) \n";
Dump_out << "---- :: WeaponOpt Dump (---) :: ----" << DEBUG << "\t"
    << DUMP << "\t" << PERF_OUT << "\t" << freq_out << "\n";

Dump_out << "---- :: WeaponOpt Dump (---) :: ----"
    << "Convergence Control : (mutation_pr, fraction, pd_objective) \n";
Dump_out << "---- :: WeaponOpt Dump (---) :: ----" << mutation_pr << "\t"
    << fraction << "\t" << pd_objective << "\n";

Dump_out << "---- :: WeaponOpt Dump (---) :: ----"
<< "Weapon Parameters : \n";
    for( k=0; k<= lvec[total_WH]; k++ )
        Dump_out << sys[k];
Dump_out << "---- :: WeaponOpt Dump (---) :: ----"
<< "Weapon association to scenario : \n";
    for( k=0; k<lvec[max_scenarios]; k++ )
    {
        Dump_out << "---- :: WeaponOpt Dump (---) :: ----";
        for( int kk=0; kk<=sys2scene[k][0]; kk++ )
            Dump_out << sys2scene[k][kk] << "\t";
        Dump_out << "\n";
    }

if( DUMP )
{
    Dump_out << "---- :: WeaponOpt Dump (---) :: ----"
    << "Chromosome Information : \n";
    Dump_out << soln << "\n";
    Dump_out << "---- :: WeaponOpt Dump (---) :: ----"
    << "Target Information : \n";
    for( k=0; k<lvec[max_scenarios]; k++ )
        scenario[k].Dump_Targeting( Dump_out );
}

Dump_out << "---- :: WeaponOpt Dump (end) :: ---- \n";

return ;
}; // Block: 5
// ++++++

void WeaponOpt::Load_Data( istream& in )

{
    char tmp_in[15];
    char tmp_file[30];

```

```

int Fix_WH, Nu_WH;
exists=TRUE;
float tmp_del, tmp_av;
in >> obj_name;
in >> tmp_in;
while( strcmp( tmp_in,"eoi" ) !=0 )
{
//
//          Output The Keywords As They Are Being Loaded
//
cout << "----- WeaponOpt Keyword : " << tmp_in << "\n" << flush;
if( strcmp( tmp_in,"Weapons" ) == 0 )
{
//
//          Note: for input optimization, sensitivity studies we are allowing
//          for a scaling of the inventories. Thsi can be done is ScaledInv
//          is loaded prior to the weapons.
//
in >> Fix_WH >> Nu_WH ;
lvec[ total_WH ] =Fix_WH + Nu_WH ;
sys = new WEAPON_[ lvec[ total_WH ] +1 ];
for( int k=1; k<= lvec[ total_WH ]; k++ )
{
sys[ k ].WEAPON_Load( in );
sys[ k ].inventory = int( float( sys[ k ].inventory ) *Scale_Inv );
}
}
else if( strcmp( tmp_in,"ScaledInv" ) == 0 )
{
in >> Scale_Inv ;
}
else if( strcmp( tmp_in,"Population" ) == 0 )
in >> lvec[ es_pop ];
else if( strcmp( tmp_in,"Convergence" ) == 0 )
in >> lvec[ es_iter ];
else if( strcmp( tmp_in,"ObjectivePD" ) == 0 )
{
in >> pd_objective ;
lvec[ pd ]= int( 1000*pd_objective );
}
else if( strcmp( tmp_in,"Seed" ) == 0 )
{
in >> seed_val ;
random.seed( seed_val );
}
else if( strcmp( tmp_in,"Mutation" ) == 0 )

```

```

{
    in >> mutation_pr ;
    lvec[ mut_pr ]= int( 1000*mutation_pr );
}
else if( strcmp( tmp_in,"Crossover" ) == 0 )
{
    in >> Xover_pr ;
    lvec[ XO_pr ]= int( 1000*Xover_pr );
}
else if( strcmp( tmp_in,"Constraint" ) == 0 )
{
    in >> lvec[ total_optimize ];
    constraint[0] = new float[ lvec[ total_optimize ] ];
    constraint[1] = new float[ lvec[ total_optimize ] ];
    constraint_tmp[0] = new float[ lvec[ total_optimize ] ];
    constraint_tmp[1] = new float[ lvec[ total_optimize ] ];
    in >> fraction;
    lvec[ fr ]= int( 1000*fraction );
    for( int kk=0; kk<lvec[ total_optimize ]; kk++ )
    {
        in >> constraint[0] [kk] >> constraint[1] [kk];
        tmp_del = constraint[1] [kk] - constraint[0] [kk];
        tmp_av = (constraint[1] [kk] + constraint[0] [kk])/2.0 ;
        constraint_tmp[0] [kk] = 1.0e-4*tmp_del ;
        constraint_tmp[1] [kk] = fraction*tmp_del ;
    }
}
else if( strcmp( tmp_in,"Scenarios" ) == 0 )
{
    in >> lvec[ max_scenarios ];
    scenario = new Targeting[ lvec[ max_scenarios ] ];
    sys2scene = new int*[ lvec[ max_scenarios ] ];
    if( lvec[ total_WH ] <= 0 )
        cerr << "ERROR : Weapons must be declared prior to Optimize; "
        << " Abort and modify input data set \n" << flush;

    for( int k11=0; k11< lvec[ max_scenarios ]; k11++ )
        sys2scene[k11] = new int[ lvec[ total_WH ] +1 ];

//
//      Load In The System To Scenario Correlation Matrix
//      Note: location "0" is the number of WH, remainder is
//      the number of the masterlist WH assigned to the scenario
//

    int k_cor;
    for( int k0=0; k0< lvec[ max_scenarios ]; k0++ )

```

```

    {
        scenario[k0].APPEND_rslts=FALSE ;
        in >> sys2scene[k0][0] ;
        for( int k1=1; k1<= sys2scene[k0][0]; k1++ )
            in >> sys2scene[k0][k1] ;
    }

//
//      Need To Load New Concepts Into Selection Matrices
//

in >> tmp_file;
cout << "---- Begin Senario Load from file : " << tmp_file << "\n" << flush ;
ifstream tmp_bs( tmp_file );
for( int ijk=0; ijk<lvec[ max_scenarios ]; ijk++ )
{
    scenario[ijk].total_WH=sys2scene[ijk][0];
    scenario[ijk].sys= new WEAPON_ [ scenario[ijk].total_WH+1 ];
    for( int k12=1; k12<=scenario[ijk].total_WH; k12++ )
    {
        k_cor= sys2scene[ijk][ k12 ] ;
        scenario[ijk].sys[k12].ceps= new float[ sys[k_cor].total_fz_opt ];
        scenario[ijk].sys[k12].hobs= new float[ sys[k_cor].total_fz_opt ];
        scenario[ijk].sys[k12].relia= new float[ sys[k_cor].total_fz_opt ];
        scenario[ijk].sys[k12].opt_inv= new int[ sys[k_cor].total_fz_opt ];
        scenario[ijk].sys[k12]= sys[k_cor] ;
    }
    tmp_bs >> scenario[ ijk ];
}
}
else if( strcmp( tmp_in,"Debug" ) == 0 )
    DEBUG = TRUE ;
else if( strcmp( tmp_in,"Dump" ) == 0 )
    DUMP = TRUE ;
else
{
    cerr << "Error: (WeaponOpt Input) unacceptable input option : " << tmp_in
    << "\n" << flush;
}; // Block: 6
in >> tmp_in;
}; // Block: 7

int index;
float pr2;
soln.es_pop = lvec[ es_pop ] ;
soln.max_factors = lvec[ total_optimize ] ;
soln.mutation_pr = mutation_pr ;
soln.Xover_pr = Xover_pr ;

```

```

cout << "----- WeaponOpt :: Input Complete\n" << flush;
return;
};    // Block: 8
//      ++++++

void WeaponOpt::Set_Allocation( float* Means )

{
//
//      This routine provides the assessment of solution fitness; ie. how
//      well does the new concept and targeting options perform against
//      a mission spectrum of targets.
//
int k_opt=0 ;
for( int k1=1; k1<=lvec[ total_WH ]; k1++ )
{
if( sys[k1].Yield_opt == TRUE )
{
sys[k1].yield = Means[ k_opt ];
k_opt++;
}

if ( sys[k1].Inv_opt == TRUE )
{
sys[k1].inventory = int( Means[ k_opt ] );
k_opt++;
}

if ( sys[k1].new_fz_opt > 0 )
{
for( int k2=sys[k1].base_fz_opt; k2<sys[k1].total_fz_opt; k2++ )
{
sys[k1].ceps[k2] = Means[ k_opt ];
k_opt++;
}
}
}
return;
};    // Block: 9
//      ++++++

float WeaponOpt::WS_Fitness( int iter, int popu, ostream&
Dump_out )

{
//

```

```

//          This routine provides the assessment of solution fitness; ie. how
//          well does the new concept and targeting options perform against
//          a mission spectrum of targets.
//
float sum_fit=0.0;
float *fit;
int k_cor;
fit = new float[ lvec[ max_scenarios ] ];
for( int k=0; k< lvec[ max_scenarios ] ; k++ )
{
//
//          Need To Load New Concepts Into Selection Matrices
//
scenario[k].total_WH = sys2scene[k][ 0 ];
for( int k1=1; k1<=sys2scene[k][ 0 ]; k1++ )
{
k_cor= sys2scene[k][ k1 ];
scenario[k].sys[k1]= sys[k_cor];
}
if( DUMP )
{
Dump_out << "Dump Iteration : " << iter << "\tScenario : " << k
<< "\tPopulation : " << popu << "\n";
Dump_WeaponOpt( Dump_out );
}
//
//          Begin Optimization
//
seed_val= random.draw( );
scenario[k].RN_seed= seed_val ;
fit[ k ] =scenario[k].Targeting_Opt( );
sum_fit += fit[ k ];
}
//
//          Cleanup Storage
//
delete fit;
return( sum_fit );
}; // Block: 10

//          ++++++

float WeaponOpt::WS_Fitness_Validate( int iter )

{
//

```



```

//          This routine provides the assessment of solution fitness; ie. how
//          well does the new concept and targeting options perform against
//          a mission spectrum of targets.
//
float sum_fit=0.0;
float *fit;
int k_cor;
fit = new float[ lvec[ max_scenarios ] ];
for( int k=0; k< lvec[ max_scenarios ]; k++ )
{
//
//          Need To Load New Concepts Into Selection Matrices
//
//
for( int k1=1; k1<=sys2scene[k][ 0 ]; k1++ )
{
k_cor= sys2scene[k][ k1 ];
scenario[k].sys[k1]= sys[k_cor];
}
//
//          Begin Optimization
//
seed_val= random.draw( );
scenario[k].RN_seed= seed_val ;
fit[ k ] =scenario[k].Targeting_Opt( );
scenario[k].Perf_Results( );
sum_fit += fit[ k ];
}
//
//          Cleanup Storage
//
delete fit ;
return( sum_fit );
};    // Block: 11
//          ++++++

```

void WeaponOpt::WS_Opt()

```

{
//
// Com 2    This Routine Provides Control For The Optimization Of Weapon
//          System Designs. Optimization Defined Within A Context Of
//          Force Structure.
//
//          Note: Scenarios Must Be Set Up And Loaded From This Class.
//          Inventories, Targeting Options, And Missions Are Defined In

```

```

//          The Allocation Class.
//
cout << " ----- WeaponOpt :: ( WS_Opt ) Begin system optimization\n" << flush ;
streampos f_strt, f_end, f_zero ;
fstream Dump_out("WSO_Dump", ios::in|ios::out ) ;
f_zero=0;

float *fit, *raw_fit ;
float *fit_TmSr;
fit = new float[ lvec[ es_pop ] ];
raw_fit = new float[ lvec[ es_pop ] ];
fit_TmSr = new float[ lvec[ es_iter ] ];

int term_crit=FALSE ;
float run_ave=0.0 ;
if( DEBUG )
    G_debug_out << "\n" <<
        "+++++ Weapon System Fitness Debug ++++++\n"
        << flush;
soln.Setup_ES_Chromo( constraint, constraint_tmp);
//
// Com 3      Begin Iteration To Find Optimal Weapon System Solution.
//
for( int iter=0; iter<lvec[ es_iter ]; iter++ )
    {
//
//          Force Storage Of One Generation Of Weapon To Target Suitability
//
//          Dump File Position Location Identification
f_strt = Dump_out.tellp();

if( iter == (lvec[ es_iter ]-1) )
    PERF_OUT=TRUE;
else
    PERF_OUT=FALSE;
//
// Com 4      Build The Weapon System Attack Options Matrix
//
for( int k=0; k<lvec[ es_pop ]; k++ )
    {
        G_debug_out << "\n\n" << "+++++ Weapon System Iteration: "
        << iter << " Population: " << k << flush;
//
//          Setup And Calculate The Fitness For The Chromo Population
//

```

```

    if( k==0 )
    {
        G_debug_cntl = TRUE ;
        cout << "\n" << "+++++ Weapon System Iteration: "
        << iter << " Population: " << k ;
    }

    Set_Allocation( soln.es_mean[ k ] ) ;
    fit[ k ] = WS_Fitness( iter,k, Dump_out );
    raw_fit[ k ] = fit[ k ] ;

}

//
//          Identiy max and min fitness in order to perform a relative fitness
//          scaling as opposed to an absolute fitness.
//
float mn_fit=9999 ;
float mx_fit=0 ;
for( int k0=0; k0< lvec[ es_pop ]; k0++ )
{
    if( raw_fit[k0] <= mn_fit ) mn_fit=raw_fit[k0] ;
    if( raw_fit[k0] >= mx_fit ) mx_fit=raw_fit[k0] ;
}

//
//          Setup storage for fitness time series analysis.
//          ( convergence criteria ??)
//
float conv_fac=10000.0 ;
fit_TmSr[ iter ] = mx_fit ;
if( abs( run_ave-mx_fit )*conv_fac <= 1.0 )
    if( (mx_fit-mn_fit)*conv_fac <= 1.0 )
    {
        term_crit= TRUE ;
        PERF_OUT=TRUE;
    }
if( run_ave > 0.0000001 )
    run_ave =(run_ave + mx_fit)/2.0 ;
else
    run_ave = mx_fit ;

G_debug_out << "\n+++++ Weapon System Fitness : \t" ;
for( k0=0; k0< lvec[ es_pop ]; k0++ )
{
    fit[ k0 ] =(fit[k0] - mn_fit)* conv_fac ;
    G_debug_out << " ( "<< raw_fit[ k0 ] << " - " << fit[ k0 ] << " )" ;
}

```

```

    }

    if( DEBUG )
    {
        for( k=0; k<lvec[ es_pop ]; k++ )
        {
            G_debug_out << "\nResults, all Chromosomes: ( Iter " << iter << " Pop "
            << k << " )\t";
            G_debug_out << "\t" << raw_fit[ k ];
            for( int kck=0; kck<lvec[total_optimize]; kck++ )
                G_debug_out << "\t" << soln.es_mean[k][kck] ;
        }
    }

//
// Com 5      Transfer Fitness To ES Algorithms
//
    float bs_TMP;
    bs_TMP= soln.ES_Chromo_Fitness( fit );

//
//          Setup A Convergence Monitor Function
//
    if( DEBUG )
    {
        G_debug_out << "\n" << raw_fit[ soln.opt_soln ] << " <--> ";
        for( k=0; k<lvec[ total_optimize ]; k++ )
            G_debug_out << soln.es_mean[soln.opt_soln][k] << " ";
        G_debug_out << flush;
    }

//
// Com 6      Create The Next Generation
//
    soln.Next_Gen_ES_Chromo( );

//
//          Note: The Validation Is Performed On Population Member 0
//          Because Of The Greedy Optimization Algorithm.
//          (Stored Best Soln In Pop Zero)
//
    if( PERF_OUT )
    {
        Set_Allocation( soln.es_mean[ 0 ] );
        fit[ 0 ] = WS_Fitness_Validate( iter );
        cout << "\nFitness Time Series : ";
        for( int kl1=0; kl1<iter; kl1++ )
            cout << " " << fit_TmSr[ kl1 ] ;
    }

//

```

```

//          If The Iterations Have Gone Beyond 0 Then Begin Shifting
//          DUMP Data
//
f_end = Dump_out.tellp();
if( DUMP && iter > 0 )
{
    File_Adj( Dump_out, f_strt, f_end );
}

//
//          Termination criteria associated with time series on fitness reached
//
if( term_crit ) break ;
}

if( DEBUG )
    G_debug_out << "\n" <<
    "+++++ Weapon System Fitness Debug End ++++++\n"
    << flush;

//
//          Cleanup Storage
//
delete fit;
delete raw_fit;
delete fit_TmSr;
Dump_out.close();
return;
}; // Block: 12
//          ++++++
//          This routine moves information from the end of the dump file to
//          the begining of the dump file. The purpose is to store only the
//          last two iterations worth of dump infomration.
//
//          tellp() or tellg() gets the position of the file pointer; while
//          seekp(0 and seekg sets the file position pointer to a value.
//

```

**void WeaponOpt::File_Adj(fstream& FL_b, streampos f_strt,
streampos f_end)**

```

{
    char ch;
    streampos f_1, f_2 ;
    f_1 = 0;
    f_2 = f_strt;
    FL_b.seekg(f_2);
    while( FL_b.tellg() < f_end )

```

```
{  
    FL_b.get(ch) ;  
    f_2 = FL_b.tellp();  
  
    FL_b.seekg(f_1);  
    FL_b << ch ;  
    f_1 = FL_b.tellp();  
    FL_b.seekg(f_2);  
}  
FL_b.seekg(f_strt);  
FL_b << "\n\n" ;  
return ;  
};    // Block: 13
```

APPENDIX J Code listing for weapon system optimization algorithms.

```
//       +++++ Main Weapon System Optimization +++++

#include "/home/mesengl/Data_codes/Fuzzy_Targeting/header.h"
//
//           Global debug
//
ofstream G_debug_out("Z_Global_debug" );
int G_debug_cntl = FALSE;

#include "/home/mesengl/Data_codes/Fuzzy_Targeting/FzSet.h"
#include "/home/mesengl/Data_codes/Fuzzy_Targeting/FzHedge.h"
#include "/home/mesengl/Data_codes/Fuzzy_Targeting/FzVariable.h"
#include "/home/mesengl/Data_codes/Fuzzy_Targeting/FzRule.h"
#include "/home/mesengl/Data_codes/Fuzzy_Targeting/FzIssue.h"
#include "/home/mesengl/Data_codes/Fuzzy_Targeting/FzDecision.h"

#include "MES_random.h"
#include "MES_ES_opt.h"
#include "MES_Weapon.h"
#include "MES_Mission.h"
#include "MES_Targeting.h"
#include "MES_WS_Opt.h"
//       +++++

int main( int argc, char *argv[])

{
cout << "+++++ Begin Assessment +++++\n" << flush;
ifstream weap_in( argv[2] );
cout << "Input data file : " << argv[2] << "\n" << flush ;
WeaponOpt NuSys;
NuSys.Load_Data( weap_in ) ;
//
//           Set a new output file based on info in argv[1]
//
strcpy( NuSys.scenario[0].Results_file, argv[1] ) ;
NuSys.WS_Opt();

for( int k=0; k<NuSys.lvec[NuSys.total_optimize]; k++ )
    cout << "\n" << NuSys.soln.es_mean[0][k] ;
cout << "\n";
cout << "+++++ End of Assessment +++++\n";
};     // Block: 14
```

// ++++++ Weapon System Allocation (Targeting) ++++++

// ++++++

//

//

//

//

//

//

//

//

//

//

//

//

//

//

//

//

//

//

//

//

//

//

//

//

//

//

//

//

//

//

//

//

//

//

//

Weapon System Allocation

This class defines the allocation activities associated with weapon system optimization or for simple weapon to target allocation activities. This constitutes one of the three major calculational activities which comprise weapon system optimization but also a stand-alone sub function.

The algorithm must be able to treat multiple scenarios in support of the system optimization activities. For simple allocations this capability is not required.

References:

Defense Intelligence Agency, "Physical Vulnerability Handbook for Nuclear Weapons", OGA-2800-23-92

Cox, E., *The Fuzzy Systems Handbook*, AP Professional (a div of Harcourt Brace & Co.), 1994, ISBN 0-12-194270-8

Klir, G.J., Folger, T.A., *Fuzzy Sets, Uncertainty & Information*, Prentice Hall, 1988, ISBN 0-13-345984-5

R.R. Yager; "Multiple objective decision-making using fuzzy sets", Iona Colledge, Int J Man-Machine Studies, 1977 9, 375-382.

JSTPS TR 91-1, "PDCALC A Computer Program For The Probability Of Damage Calculations", 1991.

H. Brode, S. Speicher, "Air Blast From Nuclear Bursts-analytic Approximations", Pacific-Sierra Research Corp., PSR Report 1419-1.

S. Speicher, H. Brode, "Extremely High Overpressure Analytic Expression For Burst Height, Range And Time - Over An Ideal Surface", Pacific-Sierra Research Corp., PSR Note 611

Capt. G. Martelle, "3DPD HANDBOOK, Formulas and Algorithms for Computing The Probability of Damage of New Generation Nuclear Weapons", AF Operational Test & Evaluation Center.

Code Structure :

//

//

//

//

//

//

//

//

//

//

//

//

//

//

//

ProgramConstruction:

The driver for the allocation algorithms is simply a main program in which an object is declared, data is loaded, and the optimization is performed. When this algorithm is used in conjunction with the weapon system optimization, control is treated in the **WeaponOpt** class. IN stand-alone mode an example of a driver for the optimization follows:

```

ifstream allo_in( argv[1] );
ofstream d_out("Validate1.dat");
Targeting tst_allo;
allo_in >> tst_allo;
tst_allo.IC_Alloc();
  
```

```

//          tst_allo.Dump_suitability( d_out );
//          tst_allo.Targeting_Opt( );
//
// The first 2 statements define the input and output files. The third
// line defines the Targeting class object, we then load the data via
// the overloaded operator >>. The function call tst_allo.IC_Alloc()
// is the call which presets and defines information for use in the
// allocation effort.. Finally, pre-processed information is dumped to
// file and the optimization algorithm is fired.
//
// Input Format: (typical dataset)
//
//          scenario1
//          Weapons 2 0
//          Yield 333 Inventory 75 Options 0 2 Response 1.0 eoi
//          Yield 111 Inventory 225 Options 4 0 CEPs 450 400 750 350
//          Response 1.0 eoi
//          Dump
//          Convergence 50
//          Acceptance 1.0
//
//          Targets Targets 250
//          608 41L8 0.7317663603 61.93145 44.1054146 46.25955643
//          465 37M6 0.5177064626 58.98407 101.7777439 179.97161811
//          500 37M6 0.4310704167 51.18029 157.6116195 229.35555875
//          ...
//          eoi
//
// Input Keywords include:
// Weapons, Yield, Inventory, Options , CEPs, Response,
// Convergence, ObjectivePD, Mutation, Seed, Dump, Debug,
// & eoi for termination.
//
// Targeting Member Functions:
//
// Setup, input and output of information:
// IC_Alloc, Storage_Setup, Suitability_Setup
// Dump_suitability, Dump_Targeting,
//
// Fitness and Performance evaluations are performed in:
// Fitness, Obj_Time, Obj_Yield, Obj_Pk, Perf_Results, Pssk
//
// Optimization evaluations :
// Next_Iteration, Targeting_Opt
//
//          ++++++

```

```

//      ++++++
class Targeting

{
public:
//
//      Define addiitonal model parameters:
//
int exists;
char* obj_name;
char* Results_file ;

long RN_seed;
int DEBUG, DUMP, FUZZY_TACTIC ;
int APPEND_rslts;
float acceptance, Mission_acceptance, Mission_fitness;

int max_targets ;
int base_line_WH, new_WH, total_WH ;
int total_inventory, no_option;
int Restart_iter, iterations ;
float pd_objective, F1_form_factor, beta[4];
float Scale_Inv ;

WEAPON_ *sys;
Mission_ startX;
randint random;
int **opt_index;
int *allo_vec;
float **suitability;
FzDecision tactic;

Targeting();           // Constructor
//
//      overload the input and output operators
//
friend istream& operator>>( istream& in, Targeting& );
friend ostream& operator<<( ostream& out, Targeting& );
//
//      Define addiitonal member functions:
//
float Dist_Gamma( float, float, float ) ;
float Dist_Sigmoid( float, float, float, float ) ;
void Dump_suitability( );
void Dump_Targeting( ostream& out );

```

```

void IC_Alloc( );
float Fitness(int, int, float*, float, int );// Goal Fitness Function
float Next_iteration( );
float Obj_HOB( int, int, int );
float Obj_Pk( int, float, float );
float Obj_Time( int, int );           // Time Urgency Fitness Function
float Obj_Yield( float );           // Minimum Yield Fitness Function
void Perf_Results( );
float Pssk( int, float, float );     // Determine the probability of "single shot" kill
void Storage_Setup( );
void Suitability_Setup( );
float Targeting_Opt( );             // Opt. The Targeting Of Weapons To Targets

}; // Block: 15

```

Targeting::Targeting()

```

{
    obj_name = new char[10];
    Results_file = new char[ 25 ];
//
//           Define default conditions
//
    exists=FALSE;
    strcpy( obj_name,"None" );
    strcpy( Results_file,"Zresults" );
    RN_seed=1111111111;
    DEBUG=FALSE;
    DUMP=FALSE;
    FUZZY_TACTIC= FALSE ;
    APPEND_rslts=TRUE;
    acceptance=1.25;

    Restart_iter=0;
    iterations=100;

    max_targets=1;
    base_line_WH=1;
    new_WH=0 ;
    total_WH= base_line_WH+new_WH ;
    no_option=0;

    pd_objective=0.75;
    F1_form_factor=0.025;
    Scale_Inv = 1.0 ;
//

```

```

// Com 7      Beta Is The Decision Model Weighting Coeficients.
//
for( int i=0; i<4; i++ )
    beta[i]=1.0;
};    // Block: 16
//      ++++++

istream& operator>>( istream& in, Targeting& adum )

{
    char tmp_in[15];
    char tmp_file[15];
    adum.exists=TRUE;
    int tmp;
    in >> adum.obj_name;
    in >> tmp_in;
    while( strcmp( tmp_in,"eoi" ) !=0 )
    {
//
//      Output Of The Keywords Read For Targeting
//
        cout << "----- Targeting Keyword : " << tmp_in << "\n" << flush;
        if( strcmp( tmp_in,"Targets" ) == 0 )
        {
            in >> adum.startX;
            adum.max_targets = adum.startX.max_targets ;
        }
        else if( strcmp( tmp_in,"Weapons" ) == 0 )
        {
//
//      First load baselines and other Targeting parameters
//      Next load the CEP values and associated stockpile assignments
//
//      Note: for input optimization, sensitivity studies we are allowing
//      for a scaling of the inventories. Thsi can be done is ScaledInv
//      is loaded prior to the weapons.
//
            in >> adum.base_line_WH >> adum.new_WH;
            adum.total_WH = adum.base_line_WH + adum.new_WH;
            adum.sys= new WEAPON_[ adum.total_WH+1 ];
            for( int k=1; k<= adum.total_WH; k++ )
            {
                adum.sys[ k ].WEAPON_Load( in );
                adum.sys[ k ].inventory = int( float( adum.sys[ k ].inventory ) *
                    adum.Scale_Inv );
            }
        }
    }
}

```

```

    }
    else if( strcmp( tmp_in,"ScaledInv" ) == 0 )
    {
        in >> adum.Scale_Inv ;
    }
    else if( strcmp( tmp_in,"Convergence" ) == 0 )
        in >> adum.iterations;
    else if( strcmp( tmp_in,"FuzzyTactics" ) == 0 )
    {
        in >> tmp_in;
        adum.tactic.Load_Model( tmp_in );
        adum.FUZZY_TACTIC= TRUE ;
    }
    else if( strcmp( tmp_in,"Acceptance" ) == 0 )
        in >> adum.acceptance;
    else if( strcmp( tmp_in,"Debug" ) == 0 )
        adum.DEBUG = TRUE ;
    else if( strcmp( tmp_in,"Dump" ) == 0 )
        adum.DUMP = TRUE ;
    else if( strcmp( tmp_in,"ObjectivePD" ) == 0 )
//
//          Objective Mission Performance
//
        in >> adum.pd_objective;
    else if( strcmp( tmp_in,"PD_fitness" ) == 0 )
//
//          Factor Defining The Degree Of Acceptance Of The Objective Pd
//
        in >> adum.F1_form_factor;
    else
    {
        cerr << "Error: (Targeting Input) unacceptable input option : " << tmp_in
        << "\n" << flush;
    };    // Block: 17
    in >> tmp_in;
    };    // Block: 18
//
//          Need To Initialize storage
//
    cout << "----- Targeting :: Input Complete: Storage_Setup next\n" << flush;
    adum.Storage_Setup();
    cout << "----- Targeting :: Storage_Setup Complete\n" << flush;
    return in;
    };    // Block: 19
//          ++++++

```

ostream& operator<<(ostream& out, Targeting& A)

```
{
//
//          provide coding to represent the ouput desired for this object
//
out << "\n\n" << A.obj_name << "\n";

out << "Objective Pd " << A.pd_objective << "\n";
out << "Targets " << A.max_targets << "\n";
out << "Weapons " << A.total_WH << "\n";
for( int k=1; k<= A.total_WH; k++ )
    out << A.sys[k] << " " << "\n";
out << "Convergence " << A.iterations << "\n";
return out;
};    // Block: 20
//      ++++++
```

float Targeting::Dist_Gamma(float alp, float bet, float x)

```
{
//
//          This Routine returns values associated with a gamma distribution
//
float rslt ;
float F5, Fnorm, mode_val ;
float gam_alf= 1.0;
//
//          Evaluate for the probabiity given a timing objective.
//
for( int alf=alp-1; alf>0; alf-- )
    gam_alf *= alf;
F5= pow( bet, -alp ) * pow( x, (alp-1)) * exp( -x/bet ) / gam_alf ;
//
//          Need To Normalize The Timing Fitness To Mitigate Gamma
//          Distribution Effects.
//
mode_val= bet*(alp-1) ;
Fnorm= pow( bet, -alp ) * pow( mode_val, (alp-1)) *
    exp( -mode_val/bet ) / gam_alf ;
rslt = F5/Fnorm;
return (rslt);
};    // Block: 21
//      ++++++
```

float Targeting::Dist_Sigmoid(float alp, float bet, float gam,

float x)

```
{
//
//      This Routine returns values associated with a sigmoid distribution.
//      We will assume that a decreasing sigmoid ( IC type response )
//      will utilize a standard form. An increasing sigmoid begins with a
//      value of zero progresses through the inflection point to a value
//      of one. ( questionable assumption, a PI function might be better ).
//
//      float rslt ;
//
//      Is It An Increasing Or Decreasing Sigmoid ?
//
//      beta < 0 => Decreasing !!
//      beta > 0 => Increasing !!
//
//      rslt = 1.0 / ( 1.0 + exp( -(x- $\alpha$ )/ $\beta$  ) ) ;
//
//      Provide for a cutoff
//
//      if( x >=  $\gamma$  )
//          rslt=0.005 ;
//      return (rslt);
}; // Block: 22
//      ++++++
```

void Targeting::Dump_suitability()

```
{
//
//      This Routine Dumps Information Associated With The Allocation
//      Of Weapons To Targets..
//
//
//      G_debug_out << "\n Suitability Pre-Calculations \n" << " ----- total warheads : "
//      << total_WH << "\n";
//      for( int k=0; k<max_targets; k++ )
//      {
//          G_debug_out << startX.targs[k].vntk << " " << startX.targs[k].catcode << " : ";
//          for( int sys_index=0; sys_index<=total_WH; sys_index++ )
//              G_debug_out << " ( "<<opt_index[k][sys_index]<< " "
//              <<suitability[k][sys_index]<< " ) ";
//          G_debug_out << "\n";
//      }
//      G_debug_out << "\nGoal Acceptance ( Unconstrained stockpiles ) : "
```



```

<<Mission_acceptance<<" ( Acceptance level = "<<acceptance<<" )\n";
G_debug_out <<"Goal Fitness ( Unconstrained stockpiles ) : "
<<Mission_fitness<<"\n";
return ;
}; // Block: 23
//      ++++++

void Targeting::Dump_Targeting( ostream& out )

{
//
//      This Routine Dumps Information Associated With The Targeting
//      Object Setup.
//
out << "---- :: Targeting Dump (bgn) :: ---- " << obj_name << "\n";
if( DUMP)
{
out << "---- :: Targeting Dump (---) :: ---- "
<< "Storage Parameters : (max_targets, xxx_WH(3), total_inventory) \n" ;
out << "---- :: Targeting Dump (---) :: ---- " << max_targets << " "
<< base_line_WH << "\t" << new_WH << "\t" << total_WH << "\t"
<< total_inventory << "\n";

out << "---- :: Targeting Dump (---) :: ---- Control Parameters : "
<< "(DEBUG, APPEND_rslts, Restart_iter, iterations, RN_seed) \n"
<< "---- :: Targeting Dump (---) :: ---- " << DEBUG << "\t"
<< APPEND_rslts << "\t" << Restart_iter << "\t" << iterations << "\t"
<< RN_seed << "\n";

out << "---- :: Targeting Dump (---) :: ---- Convergence Parameters : "
<< "(acceptance, Mission_acceptance, Mission_fitness,pd_objective)\n" ;
out << "---- :: Targeting Dump (---) :: ---- "
<< acceptance<< "\t" << Mission_acceptance << "\t" << Mission_fitness
<< "\t" << pd_objective << "\n" ;
out << "---- :: Targeting Dump (---) :: ---- Convergence Parameters : "
<< "(F1_form_factor, beta[3]) \n" ;
out << "---- :: Targeting Dump (---) :: ---- " << F1_form_factor << "\t"
<< beta[0] << "\t" << beta[1] << "\t" << beta[2] << "\t" << "\n" ;

out << "---- :: Targeting Dump (---) :: ---- Weapon Parameters : \n";
for( int k=0; k<= total_WH; k++ )
out << sys[k] ;
}

out << "---- :: Targeting Dump (---) :: ---- Mission Parameters : \n";
startX.Dump_Mission_( out );

```

```

out << "---- :: Targeting Dump (end) :: ---- \n";
return ;
}; // Block: 24
//      ++++++

void Targeting::IC_Alloc( )

{
//
//      This Routine Provides Setups For The Targeting Problem.
//
int k_option;
//
//      The First Step Of The Process Is To Use The Fuzzy "Tactical" Model
//      To Establish Target Priorities And Strategic Time Line Criteria.
//
//      Model tracking information is stored in file "Z_tactics"
//      The objective is to loop through each target in the scenario and
//      estimate the "weight" parameter and the "time urgency" parameter
//      associated with each target. Run_model() must accept the
//      parameters and transfer the results for restorage in the appropriate
//      target information.
//
if( FUZZY_TACTIC )
{
float in[10];
float out[10];
for( int fz=0; fz<max_targets; fz++ )
{
in[0]= startX.targs[fz].lethality;
in[1]= startX.targs[fz].mission;
in[2]= startX.targs[fz].function;

tactic.Assess_Issues( in,out );

startX.targs[fz].Wt = out[0];
startX.targs[fz].Tm_obj = out[1];
startX.targs[fz].HOB_obj = out[2];
}
}

//
//      Setup An Experimental Distribution To Identify What The "Weight"
//      Cutoff Will Be. (Target Rich Environments)
//
//

```

```

//          Identify Total Warhead Inventories
//
int total_stockpile=0 ;
for( int k2=1; k2<=total_WH; k2++ )
    total_stockpile += sys[k2].inventory ;
total_inventory= total_stockpile;
//
//          Note: space allocation for tmp_wt should be the max of either
//          max_targets or total_inventory not the sum. The sum is easier to
//          execute and will not degrade performance.
//
float *tmp_wt;
tmp_wt= new float[ max_targets+total_inventory ];
for( int fz=0; fz<max_targets; fz++ )
    tmp_wt[fz]= startX.targs[fz].Wt ;
random.Sort( tmp_wt, max_targets );
float ratio, wt_cutoff ;
ratio = float( total_inventory )/float( max_targets ) ;
if( ratio >=1.0 )
    ratio= 1.0 ;
wt_cutoff= tmp_wt[ total_inventory ] ;
//
//          Create An Experimental Distribution For Defining Initial Targetings
//
int *tmp_hist;
tmp_hist= new int[ total_inventory ];
int indx=0;
for( int m=1; m<=total_WH; m++ )
    for( int n=0; n<sys[ m ].inventory; n++ )
    {
        tmp_hist[indx]=m;
        indx ++;
    }
random.Setup_exp_dist( tmp_hist, total_inventory );
//
//          Set The Initial Targetings
//
for( int k=0; k<max_targets; k++ )
{
    if( startX.targs[k].Wt > wt_cutoff )
    {
        k_option= int( random.exp_dist( ) ) ;
        allo_vec[k] = k_option;
    }
    else
        allo_vec[k] = no_option;
}

```

```

    }
//
//      Next Calculat The Suitabilities Of Potential Soutions
//
    Suitability_Setup();
//
//      Cleanup storage
//
    delete tmp_hist;
    return ;
};    // Block: 25
//      ++++++

float Targeting::Fitness( int targ_indx, int sys_indx, float *cep,
float yld, int fz_opts )

{
//
//      This routine provides the fitness based on goal or mission targets
//      damage expectancy. Too little or too much is not a good solution.
//
    float Pk, F0,F1,F2,F3,F4, Fsum ;
    int indx=0;
    float max_pk=0.0;
    float min_pk=99999.0;
    int indx_cep=0;
    float max_cep=0.0;
    for( int k0=0; k0< fz_opts; k0++ )
        if( cep[k0] >= max_cep )
        {
            max_cep= cep[k0];
            indx_cep=k0;
        }
//
//      Assess System response time Suitability
//
    F1= beta[1]*Obj_Time( targ_indx, sys_indx );
//
//      Assess Yield Suitability
//
    F2= beta[2]*Obj_Yield( yld );
//
//      Assess Probability Of Kill Suitability
//
    for( int k=0; k< fz_opts; k++ )

```

```

{
F0 = beta[0]*Obj_Pk( targ_indx, cep[k], yld );
F3 = beta[3]*Obj_HOB( targ_indx, sys_indx, k );
F4 = sys[sys_indx].relia[ k ] ;
//
//      Note: Discovered that some fitnesses are true trades, Pk vs yield
//      while others are critical. Trades allow summation while the critical
//      conditions should be a product type total fitness.
//
Fsum= F1*F4*(F0+F2+F3) ;

if( Fsum <min_pk )
    min_pk= Fsum;
if( Fsum > max_pk )
{
    max_pk=Fsum;
    indx= k;
}
if( ( max_pk-min_pk) <= 0.0001 )
    indx= indx_cep;
};
opt_index[ targ_indx ][sys_indx] = indx;
return( max_pk );
}; // Block: 26
//      ++++++

```

float Targeting::Next_Iteration()

```

{
//
//      This Routine Performs Series Of Iterations In An Effort To Relax
//      The Problem To A Near Optimal Solution.
//
int *draw_from;
draw_from= new int[ max_targets ];
int indx= 0;
int m ;
for( int df=0; df<max_targets; df++ )
{
    m= allo_vec[df];
    if( suitability[df][m] <= acceptance )
    {
        draw_from[ indx ] = df;
        indx++;
    }
}
}

```

```

//
//          Perform Statistical Modifications To The Allocation Vector
//
int loc1, loc2, loc3, tmp, k_option ;
float sOr, sNr, sO, sN, pr2 ;
for( int k=0; k< indx; k++ )
{
    loc1=draw_from[ k ];
    pr2 = random.fdraw();
    loc2 = int( pr2*indx );
    loc3= draw_from[ loc2 ];
    k_option= 0;
}
//
//          Check To See If Exchange or Replacement Is A Gain
//
//          NOTE: Using average suitabilities insures that an exchange is
//          better overall.
//
sO= ( suitability[ loc1 ][ allo_vec[ loc1 ] ] +
    suitability[ loc3 ][ allo_vec[ loc3 ] ] )/2.0;
if( random.e_siz > 1 )
{
    k_option= int( random.exp_dist( ) );
    sN= ( suitability[ loc1 ][ k_option ] +
        suitability[ loc3 ][ allo_vec[ loc3 ] ] )/2.0;
}
else
{
    k_option= 0;
    sN= 0.0;
}
sNr= ( suitability[ loc1 ][ allo_vec[ loc3 ] ] +
    suitability[ loc3 ][ allo_vec[ loc1 ] ] )/2.0;
//
//          Is There An Improvement ?? Then Which Is The Best Exchange
//          Or Replacement
//
if( sO < sNr || sO < sN )
{
    if( sNr >= sN )
    {
        tmp= allo_vec[ loc1 ];
        allo_vec[ loc1 ]= allo_vec[ loc3 ];
        allo_vec[ loc3 ]= tmp ;
    }
    else

```

```

        {
            tmp= allo_vec[loc1];
            allo_vec[loc1] = k_option;
            random.exp_dist_Rep( tmp );
            k_option=0;
        }
    }
    if( k_option != 0 )
        random.exp_dist_Rep( k_option );
    }
    float mission_fit =0;
    indx= 0;
    for( df=0; df<max_targets; df++ )
    {
        m= allo_vec[df];
        if( suitability[df][m] >= acceptance )
            indx++;
    }
    mission_fit = float( indx )/ float( max_targets ) ;
//
//          Cleanup Storage
//
    delete draw_from;
    return mission_fit ;
};    // Block: 27
//          ++++++

float Targeting::Obj_HOB( int targ_indx, int sys_indx, int
opt_indx )

{
//
//          Simple Evaluation Of System Performance For A Weapon Concept
//          Against a Target (targ_indx): HOB constraints are evaluated.
//
    float FzHt, TgHt, F1 ;
    float F2_form_factor ;
    FzHt=sys[sys_indx].hobs[ opt_indx ] ;
    TgHt= startX.targs[ targ_indx ].HOB_obj ;
    F2_form_factor= 0.5*FzHt ;
    if( FzHt > 99998.0 && FzHt <100000.0 )
        F1 = 1.0;
    else
        F1 = exp(-(FzHt -TgHt)*(FzHt -TgHt)/F2_form_factor ) ;
    return( F1 );
};    // Block: 28

```

```
//      ++++++
float Targeting::Obj_Pk( int targ_indx, float cep, float yld )

{
//
//      Simple Evaluation Of System Performance For A Weapon Concept
//      Against a Target (targ_indx ): Pssk constraints evaluated.
//
float Pk, F1 ;
Pk = Pssk( targ_indx, cep, yld );
F1 = exp(-(Pk-pd_objective)*(Pk-pd_objective)/F1_form_factor );
return( F1 );
}; // Block: 29
//      ++++++
```

```
float Targeting::Obj_Time( int targ_indx, int sys_indx )

{
//
//      This routine captures fitness for a time urgency metric; ie.
//      reconstitution targets, time urgent targets, and those of uniform
//      importance as functions of time.
//
//      We are assuming that the system response can be captured
//      by a representative "gamma" distribution. This distribution can
//      and has been used in queing problems and will be suitability for
//      this application. Gamma distributions require two parameters,
//       $\alpha$  and  $\beta$  They should be reasonably easy to define since:
//
//      Mean =  $\alpha \beta$ 
//      Variance =  $\alpha \beta^2$ 
//      Mode =  $\beta (\alpha - 1)$  if  $\alpha \geq 1$  zero (0) otherwise
//
//      Knowing any two of the three characteristics can define the
//      distribution form. The function assesses the probability of
//      launching a system based on the targeting requirements
//      (time-urgency).
//      NOTE: correlation assumes  $\alpha$  is an integer.
```


//

//

//

// The routine has been expanded, 5/11/98, to consider sigmoid
// functions as well as the gamma function. This routine will call
// either the Dist_Gamma or the Dist_Sigmoid function.
//

```
float F5 ;  
float alp,bet,gam, response_tm ;  
alp = sys[ sys_indx ].alpha ;  
bet = sys[ sys_indx ].beta ;  
gam = sys[ sys_indx ].gamma ;  
response_tm= startX.targs[ targ_indx ].Tm_obj ;  
//  
// Evaluate for the probability given a timing objective.  
//  
if( gam < 0.0 )  
    F5 = Dist_Gamma( alp, bet, response_tm ) ;  
else
```

```

    F5 = Dist_Sigmoid( alp, bet, gam, response_tm ) ;
    return( F5 );
}; // Block: 30
//      ++++++

```

float Targeting::Obj_Yield(float yld)

```

{
//
//      This routine provides a rule to minimize yield applied to a target.
//      Note: This Routine Is Valid If The Yield Options
//            Are Not Less Than 100
//
    float min_yield=25.0 ;
    float F2;
    F2 = pow( (min_yield/yld),0.3333) ;
    if ( F2 > 1.0 )
        F2 = 1.0;

    return( F2 );
}; // Block: 31
//      ++++++

```

void Targeting::Perf_Results()

```

{
    ofstream Monitor_out(Results_file,ios::app);

    int sys_index, indx1, avn, ak, wr_out ;
    char att;
    float Pk, yld, tmp_fl ;
    float mission_PD=0.0 ;
    float cep, hob ;
    Monitor_out.precision(6);
//
//      Create A Database Of Mission Performance And Targeting
//
    Monitor_out << "\nPerformance results: \n" << " :: vntk :: ";
    if( DEBUG )
        Monitor_out << " catcode :: r95 :: T depth :: Leth :: Mis :: Func :: ";
    Monitor_out << " Wt :: Timing :: HOB Pref :: wr :: index :: yield :: cep :: ";
    Monitor_out << " hob :: Pk ::\n";
    for( int k=0; k<max_targets; k++ )
    {
        sys_index = allo_vec[k];
        avn= startX.targs[ k ].vn;
    }
}

```

```

att= startX.targs[ k ].tc;
ak= startX.targs[ k ].k;
if( sys_index > 0 )
{
    indx1 = opt_index[k][sys_index] ;
    cep= sys[ sys_index ].ceps[ indx1 ];
    hob= sys[ sys_index ].hobs[ indx1 ];
    yld= sys[ sys_index ].yield;
    Pk =Pssk( k, cep, yld );
    wr_out= startX.wr0;
}
else
{
    Pk=0.0;
    cep=1e10;
    wr_out=0.0;
}
mission_PD += Pk ;
if( att == 'G' )
    Monitor_out << startX.targs[ k ].gvn1 << att << startX.targs[ k ].dsig1
        << startX.targs[ k ].tg1 << "\t";
else
    Monitor_out << avn << att << ak << "\t ";

    if( DEBUG )
    {
        Monitor_out << startX.targs[ k ].catcode << " " << startX.targs[ k ].r95
            << "\t" << startX.targs[ k ].targ_depth << "\t";
        Monitor_out << startX.targs[ k ].lethality << " "
            << startX.targs[ k ].mission << " " << startX.targs[ k ].function << "\t";
    }
    Monitor_out << startX.targs[ k ].Wt << "\t" << startX.targs[ k ].Tm_obj
        << "\t" << startX.targs[ k ].HOB_obj << "\t";
    Monitor_out << wr_out << "\t" << sys_index << " " << yld << " "
        << cep << " " << hob << "\t" << Pk << "\n";
}
//
//          Create A Series Of Allocation Databases
//
for( int ka=1; ka<=total_WH; ka++ )
{
    Monitor_out << "\nWeapon Targetings : ( inv no. " << ka << " )\n";
    for( int kb=0; kb<max_targets; kb++ )
    {
        sys_index = allo_vec[kb];
        indx1 = opt_index[kb][sys_index] ;
    }
}

```

```

if( sys_index == ka )
{
    cep= sys[ sys_index ].ceps[indx1];
    hob= sys[ sys_index ].hobs[indx1];
    yld= sys[ sys_index ].yield;
    avn= startX.targs[ kb ].vn;
    att= startX.targs[ kb ].tc;
    ak= startX.targs[ kb ].k;
    Pk =Pssk( kb, cep, yld );
    wr_out= startX.wr0;

    if( att == 'G' )
    {
        Monitor_out << startX.targs[ kb ].gvn1 << att << startX.targs[ kb ].dsig1
            << startX.targs[ kb ].tg1 << "\t";
        Monitor_out << startX.targs[ kb ].gvn2 << att
            << startX.targs[ kb ].dsig2 << startX.targs[ kb ].tg2 << "\t";
    }
    else
        Monitor_out << avn << att << ak << "\t";

    if( DEBUG )
    {
        Monitor_out << startX.targs[ kb ].catcode << " " << startX.targs[ kb ].r95
            << "\t" << startX.targs[ kb ].targ_depth << "\t";
        Monitor_out << startX.targs[ kb ].lethality << " "
            << startX.targs[ kb ].mission << " " << startX.targs[ kb ].function << "\t";
    }
    Monitor_out << startX.targs[ kb ].Wt << "\t" << startX.targs[ kb ].Tm_obj
        << "\t" << startX.targs[ kb ].HOB_obj << "\t";
    Monitor_out << wr_out << "\t" << sys_index << " " << yld << " "
        << cep << " " << hob << "\t" << Pk << "\n";
    }
}
}

//
//      Print Out Allocated Inventories
//
float mission_fit =0;
int m ;
int indx= 0;
for( int df=0; df<max_targets; df++ )
{
    m= allo_vec[df];
    mission_fit += suitability[df][m] ;
    if( suitability[df][m] >= acceptance )

```

```

        indx++;
    }
    mission_PD /= float( max_targets );
    Monitor_out << "\nMission PD : " << mission_PD << "\n";
    mission_fit = mission_fit / float( max_targets );
    Monitor_out << "Acceptance Criteria : " << acceptance << " Mission Fitness : "
    << mission_fit << "\n";
    Monitor_out << "\n----- Weapon Systems ----- \n";
    for( k=0; k<= total_WH; k++ )
        Monitor_out << "Weapon System : " << k << "\n" << sys[ k ] << "\n";
//
//          Cleanup Storage
//
    return ;
}; // Block: 32
//          ++++++

```

float Targeting::Pssk(int targ_indx, float cep, float yld)

```

{
    float Pk = 1.0 ;
    float a, eff_cep, R95 ;
    char atc;
//
//          Define The Weapon Radius For The Target
//
    startX.Vul_Setup( targ_indx, yld );
//
//          Use A Simple WR/Cep Correlation For Kill Probability
//
//          The effective CEP is used to capture the r95 characteristics of a
//          target. This formula was pulled from PDCALC.
//
    R95 = startX.targets[ targ_indx ].r95 ;
    eff_cep = pow( ( cep*cep + 0.231* R95*R95 ), 0.5 );
    a = (startX.wr0/eff_cep)*(startX.wr0/eff_cep);
    Pk = 1.0 - pow( 0.5,a );
    return( Pk );
}; // Block: 33
//          ++++++

```

void Targeting::Storage_Setup()

```

{
//
// Com 8          Pass Parameters Read Into The Targeting Class To Initialize

```

```

//          The Genetic algorithms.
//
    allo_vec=new int[ max_targets ];
    opt_index=new int*[ max_targets ];
    suitability = new float*[ max_targets ];
    for( int su=0; su<max_targets; su++ )
    {
        opt_index[su] = new int[ total_WH+1 ];
        suitability[su] = new float[ total_WH+1 ];
    }
//
//          Identify Total Warhead Inventories
//
    int total_stockpile=0 ;
    random.seed( RN_seed );
    for( int k=1; k<=total_WH; k++ )
        total_stockpile += sys[k].inventory ;
    total_inventory= total_stockpile;

    return ;
}; // Block: 34
//          ++++++

void Targeting::Suitability_Setup( )

{
//
//          This Routine Performs Suitability Setups For The
//          Allocation Algorithm.
//
    int sys_index, i2, indx ;
    float total_fitness, yld ;
    float *cep ;
    for( int k=0; k<max_targets; k++ )
        for( sys_index=0; sys_index<=total_WH; sys_index++ )
        {
            if( sys_index != no_option )
            {
                cep= sys[ sys_index ].ceps ;
                yld= sys[ sys_index ].yield ;
                i2= sys[ sys_index ].total_fz_opt ;
//
//          Find Optimum Performance For Each System Against Target K
//
                total_fitness = Fitness( k, sys_index, cep,yld, i2 );
            }
        }
}

```

```

        else
        {
            total_fitness=0.0;
            opt_index[ k ][sys_index] = 0;
        }
        suitability[k][sys_index]=total_fitness ;
    }
//
//          Examine Allocation If Unconstrained
//
    int tmp_indx;
    indx= 0;
    Mission_fitness=0.0;
    for( k=0; k< max_targets; k++ )
    {
        tmp_indx=0;
        for( int jj=1; jj<= total_WH; jj++ )
        {
            if( suitability[k][ jj ] > suitability[k][ tmp_indx ] )
                tmp_indx=jj ;
        }
        if( suitability[k][tmp_indx] >= acceptance )
            indx++;
        Mission_fitness +=suitability[k][tmp_indx] / float( max_targets ) ;
    }
//
//          Optimum Fitness Acheivable With Unlimited Inventories
//
    Mission_acceptance = float( indx )/ float( max_targets ) ;

//
//          Cleanup Storage
//
    return ;
}; // Block: 35
//          ++++++

```

float Targeting::Targeting_Opt()

```

{
    float mission_fitness;
    float mission_fit ;
    float fit;
    float debug_fitness;

```

```

    IC_Alloc( );

```

```

if( DEBUG ) Dump_suitability( );

float *fit_TmSr;
fit_TmSr = new float[ iterations ];
float run_ave=Mission_fitness ;
int term_crit=FALSE ;

for( int iter=Restart_iter; iter<iterations; iter++ )
{
//
//          Determin the total Targeting of weapons for a chromosome
//
for( int k=0; k<= total_WH; k++ )
{
    sys[ k ].allocated =0;
    for( int kop=0; kop < sys[ k ].total_fz_opt ; kop++ )
        sys[ k ].opt_inv[ kop ] =0 ;
}
//
// Com 9      Note: allo_vec Provides The Mapping Between Targets And WH's
//
int jj, indx0 ;
for( k=0; k< max_targets; k++ )
{
    jj = allo_vec[ k ] ;
    sys[ jj ].allocated +=1;
    indx0 = opt_index[k][jj] ;
    if( jj != 0 ) sys[ jj ].opt_inv[ indx0 ] += 1 ;
}
//
// Com 10     Create The Next Iteration
//
mission_fitness= Next_Iteration( );
mission_fit =0;
for( int df=0; df<max_targets; df++ )
    mission_fit += suitability[df][ allo_vec[df] ] ;
mission_fit /= float( max_targets ) ;
//
//          Evaluate convergence criteria (deviation from running average)
//
float conv_fac=10000.0 ;
fit_TmSr[ iter ] = mission_fit ;
if( abs( run_ave-mission_fit )*conv_fac <= 1.0 )
    term_crit= TRUE ;

if( run_ave > 0.0000001 )

```



```

    run_ave=(run_ave + mission_fit)/2.0 ;
else
    run_ave = mission_fit ;

//
//          Setup For A Debug Calculation And Output
//
if( DEBUG )
{
    G_debug_out << "Iter : "<<iter<< " Mission Acceptance = "
    << mission_fitness << " Mission Fitness = "<< mission_fit
    <<" \nTargeting Vector \n" ;
    int indx1,indx2 ;
    for( int kb=0; kb<max_targets; kb++ )
    {
        indx2= allo_vec[kb] ;
        indx1= opt_index[kb][ indx2 ] ;
        G_debug_out << "(" ;
        G_debug_out << "<< indx2 << " " ;
        G_debug_out <<indx1<< " "<< suitability[kb][indx2] << " ) " ;
    }
    G_debug_out << "\n";
    G_debug_out << "\nWeapon System Inventories \n";
    for( int kd=0; kd<= total_WH; kd++ )
    G_debug_out << kd << "\t " << sys[ kd ] << "\n";
}

//
//          Provide tracking information for normal and post analysis
//
//          Tracking output controlled by WS_Opt object
if( G_debug_cntl )
{
    for( int kd=1; kd<= total_WH; kd++ )
    {
        cout << "\nWarHead " << kd << " " << mission_fit << "\t " ;
        for( int kd2=0; kd2< sys[ kd ].total_fz_opt; kd2++ )
            cout <<" "<< sys[ kd ].ceps[kd2] <<" :: "<< sys[ kd ].opt_inv[kd2]<< " ";
        cout << "\n" << flush ;
    }
    G_debug_cntl=FALSE ;
}
if( term_crit || iter>=iterations-1)
{
    G_debug_out << "\n++++++ ++++++ Target Opt Iter : "<<iter ;
    G_debug_out << " (Population fitness : " << mission_fit <<" )\t";
    for( int kd=0; kd<= total_WH; kd++ )

```

```

    {
        G_debug_out << "\n" << kd << "\t " ;
        for( int kd2=0; kd2< sys[ kd ].total_fz_opt; kd2++ )
            G_debug_out << " " << sys[ kd ].ceps[kd2] << " :: " << sys[ kd ].opt_inv[kd2]
                << " ";
        G_debug_out << "\n";
    }
    break;
}
}

//
//          Output The Results Of The Targeting
//
if( APPEND_rslts == TRUE )
    Perf_Results( );
//
//          Cleanup Storage
//
random.CleanUp_dist( );
return( mission_fit );
}; // Block: 36

```

```
//      ++++++ Genetic Alorgythm Optimization (GA_Alloc ) ++++++
```

```
//      ++++++
// Com 11 The GA Allocation Object Provides Optimizations Based On
//      Selections From A Set Of Predefined Capabilities; Eg. Allocation
//      Of Weapons To Targets, Vehicles To Routes Etc.
//
//      Operators include: Mutation, X-over, (Splice, inversion ???)
//      ++++++
//      ++++++
```

```
class GA_Alloc
```

```
{
public:
//
//      Define addiitonal model parameters:
//      selections - maximum options to choose to allocat to a gene
//      max_gene - maximum genes on a chromosome. ( eg. max targets )
//      ga_pop - Number of chromosomes in the optimization population.
//
    int exists;
    char* obj_name;
    int selections, max_gene, ga_pop;
    int **ga_chromo;
    int **ga_chromo_O;
    float *repro_pr;

    randint random;
    int opt_soln;
    float mutation_pr, xchange_pr;

    GA_Alloc( );           // constructor
//
//      overload the input and output operators
//
    friend istream& operator>>( istream& in, GA_Alloc& );
    friend ostream& operator<<( ostream& out, GA_Alloc& );
//
//      Define addiitonal member functions:
//
    float GA_Alloc_Fitness( float* );
    int GA_Xchange( int, int*, int );
    int GA_Mutate( int );
    int GA_Reproduce( int );
    void Init_GA_Alloc();           // define storage chromo
```

```
float Next_Gen_GA_Alloc( float**, float );
void Setup_GA_Alloc( long ); // Initialize an evolutionary strategy chromo

}; // Block: 37
```

GA_Alloc::GA_Alloc()

```
{
    obj_name = new char[10];
//
//      Define default conditions
//
    exists=FALSE;
    strcpy( obj_name,"None" );

    ga_pop = 1;
    max_gene = 1;
    selections = 1;
    mutation_pr=0.01;
}; // Block: 38
//      ++++++
```

istream& operator>>(istream& in, GA_Alloc& adum)

```
{
    char tmp_in[15];
    adum.exists=TRUE;
    in >> adum.obj_name;
    in >> tmp_in;
    while( strcmp( tmp_in,"eoi" ) !=0 )
    {
        cout << "----- GA_Alloc Keyword : " << tmp_in << "\n" << flush;
        if( strcmp( tmp_in,"Selections" ) == 0 )
        {
            in >> adum.selections;
        }
        else if( strcmp( tmp_in,"GA_setups" ) == 0 )
        {
            in >> adum.ga_pop;
            in >> adum.max_gene;
            adum.Init_GA_Alloc( );
        }
        else if( strcmp( tmp_in,"Probability" ) == 0 )
        {
            in >> tmp_in;
        }
    }
}
```

```

        if( strcmp( tmp_in,"Mutation" ) == 0 )
            in >> adum.mutation_pr;
        else if( strcmp( tmp_in,"X-over" ) == 0 )
            in >> adum.xchange_pr;
        else
            cerr << "Error: (GA_Alloc:Probability Input) unacceptable input option
: "
                << tmp_in << "\n" << flush;
    }
    else
        cerr << "Error: (GA_Alloc Input) unacceptable input option : "
            << tmp_in << "\n" << flush;
    in >> tmp_in;
}; // Block: 39
return in;
}; // Block: 40
//      ++++++

```

ostream& operator<<(ostream& out, GA_Alloc& A)

```

{
//
//      provide coding to represent the ouput-desired for this object
//
    out << "\n\n" << A.obj_name;
    out << "\nSelections " << A.selections;
    out << "\nGA_setups " << A.ga_pop << " " << A.max_gene;
    out << "\nProbability Mutation " << A.mutation_pr;
    out << "\nProbability X-over " << A.xchange_pr;
    return out;
}; // Block: 41
//      ++++++

```

float GA_Alloc::GA_Alloc_Fitness(float *data)

```

{
//
// Com 12 Need To Perform Whatever Normalization Is Required For Next Step
//
    float average=0.0;
    float variance=0.0;
    float min_val, max_val;
    float a,b, delta,f_prime;
    float scale_factor=2.0;
//
// Com 13 Capture Statistics Of The Fitness Values

```

```

//
min_val=data[0];
max_val=data[0];
for( int i=0; i<ga_pop; i++ )
{
    average += data[i];
    if( data[i] <= min_val )
        min_val = data[i];
    if( data[i] >= max_val )
        max_val = data[i];
}
average = average/ga_pop;
for( i=0; i<ga_pop; i++ )
    variance += (data[i]-average)*(data[i]-average) ;
variance = variance / (ga_pop-1) ;
//
// Com 14 Perform Fitness Scaling And Shifts As Defined In
// DE Goldbergs, Genetic Algorithms pgs 76-79.
//
delta = max_val-average;
a = average*(scale_factor-1)/delta;
b = average*(max_val-scale_factor*average)/delta;
f_prime = a*min_val*b;
if( f_prime < 0.0 )
{
    delta=average-min_val;
    a=average/delta;
    b=-average*min_val/delta;
}
//
// Com 15 Scale Results Of Fitness Evaluations
//
float sum_fit=0;
for( int j=0; j<ga_pop; j++ )
{
    data[ j ] = a*data[ j ] + b;
    sum_fit += data[j];
}
repro_pr[0] = 0.0;
for( j=0; j<ga_pop; j++ )
{
    data[ j ] = data[ j ] / sum_fit ;
    repro_pr[j+1] = repro_pr[j]+data[j];
}
//
// Com 16 Find The Optimal Solution

```

```

//
max_val=data[0];
opt_soln=0;
for( i=0; i<ga_pop; i++ )
{
    if( data[i] >= max_val )
    {
        max_val=data[i];
        opt_soln=i;
    }
};
return( max_val );
};    // Block: 42
//      ++++++

```

int GA_Alloc::GA_Xchange(int jj, int *suit_set, int indx)

```

{
    float pr_bs;
    float pr1,pr2 ;
    int tmp_ga, loc1, loc2, loc3 ;

    pr_bs=xchange_pr ;
    for( int k=0; k< indx; k++ )
    {
        loc1=suit_set[ k ];
        pr2 = random.fdraw();
        loc2 = int( pr2*indx );
        loc3= suit_set[ loc2 ];
        tmp_ga= ga_chromo[ jj ][ loc1 ];
        ga_chromo[ jj ][ loc1 ]= ga_chromo[ jj ][ loc3 ];
        ga_chromo[ jj ][ loc3 ]= tmp_ga ;
    }
    return( loc3 );
};    // Block: 43
//      ++++++

```

int GA_Alloc::GA_Mutate(int jj)

```

{
    int mut_loc=0;
    int inc, cycle;
    float pr1, pr3;

    for( int IG=0; IG<max_gene; IG++ )
    {

```

```

pr1 = random.fdraw();
if( pr1 < mutation_pr )
{
    pr3 = random.fdraw();
//
//      Note: we are performing a cyclic permutation here to insure
//      selection of a viable solution. This reflects mutation in an
//      integer space, as opposed to binary space.
//
    inc= int( pr3*selections ) + ga_chromo[ jj ][ IG ];
    cycle= inc - ( inc/selections ) * selections;
    ga_chromo[ jj ][ IG ]= cycle;
    mut_loc++;
}
}
return( mut_loc );
};    // Block: 44
//      ++++++

```

int GA_Alloc::GA_Reproduce(int jj)

```

{
    float pr_val;
    int nu_chain=0;

    pr_val= random.fdraw( );
    for( int k=0; k<ga_pop; k++ )
        if( pr_val >= repro_pr[k] && pr_val < repro_pr[k+1] )
        {
            nu_chain=k;
            for( int j=1; j<max_gene; j++ )
                ga_chromo[ jj ][ j ] = ga_chromo_O[ k ][ j ];
        }
    return( nu_chain );
};    // Block: 45
//      ++++++

```

void GA_Alloc::Init_GA_Alloc()

```

{
//
// Com 17 Allocate Space For The Ga Variables.
//
    ga_chromo = new int*[ga_pop+1];
    ga_chromo_O = new int*[ga_pop+1];
    repro_pr = new float[ ga_pop+1 ];

```



```

for( int i=0; i<=ga_pop; i++ )
{
    ga_chromo_O[ i ] = new int[max_gene];
    ga_chromo[ i ] = new int[max_gene];
}
return;
};    // Block: 46
//      ++++++

float GA_Alloc::Next_Gen_GA_Alloc( float **suit, float
acceptance )

{
    float pr_val=0.0;
//
// Com 18 Randomly Select A Member Of The Old Pop. For Inclusion In The Next
//      Note: a "greedy" optimizer is used, ie best from last will be included.
//
    for( int i=0; i<ga_pop; i++ )
        for( int j=0; j<max_gene; j++ )
            ga_chromo_O[ i ][ j ] = ga_chromo[ i ][ j ];
//
// Com 19 Begin The Selection Process
//
    for( int j=0; j<max_gene; j++ )
        ga_chromo[ 0 ][ j ] = ga_chromo_O[ opt_soln ][ j ];
//
//      Create A Database Of Mutation, Xchange And Reproduction Info
//
//      ofstream GA_OUT("test.dat",ios::app);
//      GA_OUT <<"\n";
//
    int r1,r2,m1,m2,x0,x1 ;
    int kk=0;
    int *draw_from;
    draw_from= new int[ max_gene ];

    while( kk<ga_pop )
    {
//
// Com 20 Create A Vector Of Poor Targeting Suitabilities
//
        int indx= 0;
        for( int df=0; df<max_gene; df++ )
            if( suit[kk][df] <= acceptance )
            {

```

```

        draw_from[ indx ] = df;
        indx++;
    }

    r1=GA_Reproduce( kk );
//
// Com 21 Perform Cross-over operations
//
    x0=GA_Xchange( kk, draw_from, indx );
//
// Com 22 Perform mutation operations
//
    m1=GA_Mutate( kk );
    kk+=1;
//
//      GA_OUT << r1<<" "<<r2<<" "<<x0<<" "<<m1<<" "<<m2<<"\n";
//
    };

    return( pr_val );
}; // Block: 47
//      ++++++

```

void GA_Alloc::Setup_GA_Alloc(long seed_val)

```

{
    float pr_between;
//
// Com 23 Define A Random Starting Chromosome Population.
//      Note: Defining Random Starting Values Between 1 And Selections.
//
    int k_option;
    random.seed( seed_val );
    for( int i=0; i<ga_pop; i++ )
    {
        for( int k=0; k<max_gene; k++ )
        {
            pr_between= random.fdraw();
            k_option= int( pr_between*selections );
            ga_chromo[i][k] = k_option;
        }
    }
    return;
}; // Block: 48

```

```
//      +++++ Evolutionary Strategy Opt (ES_Chromosome ) +++++
```

```
//      ++++++
//
//      Define an evolutionary strategy object
//
//      Operators include: Mutation
//
//      ++++++
```

class ES_Chromosome

```
{
public:
//
//      Define Addiitonal Model Parameters:
//
    int exists;
    char* obj_name;
    int max_factors, es_pop;
    float** mean_limits;           // limits of the mean factor "i" i=1,max_factors
    float** strat_limits;         // limits of the strategy "i" i=1,max_factors
    float **es_mean;
    float **es_strat;
    float **es_mean_O;
    float **es_strat_O;
    float *repro_pr;

    randint random;
    int opt_soln;
    float mutation_pr, Xover_pr;
//
//      Define addiitonal member functions: _Fitness
//
    ES_Chromosome();
    void Setup_ES_Chromo(float**, float** );
    float ES_Chromo_Fitness( float *data );
    float Next_Gen_ES_Chromo( );
    float Mutate( );
    float Xover( );
//
//      Overload The Input And Output Operators
//
    friend istream& operator>>( istream& in, ES_Chromosome& );
    friend ostream& operator<<( ostream& out, ES_Chromosome& );
}; // Block: 49
```

```
//      ++++++
//      NOTE: The integers in the constructor must be adjusted to reflect
//      the number of variables and arguments in the model
```

ES_Chromosome::ES_Chromosome()

```
{
    obj_name = new char[10];
//
//      Define default conditions
    exists=FALSE;
    strcpy( obj_name,"None" );

    max_factors = 1;
    es_pop = 1;
    mutation_pr=0.01;
    mean_limits = new float*[2];
    strat_limits = new float*[2];

};    // Block: 50
```

```
//      ++++++
```

istream& operator>>(istream& in, ES_Chromosome& adum)

```
{
    char tmp_in[15];
    adum.exists=TRUE;
    in >> adum.obj_name;
    in >> tmp_in;
    while( strcmp( tmp_in,"eoi" ) !=0 )
    {
        cout << "----- ES_Chromosome Keyword : " << tmp_in << "\n" << flush;
        if( strcmp( tmp_in,"Factors" ) == 0 )
        {
            in >> adum.max_factors;
            adum.mean_limits[0] = new float[adum.max_factors];
            adum.mean_limits[1] = new float[adum.max_factors];
            adum.strat_limits[0] = new float[adum.max_factors];
            adum.strat_limits[1] = new float[adum.max_factors];
        }
        else if( strcmp( tmp_in,"Population" ) == 0 )
            in >> adum.es_pop;
        else if( strcmp( tmp_in,"Means" ) == 0 )
        {
```

```

        for( int i=0; i<adum.max_factors; i ++ )
            in >> adum.mean_limits[0][i] >> adum.mean_limits[1][i];
    }
    else if( strcmp( tmp_in,"Strategies" ) == 0 )
    {
        for( int i=0; i<adum.max_factors; i ++ )
            in >> adum.strat_limits[0][i] >> adum.strat_limits[1][i];
    }
    else if( strcmp( tmp_in,"Probability" ) == 0 )
    {
        in >> tmp_in;
        if( strcmp( tmp_in,"Mutation" ) == 0 )
            in >> adum.mutation_pr;
        else
            cerr << "Error: (ES_chromo:Prob Input) unacceptable input option : "
                 << tmp_in << "\n" << flush;
    }
    else
        cerr << "Error: (ES_chromosome Input) unacceptable input option : "
             << tmp_in << "\n" << flush;
    in >> tmp_in;
};    // Block: 51
return in;
};    // Block: 52

//      ++++++

ostream& operator<<( ostream& out, ES_Chromosome& A )

{
//
//      provide coding to represent the ouput desired for this object
//
    out << "\n\t---- :: ES_Chromosome Dump (bgn) :: ----" << A.obj_name;
    out << "\n\t---- :: ES_Chromosome Dump (---) :: ---- Factors "
        << A.max_factors << "\n";
    out << "\n\t---- :: ES_Chromosome Dump (---) :: ---- Population "
        << A.es_pop << "\n";
    out << "\n\t---- :: ES_Chromosome Dump (---) :: ---- Means " << "\n";
    for( int i=0; i<A.max_factors; i ++ )
        out << "\n\t---- :: ES_Chromosome Dump (---) :: ---- " << A.mean_limits[ 0 ][i]
            << "\t" << A.mean_limits[ 1 ][i] << "\n";
    out << "\n\t---- :: ES_Chromosome Dump (---) :: ---- Strategies " << "\n";
    for( i=0; i<A.max_factors; i ++ )
        out << "\n\t---- :: ES_Chromosome Dump (---) :: ---- " << A.strat_limits[0][i]
            << "\t" << A.strat_limits[1][i] << "\n";
}

```

```

out << "\t---- :: ES_Chromosome Dump (---) :: ---- Probability Mutation "
<< A.mutation_pr << "\n";

for( int kk=0; kk< A.es_pop; kk++ )
{
out << "\t---- :: ES_Chromosome Dump (---) :: ---- Population " << kk << "\n";
for( int k2=0; k2<A.max_factors; k2++ )
out << "\t---- :: ES_Chromosome Dump (---) :: ---- " << A.es_mean[kk][k2]
<< "\t" << A.es_strat[kk][k2] << "\n";
}

out << "\t---- :: ES_Chromosome Dump (end) :: ---- \n" << flush;
return out;
}; // Block: 53

//      ++++++

void ES_Chromosome::Setup_ES_Chromo(float** ml, float** sl )
{
float pr_between;
//
//      Seup An Equivalence Between The Opt Routine And The Es Arrays.
//
mean_limits = ml;
mean_limits[0]= ml[0];
mean_limits[1]= ml[1];
strat_limits = sl;
strat_limits[0] = sl[0];
strat_limits[1] = sl[1];

es_mean = new float*[es_pop];
es_strat = new float*[es_pop];
es_mean_O = new float*[es_pop];
es_strat_O = new float*[es_pop];
repro_pr = new float[es_pop+1];
//
//      define a random starting chromosome population.
//      Note: defining random starting values between upper
//      and lower limits(constraints) for both nominal and
//      strategies of search.
//
for( int i=0; i<es_pop; i++ )
{
es_mean_O[ i ] = new float[max_factors];
es_strat_O[ i ] = new float[max_factors];

```

```

    es_mean[ i ] = new float[max_factors];
    es_strat[ i ] = new float[max_factors];
    for( int k=0; k<max_factors; k++ )
    {
        pr_between= random.fdraw();
        pr_between= mean_limits[0][k] + pr_between*( mean_limits[1][k]
            -mean_limits[0][k]);
        es_mean[i][k] = pr_between;

        pr_between= random.fdraw();
        pr_between= strat_limits[0][k] + pr_between*( strat_limits[1][k]
            -strat_limits[0][k]);
        es_strat[i][k] = pr_between;
    }
}
return;
}; // Block: 54

//      ++++++

float ES_Chromosome::ES_Chromo_Fitness( float *data )

{
//
//      need to perform whatever normalization is required for next step
//
    float average=0.0;
    float variance=0.0;
    float min_val, max_val;
    float a,b, delta,f_prime;
    float scale_factor=2.0;
//
// Com 24 Capture Statistics Of The Fitness Values
//
    min_val=data[0];
    max_val=data[0];
    for( int i=0; i<es_pop; i++ )
    {
        average += data[i];
        if( data[i] <= min_val )
            min_val = data[i];
        if( data[i] >= max_val )
            max_val = data[i];
    }
    average = average/es_pop;
    for( i=0; i<es_pop; i++ )

```

```

        variance += (data[i]-average)*(data[i]-average) ;
        variance = variance / (es_pop-1) ;
//
// Com 25 Perform Fitness Scaling And Shifts As Defined In
//      DE Goldbergs, Genetic Algorithms pgs 76-79.
//
        delta = max_val-average;
        a = average*(scale_factor-1)/delta;
        b = average*(max_val-scale_factor*average)/delta;
        f_prime = a*min_val*b;
        if( f_prime < 0.0 )
        {
            delta=average-min_val;
            a=average/delta;
            b=-average*min_val/delta;
        }
//
//      Scale Results Of Fitness Evaluations
//
        float sum_fit=0;
        for( int j=0; j<es_pop; j++ )
        {
            data[ j ] = a*data[ j ] + b;
            sum_fit += data[j];
        }
        repro_pr[0] = 0.0;
        for( j=0; j<es_pop; j++ )
        {
            data[ j ] = data[ j ] / sum_fit ;
            repro_pr[j+1] = repro_pr[j]+data[j];
        }
//
//      Find The Optimal Solution
//
        max_val=data[0];
        opt_soln=0;
        for( i=0; i<es_pop; i++ )
        {
            if( data[i] >= max_val )
            {
                max_val=data[i];
                opt_soln=i;
            }
        };
        return( max_val );
}; // Block: 55

```



```

//      ++++++
float ES_Chromosome::Next_Gen_ES_Chromo( )

{
    float pr_val=0.0;
//
//      Randomly Select A Member Of The Old Pop. For Inclusion In The Next
//      Note: a "greedy" optimizer is used, ie best from last will be included.
//
    for( int i=0; i<es_pop; i++ )
        for( int j=0; j<max_factors; j++ )
            {
                es_mean_O[ i ][ j ] = es_mean[ i ][ j ];
                es_strat_O[ i ][ j ] = es_strat[ i ][ j ];
            }
//
//      Begin The Selection Process
//
    for( int j=0; j<max_factors; j++ )
        {
            es_mean[ 0 ][ j ] = es_mean_O[ opt_soln ][ j ];
            es_strat[ 0 ][ j ] = es_strat_O[ opt_soln ][ j ];
        }
    for( int jj=1; jj<es_pop; jj++ )
        {
            pr_val= random.fdraw( );
            for( int k=0; k<es_pop; k++ )
                if( pr_val >= repro_pr[k] && pr_val < repro_pr[k+1] )
                    for( j=1; j<max_factors; j++ )
                        {
                            es_mean[ jj ][ j ] = es_mean_O[ k ][ j ];
                            es_strat[ jj ][ j ] = es_strat_O[ k ][ j ];
                        }
        }
//
//      Next Perform The Mutation Operations Required Of An Es Optimizer
//
    Mutate( );
//
//      Next Perform The Cross-over Operations Required Of An Es Optimizer
//
    Xover( );

```

```

    return( pr_val );
};    // Block: 56
//      ++++++

float ES_Chromosome::Mutate( )

{
    float pr_val=0.0;
    float Ai, alpha=1.3;
    int mut_loc, int_loc;
//
//      Note alpha set to 1.3 is recommended by Rechenberg. is used in
//      setting thestrategy parameter in ES mutation.
//
// Com 26 First Perform A Mutation On The Mean Values In The Chromosome
//
    for( int i=1; i<es_pop; i++ )
    {
        pr_val= random.fdraw( );
        if( pr_val <= mutation_pr )
        {
            int_loc= int( random.fdraw( ) *max_factors );
            if( int_loc<0 )
                int_loc=0;
            if( int_loc>=max_factors )
                int_loc=max_factors;

            es_mean[ i ][ int_loc ] = es_mean[ i ][ int_loc ]+random.gauss()*
                                   es_strat[ i ][ int_loc ];
            if( es_mean[ i ][ int_loc ] < mean_limits[0][int_loc] )
                es_mean[ i ][ int_loc ] = mean_limits[0][int_loc];
            if( es_mean[ i ][ int_loc ] > mean_limits[1][int_loc] )
                es_mean[ i ][ int_loc ] = mean_limits[1][int_loc];
        }
    }
//
// Com 27 Next Perfrom Mutation On Strategy Parameter; Use Same Pr Of
//      Mutation In Parallel With Mean Mutation
//
    for( i=1; i<es_pop; i++ )
    {
        pr_val= random.fdraw( );
        if( pr_val <= mutation_pr )
        {
            int_loc= int( random.fdraw( ) *max_factors );
            if( int_loc<0 )

```

```

        int_loc=0;
        if( int_loc>=max_factors )
            int_loc=max_factors;

        pr_val= random.fdraw( );
        if( pr_val < 0.5 )
            Ai = alpha;
        else
            Ai = 1.0/alpha;
        es_strat[ i ][ int_loc ] = es_strat[ i ][ int_loc ]*Ai;
    }
}

return( pr_val );
};    // Block: 57
//      ++++++

```

float ES_Chromosome::Xover()

```

{
    float pr_val=0.0 ;
    int pr_pop1, pr_pop2 ;
    float tmp ;
//
//      Can only cross over similar locations between populations. The
//      probability is that each location represents a different independant
//      variable thus can not be dislocated.
//
    for( int i=0; i<max_factors; i++ )
    {
        for( int k=1; k<es_pop; k++ )
        {
            pr_val= random.fdraw( );
            if( pr_val <= Xover_pr/2.0 )
            {
                pr_pop1 = 1+ int( random.fdraw( ) *(es_pop-1) );
                if( pr_pop1<1 )
                    pr_pop1=1;
                if( pr_pop1>=es_pop )
                    pr_pop1=es_pop;

                pr_pop2 = 1+ int( random.fdraw( ) *(es_pop-1) );
                if( pr_pop2<1 )
                    pr_pop2=1;
                if( pr_pop2>=es_pop )
                    pr_pop2=es_pop;
            }
        }
    }
}

```

```

    tmp = es_mean[ pr_pop1 ][ i ];
    es_mean[ pr_pop1 ][ i ] = es_mean[ pr_pop2 ][ i ];
    es_mean[ pr_pop2 ][ i ] = tmp ;
  }
}
for( k=1; k<es_pop; k++ )
{
  pr_val= random.fdraw( );
  if( pr_val <= Xover_pr/2.0 )
  {
    pr_pop1 = int( random.fdraw( ) *es_pop );
    if( pr_pop1<0 )
      pr_pop1=0;
    if( pr_pop1>=es_pop )
      pr_pop1=es_pop;

    pr_pop2 = int( random.fdraw( ) *es_pop );
    if( pr_pop2<0 )
      pr_pop2=0;
    if( pr_pop2>=es_pop )
      pr_pop2=es_pop;

    tmp = es_strat[ pr_pop1 ][ i ];
    es_strat[ pr_pop1 ][ i ] = es_strat[ pr_pop2 ][ i ];
    es_strat[ pr_pop2 ][ i ] = tmp ;
  }
}
}

return( pr_val );
};    // Block: 58

```

```
//
//
// Com 28 Controlling Object For The Allocation Function Of Strategic Weapon
// Systems Against Scenario Specific targetss
//
// Note: must be able to handle multiple scenarios
//
//
```

class Allocation

```
{
public:
//
// Define addiitonal model parameters:
//
int exists;
char* obj_name;

long RN_seed;
int DEBUG, fit_func;
float fuzzy_level;

int max_targets ;
int base_line_WH, new_WH, total_WH ;
int total_inventory, no_option;
int Restart_iter, ga_iter, ga_pop;
float pd_objective, F1_form_factor, beta[5];
float mutation_pr, xchange_pr;

WEAPON_ *sys;
Mission_ startX;
GA_Alloc soln;
int **opt_index;

Allocation(); // Constructor
//
// overload the input and output operators
//
friend istream& operator>>( istream& in, Allocation& );
friend ostream& operator<<( ostream& out, Allocation& );
//
// Define addiitonal member functions:
//
float Allocation_Opt( ); // Opt. The Allocation Of Weapons To Targets
float Allo_Fitness( int*, int*, float* );// Controller For Total Fitness Assessment
```

```

float Fitness_PD(int, float*, float, int );// Goal Pd Fitness Function
float Fitness_Stk_Lmt( int );           // Stockpile Limit Fitness Function
float Fitness_Time( int );              // Time Urgency Fitness Function
float Fitness_Wt( int );                // Target Weighting Fitness Function
float Fitness_Yield( float );           // Minimum Yield Fitness Function
void Genetic_Setup( );
void IC_Alloc( );
void Perf_Results( int );
float Pssk( int, float, float );        // Determine the probability of "single shot" kill
float Obj_Pk( int, float, float );
//
//           Define a dump and restart capability
//
void Gen_Dump( ostream& );
void Gen_Restart( istream& );
}; // Block: 59

```

Allocation::Allocation()

```

{
    obj_name = new char[10];
//
//           Define default conditions
//
    exists=FALSE;
    strcpy( obj_name,"None" );
    RN_seed=1111111111;
    DEBUG=FALSE;
    fit_func=1;
    fuzzy_level=1.25;

    Restart_iter=0;
    ga_iter=75;
    ga_pop=75;

    max_targets=1;
    base_line_WH=1;
    new_WH=0 ;
    total_WH= base_line_WH+new_WH ;
    no_option=0;

    pd_objective=0.75;
    F1_form_factor=0.025;
    mutation_pr=0.05;
    xchange_pr=0.45;
//

```

```

// Com 29    Beta Is The Decision Model Weighting Coeficients.
//
    for( int i=0; i<5; i++ )
        beta[i]=1.0;
};    // Block: 60
//      ++++++

istream& operator>>( istream& in, Allocation& adum )

{
    char tmp_in[15];
    char tmp_file[15];
    adum.exists=TRUE;
    int tmp;
    in >> adum.obj_name;
    in >> tmp_in;
    while( strcmp( tmp_in,"eoi" ) !=0 )
    {
        if( strcmp( tmp_in,"Restart" ) == 0 )
        {
            in >> tmp_file ;
            ifstream Restart_file( tmp_file ) ;
            Restart_file >> adum.Restart_iter >> adum.RN_seed;
            adum.Gen_Restart( Restart_file ) ;
        }
        else if( strcmp( tmp_in,"Targets" ) == 0 )
        {
            in >> adum.startX;
            adum.max_targets = adum.startX.max_targets ;
        }
        else if( strcmp( tmp_in,"Weapons" ) == 0 )
        {
//
//          First load baselines and other allocation parameters
//          Next load the CEP values and associated stockpile assignments
//
            in >> adum.base_line_WH >> adum.new_WH;
            adum.total_WH = adum.base_line_WH + adum.new_WH;
            adum.sys= new WEAPON_[ adum.total_WH+1 ];
            for( int k=1; k<= adum.total_WH; k++ )
                in >> adum.sys[ k ] ;
        }
        else if( strcmp( tmp_in,"Population" ) == 0 )
            in >> adum.ga_pop;
        else if( strcmp( tmp_in,"Convergence" ) == 0 )
            in >> adum.ga_iter;
    }
}

```

```

else if( strcmp( tmp_in,"Mutation" ) == 0 )
    in >> adum.mutation_pr;
else if( strcmp( tmp_in,"Xchange" ) == 0 )
    in >> adum.xchange_pr;
else if( strcmp( tmp_in,"Debug" ) == 0 )
    {
        in >> adum.fit_func;
        adum.DEBUG = TRUE ;
    }
else if( strcmp( tmp_in,"ObjectivePD" ) == 0 )
//
//          Objective Mission Performance
//
    in >> adum.pd_objective;
else if( strcmp( tmp_in,"PD_fitness" ) == 0 )
//
//          Factor Defining The Degree Of Acceptance Of The Objective Pd
//
    in >> adum.F1_form_factor;
else
{
    cerr << "Error: (Allocation Input) unacceptable input option : " << tmp_in
    << "\n" << flush;
}; // Block: 61
in >> tmp_in;
}; // Block: 62
//
//          Need To Initialize The Genetic Chromosome
//
adum.Genetic_Setup();
return in;
}; // Block: 63
//          ++++++

```

ostream& operator<<(ostream& out, Allocation& A)

```

{
//
//          provide coding to represent the ouput desired for this object
//
    out << "\n\n" << A.obj_name << "\n";

    out << "Objective Pd " << A.pd_objective << "\n";
    out << "Targets " << A.max_targets << "\n";
    out << "Weapons " << A.total_WH << "\n";
    for( int k=1; k<= A.total_WH; k++ )

```



```

    out << A.sys[k] << " " << "\n";
    out << "Population " << A.ga_pop << "\n";
    out << "Convergence " << A.ga_iter << "\n";
    out << "Mutation Pr " << A.mutation_pr << "\n";
    out << "Xover Pr " << A.xchange_pr << "\n";
    return out;
}; // Block: 64
//      ++++++

```

float Allocation::Allocation_Opt()

```

{
    float *fit;
    float *raw_fit;
    fit = new float[ ga_pop ];
    raw_fit = new float[ ga_pop ];
    float debug_fitness;
    ofstream debug_out("ZALLO_debug.dat",ios::app);

    if( DEBUG )
        debug_out << "\n" <<
            "+++++++ Allocation Fitness Debug ++++++++ \n";

    IC_Alloc( );

    float **suitability;
    suitability = new float*[ ga_pop ] ;
    for( int su=0; su<ga_pop; su++ )
        suitability[su] = new float[ max_targets ] ;

    for( int iter=Restart_iter; iter<ga_iter; iter++ )
    {
        //
        // Com 30      Setup And Calculate The Fitness For Each Member Of The
        //              Chromosome Population
        //
        for( int k=0; k<soln.ga_pop; k++ )
            fit[ k ] = Allo_Fitness( soln.ga_chromo[ k ], opt_index[k], suitability[k] );
        //
        //              Dump Info For Restart Or Debug
        //
        ofstream dump_out("ZALLO_dump.dat");
        RN_seed = soln.random.get_seed();
        dump_out << iter << " " << RN_seed << "\n";
        Gen_Dump( dump_out );
        if( DEBUG )

```

```

    {
        debug_out << "Fitness Results, all Chromosomes: ( iter "<<iter<<" )\n";
        for( int ka=0; ka<soln.ga_pop; ka++ )
            debug_out<< fit[ ka ] << " ";
    }

//
// Com 31    Transfer Fitness To GA Algorithms
//
    soln.GA_Alloc_Fitness( fit );

//
//          Setup For A Debug Calculation And Output
//
    if( DEBUG )
    {
        debug_fitness= Allo_Fitness ( soln.ga_chromo[ soln.opt_soln ],
            opt_index[soln.opt_soln], suitability[ soln.opt_soln ] );
        debug_out << "Iter : "<<iter<<" Fitness = "<<debug_fitness
        <<" ( func no. "<<fit_func<<" ) \nAllocation Vector \n" ;
        for( int kb=0; kb<max_targets; kb++ )
            debug_out << soln.ga_chromo[soln.opt_soln][kb] << " "
            << opt_index[soln.opt_soln][kb] << " ";
        debug_out << "\n";
    }

//
// Com 32    Create The Next Generation
//
    soln.Next_Gen_GA_Alloc( suitability, fuzzy_level );
}

if( DEBUG )
    debug_out << "\n" <<
    "+++++++ Allocation Fitness Debug End ++++++ \n";

//
//          Output The Results Of The Allocation
//
    int k_soln=0;
    Perf_Results( k_soln );

    return( fit[ soln.opt_soln ] );
}; // Block: 65
//          ++++++

float Allocation::Allo_Fitness( int *ga_soln, int *cep_index,
float *suit )

{
//

```

```

//          This routine provides the decision model, currently trivial, for
//          assessing the various goals and objectives of an allocation
//          algorithm.
//
int jj, sys_index, sys_inv;
float yld;
float *cep;
//
//          Determin the total allocation of weapons for a chromosome
//
for( int k=0; k<= total_WH; k++ )
    sys[ k ].allocated =0;
//
// Com 33    Note: ga_soln Provides The Mapping Between Targets And Options
//          assign Provides The Mapping Between Options And Inventories
//
for( k=0; k< max_targets; k++ )
{
    jj = ga_soln[ k ];
    sys[ jj ].allocated +=1;
}
//
//          Assess Fitness For Each Decision Variable, ie. F1-F5
//          Assume equal importance for this version: 11/20/97
//
int indx, i2 ;
float total_fitness;
for( k=0; k<max_targets; k++ )
{
    sys_index = ga_soln[k];
//
//          Note: Fitness Deined As Weighted Sum Of Obj Function;
//          Product For All Targets PI( (1+sum(fi))j )
//
total_fitness=0.0;
if( sys_index != no_option )
{
    cep= sys[ sys_index ].ceps ;
    yld= sys[ sys_index ].yield ;
    i2= sys[ sys_index ].total_fz_opt ;
    indx = Fitness_PD( k, cep,yld, i2 );
    cep_index[ k ] = indx;
    total_fitness += beta[0]*Obj_Pk( k, cep[indx],yld ) ;
    total_fitness += beta[1]*Fitness_Yield( yld );
//          total_fitness += beta[2]*Fitness_Stk_Lmt( k );
//          total_fitness += beta[3]*Fitness_Wt( k );

```

```

//          total_fitness += beta[4]*Fitness_Time( k );
    }
    suit[k]=total_fitness ;
}

//
//          Fitness For The Allocation Is The Percent Of Acceptable
//          Individual Targetings
//
total_fitness=0.0;
for( k=0; k<max_targets; k++ )
    if( suit[k] >= fuzzy_level )
        total_fitness +=1.0 ;
total_fitness /= float( max_targets );

return( total_fitness );
}; // Block: 66
//          ++++++

float Allocation::Fitness_PD( int targ_indx, float *cep, float yld,
int fz_opts )

```

```

{
//
//          This routine provides the fitness based on goal or mission targets
//          damage expectancy. Too little or too much is not a good solution.
//
float Pk, F1 ;
int indx=0;
float max_pk=0.0;
for( int k=0; k< fz_opts; k++ )
{
    Pk = Pssk( targ_indx, cep[k], yld );
    F1 = exp(-(Pk-pd_objective)*(Pk-pd_objective)/F1_form_factor );
    if( F1 > max_pk )
    {
        max_pk=F1;
        indx= k;
    }
};
return( indx );
}; // Block: 67
//          ++++++

```

float Allocation::Fitness_Stk_Lmt(int inv_num)

```

{

```

```

//
//      This routine is a fitness representations for stockpile constraints;
//      not a good idea to allocate weapons you do not have.
//
float F3=1;
int num_avail, num_alloc ;
num_avail= sys[ inv_num ].inventory;
num_alloc= sys[ inv_num ].allocated;
F3 = 1.0/( 1.0+exp(num_alloc-num_avail) );
return( F3 );
};    // Block: 68
//      ++++++

```

float Allocation::Fitness_Time(int targ_indx)

```

{
//
//      This routine captures fitness for a time urgency metric; ie.
//      reconstitution targets, time urgent targets, and those of uniform
//      importance as functions of time.
//
float F5=1.0;
return( F5 );
};    // Block: 69
//      ++++++

```

float Allocation::Fitness_Wt(int targ_indx)

```

{
//
//      This routine is the fitness correlation based on target
//      importance factors.
//
float F4=1;
F4 = startX.targs[ targ_indx ].Wt ;
return( F4 );
};    // Block: 70
//      ++++++

```

float Allocation::Fitness_Yield(float yld)

```

{
//
//      This routine provides a rule to minimize yield applied to a target.
//      Note: This Routine Is Valid If The Yield Options
//      Are Not Less Than 100

```

```

//
float min_yield=100.0 ;
float F2=1;
F2 = pow( (min_yield/yld),0.3333);
return( F2 );
};    // Block: 71
//      ++++++

void Allocation::Genetic_Setup()

{
//
// Com 34    Pass Parameters Read Into The Allocation Class To Initialize
//           The Genetic algorithms.
//
soln.selections = total_WH ;
soln.max_gene = max_targets;
soln.ga_pop = ga_pop;

opt_index=new int*[ ga_pop ];
for( int i=0; i<ga_pop; i++ )
opt_index[i]= new int[ max_targets ];

soln.mutation_pr = mutation_pr;
soln.xchange_pr = xchange_pr;
//
// Com 35    Initialize Space For Chromosomes In The Ga
//
soln.Init_GA_Alloc( );
//
//           Identify Total Warhead Inventories
//
int total_stockpile=0 ;
soln.random.seed( RN_seed );
for( int k=1; k<=total_WH; k++ )
    total_stockpile += sys[k].inventory ;
total_inventory= total_stockpile;

return ;
};    // Block: 72
//      ++++++

void Allocation::IC_Alloc( )

{
//

```

```

//          This Routine Provides GA Setups For The Allocation Problem.
//          Normal GA Setup Is Handled By The Genetic Setup
//          Routine: Init_GA_Alloc().
//
int k_option;

float pr_alloc, pr_tmp;
pr_alloc = float( total_inventory/max_targets ) ;
if( pr_alloc >=1.0 )
    pr_alloc= 1.0 ;

float *tmp_hist;
tmp_hist= new float[ total_inventory ];
for( int i=0; i<soln.ga_pop; i++ )
{
//
//          Create An Experimental Distribution For Defining Initial Allocations
//
int indx=0;
for( int m=1; m<=total_WH; m++ )
    for( int n=0; n<sys[ m ].inventory; n++ )
    {
        tmp_hist[indx]=m;
        indx ++;
    }
soln.random.Setup_exp_dist( tmp_hist, total_inventory );
//
//          Set The Initial Allocations
//
for( int k=0; k<soln.max_gene; k++ )
{
    pr_tmp= soln.random.fdraw();
    if( pr_tmp <= pr_alloc )
    {
        k_option= int( soln.random.exp_dist( ) ) ;
        soln.ga_chromo[i][k] = k_option;
    }
    else
        soln.ga_chromo[i][k] = no_option;
}

return ;
}; // Block: 73
//          ++++++

```

void Allocation::Perf_Results(int soln_index)

```
{
    ofstream Monitor_out("Zresults.dat",ios::app);
    int sys_index, indx1, avn, ak ;
    char att;
    float Pk, yld, tmp_fl ;
    float cep ;
    Monitor_out.precision(6);
    float *tmp_suit ;
    tmp_suit= new float[ max_targets ] ;
//
//          Create A Database Of Mission Performance And Allocation
//
    tmp_fl = Allo_Fitness( soln.ga_chromo[ soln_index ],
        opt_index[soln_index], tmp_suit );

    Monitor_out << "\nPerformance results: \n"
    << " :: vn :: :: t :: :: k :: :: index :: :: yield :: :: cep :: :: Pk ::\n";
    for( int k=0; k<max_targets; k++ )
    {
        sys_index = soln.ga_chromo[ soln_index ][k];
        avn= startX.targs[ k ].vn;
        att= startX.targs[ k ].tc;
        ak= startX.targs[ k ].k;
        if( sys_index > 0 )
        {
            indx1 = opt_index[ soln_index ][k];
            cep= sys[ sys_index ].ceps[ indx1 ];
            yld= sys[ sys_index ].yield;
            Pk =Pssk( k, cep, yld );
        }
        else
        {
            Pk=0.0;
            cep=1e10;
        }
        Monitor_out << avn << "\t " << att << "\t " << ak << "\t " << sys_index
        << "\t " << yld << "\t " << cep << "\t " << Pk << "\n";
    }
//
//          Create A Series Of Allocation Databases
//
    for( int ka=1; ka<=total_WH; ka++ )
    {
        Monitor_out << "\nTarget Allocations : ( inv no. " << ka << " )\n";
    }
}
```



```

for( int kb=0; kb<max_targets; kb++ )
{
    sys_index = soln.ga_chromo[ soln_index ][kb];
    indx1 = opt_index[ soln_index ][kb];
    if( sys_index == ka )
    {
        cep= sys[ sys_index ].ceps[indx1];
        yld= sys[ sys_index ].yield;
        avn= startX.targs[ kb ].vn;
        ak= startX.targs[ kb ].k;
        Monitor_out << avn << "\t " << ak << "\t " << sys_index << "\t " << yld
        << "\t " << cep << "\n";
    }
}
//
//          Print Out Allocated Inventories
//
Monitor_out << "\nWeapon System Inventories \n";
for( k=0; k<= total_WH; k++ )
    Monitor_out << k << "\t " << sys[ k ].inventory << "\t " << sys[ k ].allocated
    << "\n";
return ;
}; // Block: 74
//          ++++++

```

float Allocation::Pssk(int targ_indx, float cep, float yld)

```

{
    float Pk =1.0 ;
    float a;
    int avn,at,ak;
    char atc;
//
//          Define The Weapon Radius For The Target
//
    avn= startX.targs[ targ_indx ].vn;
    atc = startX.targs[ targ_indx ].tc;
    ak = startX.targs[ targ_indx ].k;

    startX.Vul_Setup( avn,atc,ak, yld ) ;
//
//          Use A Simple WR/Cep Correlation For Kill Probability
//
    a = (startX.wr0/cep)*(startX.wr0/cep);
    Pk = 1.0 - pow( 0.5,a );
}

```

```

    return( Pk );
};    // Block: 75
//      ++++++

float Allocation::Obj_Pk( int targ_indx, float cep, float yld )

{
    float Pk =1.0 ;
    float a, F1;
    int avn,at,ak;
    char atc;
//
//      Define The Weapon Radius For The Target
//
    avn= startX.targs[ targ_indx ].vn;
    atc = startX.targs[ targ_indx ].tc;
    ak = startX.targs[ targ_indx ].k;

    startX.Vul_Setup( avn,atc,ak, yld ) ;
//
//      Use A Simple WR/Cep Correlation For Kill Probability
//
    a = (startX.wr0/cep)*(startX.wr0/cep);
    Pk = 1.0 - pow( 0.5,a );

    F1 = exp(-(Pk-pd_objective)*(Pk-pd_objective)/F1_form_factor );
    return( F1 );
};    // Block: 76
//      ++++++

void Allocation::Gen_Dump( ostream& out )

{
//
//      Dump All Information Related To Allocation To A File For Restart
//
    int k;
    out << exists << "\n" ;
    out << DEBUG << " " << fit_func << "\n" ;
    out << max_targets << "\n" ;
    out << base_line_WH << " " << new_WH << " " << total_WH << "\n" ;
    out << ga_iter << " " << ga_pop << "\n" ;
    out << pd_objective << " " << F1_form_factor << " " << beta[0] << " "
    << beta[1] << " " << beta[2] << " " << beta[3] << " " << beta[4] << "\n" ;
    out << mutation_pr << " " << xchange_pr << "\n" ;
    for ( k=0; k<= total_WH; k++ )

```

```

    out << sys[k] << "\n" ;
    out << "\n" ;
    out << startX.max_targets << "\n" ;
    for( k=0; k<startX.max_targets; k++ )
        out << startX.targs[k];
    out << "\n" ;
    out << soln.selections << " " << soln.max_gene << " " << soln.ga_pop << " "
    << soln.mutation_pr << " " << soln.xchange_pr << " " << soln.opt_soln << "\n" ;
    for( k=0; k<soln.ga_pop; k++ )
    {
        for( int kk=0; kk<soln.max_gene;kk++ )
            out << soln.ga_chromo[k][kk] << " " << opt_index[k][kk] << " " ;
        out << "\n" ;
    }
    out << "\n" ;
    return;
}; // Block: 77
//      ++++++

```

void Allocation::Gen_Restart(istream& in)

```

{
//
//      Read All Information Related To Allocation To A File For Restart
//
    in >> exists ;
    in >> DEBUG >> fit_func ;
    in >> max_targets ;
    in >> base_line_WH >> new_WH >> total_WH ;
    sys= new WEAPON_ [ total_WH+1 ];
    for( int k=0; k<= total_WH; k++ )
        in >> sys[k] ;
    in >> ga_iter >> ga_pop ;
    in >> pd_objective >> F1_form_factor >> beta[0] >> beta[1] >> beta[2] >>
    beta[3] >> beta[4] ;
    in >> mutation_pr >> xchange_pr ;
    in >> startX.max_targets;
    startX.targs= new TARGET_ [ startX.max_targets ] ;
    for( k=0; k<startX.max_targets; k++ )
        startX.targs[k].Reset_Targ( in ) ;
    in >> soln.selections >> soln.max_gene >> soln.ga_pop >> soln.mutation_pr
    >> soln.xchange_pr >> soln.opt_soln ;
    soln.Init_GA_Alloc();
    for( k=0; k<soln.ga_pop; k++ )
        for( int kk=0; kk<soln.max_gene;kk++ )
            in >> soln.ga_chromo[k][kk] >> opt_index[k][kk] ;
}

```

```
return;  
}; // Block: 78
```

```

//      ++++++ Mission Definition ( Mission_ ) ++++++

//      ++++++
//      This class define target characteristics and vulnerabilities
//      ++++++
//      ++++++

class Mission_

{
public:
//      Define addiitonal model parameters:
//
    int exists;
    char* obj_name;
    int DUMP ;

    struct TARGET_ {
        float lat, lon, r95, elevation, targ_depth ;
        int catcode, vn, k, lethality, mission, function ;
        int gvn1, dsig1, gslop1, gvn2, dsig2, gslop2 ;
        char tc, tg1, tg2;
        char vntk[6], gvn[6];
        float Wt, Tm_obj, HOB_obj;
        TARGET_()
        { lat=0.0; lon=0.0; r95=0.001; elevation=0.0; targ_depth=0.0;
          Wt=1.0; Tm_obj=30.0; lethality=0; mission=0; function=0; }
        friend istream& operator>>(istream& in, TARGET_& t)
        {
            char tmp[6];
            in >> t.catcode >> t.lat >> t.lon >> t.r95 >> t.elevation >> t.targ_depth ;
//            in >> t.lethality >> t.mission >> t.function ;
            in >> t.vntk;
            t.tc=t.vntk[2];
            if( t.tc == 'G' )
            {
                tmp[0]=t.vntk[0];
                tmp[1]=t.vntk[1];
                tmp[2]='\0';
                t.gvn1 = atof( tmp );
                tmp[0]=t.vntk[3];
                tmp[1]='\0';
                t.dsig1 = atof( tmp );
            }
        }
    };
};

```

```

        t.tg1=t.vntk[4];
        in >>t.gvn;
        tmp[0]=t.gvn[0];
        tmp[1]=t.gvn[1];
        tmp[2]= '\0';
        t.gvn2 = atof( tmp );
        tmp[0]=t.gvn[3];
        tmp[1]='\0';
        t.dsig2 = atof( tmp );
        t.tg2=t.gvn[4];
    }
else
    {
        tmp[0]=t.vntk[0];
        tmp[1]=t.vntk[1];
        tmp[2]= '\0';
        t.vn = atof( tmp );
        tmp[0]=t.vntk[3];
        tmp[1]='\0';
        t.k = atof( tmp );
    }
return in;
}

friend ostream& operator<<(ostream& out, TARGET_& t)
{
    out << t.catcode << " " << t.lat << " " << t.lon << " " << t.r95 << " "
    << t.elevation << " " << t.targ_depth << " " << t.Wt << " " << t.Tm_obj
    << " " << t.HOB_obj << " " << t.lethality << " " << t.mission << " "
    << t.function << " ";
    if( t.tc == 'G' )
        out << t.gvn1 << t.tc << t.dsig1 << t.tg1 << " "<< t.gvn2 << t.tc
        << t.dsig2 << t.tg2 << " ";
    else
        out << t.vn << t.tc << t.k << " ";
    //      out << "\n" ;
    return out;
}
};

struct LP_ {
    float lat;
    float lon;
    LP_()
    { lat= 0.0; lon=0.0; }
    friend istream& operator>>(istream& in, LP_& t)
    {
        in >> t.lat >> t.lon;

```

```

        return in;
    }
    friend ostream& operator<<(ostream& out, LP_& t)
    {
        out << "Launch Pt( lat: "<< t.lat<< " long: "<< t.lon<< " ) " << "\n";
        return out;
    }
};

int max_targets;
TARGET_* targs;
int catcodes ;
int **trans_tbl;
int max_lps;
LP_* lps;
int pts_curve;
int DC_req;
float overpressure, dynamicpressure, g_shock, wr0;
float* rr;
float* hh;

//
//          Define addiitonal member functions:
//
Mission_();          // constructor
int Adj_vn( int, int, float );
float Brode_84( float, float );
float Brode_DP( float, float );
void CatCode_Trans( );
float CR_calc( int, int, float );
float DP_calc( int, int, float );
void Dump_Mission_( ostream& out );
float Gr_coupling( float, float );
float OP_calc( int, int, float );
void OP_contour( float );
float Relative_dist( int, int );
void Vul_Setup( int, float );
float WR_cr( float, float, float );
float WR_dp( float, float, float );
float WR_gvn( int, float, float );
float WR_op( float, float, float );

//
//          overload the input and output operators
//
friend istream& operator>>( istream& in, Mission_& );
friend ostream& operator<<( ostream& out, Mission_& );
}; // Block: 79

```

```
//      ++++++
//      NOTE: The integers in the constructor must be adjusted to reflect
//      the number of variables and arguments in the model
```

Mission_::Mission_()

```
{
    obj_name = new char[10];
//
//      Define default conditions
//
    exists=FALSE;
    strcpy( obj_name,"None" );
    DUMP=FALSE;
    max_targets=1;
    pts_curve=50;
    DC_req=FALSE;

    char tmp2[4];
}; // Block: 80
```

```
//      ++++++

istream& operator>>( istream& in, Mission_& t )
```

```
{
    char tmp_in[15];
    t.exists=TRUE;
    in >> tmp_in;
    while( strcmp( tmp_in,"eoi" ) !=0 )
    {
//
//      Output The Mission_ Keywords As They Are Loaded
//
        cout << "----- Mission_ Keyword : " << tmp_in << "\n" << flush;
        if( strcmp( tmp_in,"Targets" ) == 0 )
        {
            in >> t.max_targets;
            t.targs = new TARGET_[ t.max_targets ];
            for( int k=0; k< t.max_targets; k++)
            {
                in >> t.targs[k];
            }
        }
        else if( strcmp( tmp_in,"Launch_Pts" ) == 0 )
        {
```



```

        in >> t.max_lps;
        t.lps = new LP_[ t.max_lps ];
        for( int kk=0; kk< t.max_lps; kk++)
            in >> t.lps[kk];
    }
    else if( strcmp( tmp_in,"Dump" ) == 0 )
        t.DUMP=TRUE ;
    else if( strcmp( tmp_in,"Translation" ) == 0 )
    {
        in >> tmp_in;
        ifstream tbl_in( tmp_in );
        tbl_in >> t.catcodes ;
        t.trans_tbl = new int*[ t.catcodes ];
        for( int tb=0; tb< t.catcodes; tb++ )
        {
            t.trans_tbl[ tb] = new int[ 4 ];
            for( int tb2=0; tb2 < 4; tb2++ )
                tbl_in >> t.trans_tbl[ tb][ tb2 ];
        }
    }
    else if( strcmp( tmp_in,"Contour_pts" ) == 0 )
    {
        in >> t.pts_curve;
        t.rr = new float[ t.pts_curve ];
        t.hh = new float[ t.pts_curve ];
    }
    else
    {
        cerr << "Error: (Mission_ Input) unacceptable input option : " << tmp_in
            << "\n" << flush;
    }; // Block: 81
    in >> tmp_in;
    }; // Block: 82
//
//          Provide Mapping From Catcodes To Fuzzy Variables
//
    t.CatCode_Trans();
    cout << "----- Mission_ :: Input Complete\n" << flush;
    return in;
}; // Block: 83

//          ++++++
ostream& operator<<( ostream& out, Mission_& t )

{

```

```

//
//      provide coding to represent the ouput desired for this object
//
out << t.obj_name;
return out;
};    // Block: 84
//+++++

int Mission_::Adj_vn( int t_vn, int t_k, float yld )

{
    float avn;
    float a,b,c,d,e;
    a = t_k/10 - 1.0;
    b = pow((20.0/yld),0.3333);
    c = 0.5*pow((t_k*b/10),2) - a;
    d = sqrt( c*c - a*a );
    e = c+d;
    avn = 5.485*log(e)+t_vn;

    return( avn );
};    // Block: 85
//+++++

float Mission_::Brode_84( float xt, float yt )

{
    float a,b,c,d,e,f;
    float g,h,p,q,r;
    float x,y,z;
    float deltap,dp1,dp2,dp3,dp4,dp5;

    deltap = 0.0;
    x = xt/1000.0;
    y = yt/1000.0;
    r = sqrt( x*x + y*y );
    if( x < 0.00005 ) x = 0.00005;
    z = y/x;

    if( fabs( z ) <= 100.0 )
    {
        a = 1.22-(3.908*z*z) / ( 1.0+810.2*pow( z,5.0 ) );
        b = 2.321 + ( pow( z,18.0 ) / ( 1.0+1.1130*pow( z,18.0 ) ) ) * 6.195
          - 0.03831*pow( z,17.0 ) / ( 1.0+0.02415*pow( z,17.0 ) )
          + 0.6692 / ( 1.0 + 4164.0*pow( z,8.0 ) );
        c = 4.153 - ( 1.149*pow( z,18.0 ) ) / ( 1.0+1.641*pow( z,18.0 ) )
    }
}

```

```

        -1.1 / ( 1.0+2.771*pow( z,2.5 ) );
d = -4.166 + (25.76*pow( z,1.75 ) ) / ( 1.0+1.382*pow( z,18.0 ) )
    + 8.257*z / ( 1.0+3.219*z );
e = 1.0 - 0.004642*pow( z,18.0 ) / ( 1.0+0.003886*pow( z,18.0 ) );
f = 0.6096 + 2.879*pow( z,9.25 ) / ( 1.0 +2.359*pow( z,14.5 ) )
    - 17.15*z*z / ( 1.0+71.66*z*z*z );
g = 1.83 + 5.361*z*z / ( 1.0+0.3139*pow( z,6.0 ) );
h = -( 64.67*pow( z,5.0 ) +0.2905 ) / ( 1.0+441.5*pow( z,5.0 ) )
- 1.389*z / ( 1.0+49.03*pow( z,5.0 ) ) + ( 8.808*pow( z,1.5 ) )
    / ( 1.0+154.5*pow( z,3.5 ) ) + (0.0014*r*r / ( 1.0-0.158*r +
    0.0486*pow( r,1.5 ) + 0.00128*r*r ) ) * ( 1.0/(1.0+2.0*y));
p = 0.000629 - 2.67*y*y / ( 1.0+10000000.0*pow( y,4.3 ) );
q = 5.18 + 74000.0*pow( y,3.5 ) / ( 1.0+264000.0*pow( y,4.0 ) );

dp1 = 10.47/pow( r,a );
dp2 = b/pow( r,c );
dp3 = d*e/( 1.0+f*pow( r,g ) );
dp4 = h;
dp5 = p/pow( r,q );
deltap = dp1+dp2+dp3+dp4+dp5;
}
else
{
a = 1.22 - (3.908*z*z) / ( 1.0+810.2*pow( z,5.0 ) );
b = 2.321 + ( 6.195/1.113 ) - ( 0.03831/0.02415 );
c = 4.153 - ( 1.149/1.641 ) - 1.1/( 1.0+2.771*pow( z,2.5 ) );
d = -4.166 + 8.257*z / ( 1.0+3.219*z );
e = 1.0 - 0.004642/0.003886;
f = 0.6096 - 17.15*z*z / ( 1.0+71.66*z*z*z );
g = 1.83 + 5.361*z*z / ( 1.0+0.3139*pow( z,6.0 ) );
h = -( 64.67*pow( z,5.0 )+0.2905 ) / ( 1.0+441.5*pow( z,5.0 ) )
    - 1.389*z / ( 1.0+49.03*pow( z,5.0 ) ) + 8.808*pow( z,1.5 ) /
    ( 1.0+154.5*pow( z,3.5 ) ) + ( 0.0014*r*r / ( 1.0-0.158*r+
    0.0486*pow( r,1.5 )+0.00128*r*r ) ) * ( 1.0/( 1.0+2.0*y ) );
p = 0.000629 - 2.67*y*y / ( 1.0+10000000.0*pow( y,4.3 ) );
q = 5.18 + 74000.0*pow( y,3.5 ) / ( 1.0+264000.0*pow( y,4.0 ) );
deltap = 10.47/pow( r,a ) + b/pow( r,c ) + d*e/( 1.0+f*pow( r,g ) )
    + h + p/pow( r,q );
if( deltap < 0.0 ) deltap = 0.0;
}

return deltap;
}; // Block: 86
//+++++

```

float Mission_::Brode_DP(float xt, float yt)

```
{
//
//      Define the dynamic pressure at xt,yt
//      Note: This curve is based on a correlation from Brode & Speicher.
//      Ref. Air Blast from Nuclear Bursts-analytic approximations;
//      PSR Report 1419-1; H. Brode, S. Speicher
//
float scaled_hob, dpprs ;
float a,b,c,d, psi, xi, scaled_gr ;

psi = yt + 0.09;
xi = 0.914*pow( psi,2.5 ) + 170.0*psi / ( 1+337.0*pow( psi,0.25 ) ) ;
a = 2.28 + 12.5* yt*yt / ( 1+1.292*yt ) ;
b = 3.0 + ( 0.86+2.47*yt ) / ( 1.0+114.0*pow( yt,3.0 ) ) ;
c = (0.21+2.2*yt)/( 1.0+200.0*pow( yt,3.0 ) ) ;
d = (0.008+0.24*yt) / ( 1.0+260.0*pow( yt,3.0 ) ) ;

dpprs= a / (pow( xt,b ) + 0.025*pow( xt,4.5 ) ) + c / (xt + d*pow( xt,4.0 ) ) ;

return dpprs;
}; // Block: 87
//+++++
```

void Mission_::CatCode_Trans()

```
{
int cc,cc1,cv1,cvlp, indx ;
for( int k=0; k< max_targets; k++ )
{
cc = targs[k].catcode;
for( int tb=0; tb<catcodes-1; tb++ )
{
if( cc >= trans_tbl[ tb ][0] && cc < trans_tbl[ tb+1 ][0] )
{
cv1 = cc - trans_tbl[ tb ][0] ;
cvlp = trans_tbl[ tb+1 ][0] - cc ;
if( cv1= cvlp )
indx=tb;
else
indx=tb+1;
break;
}
}
targs[k].lethality= trans_tbl[ indx ][ 1 ];
```

```

    targs[k].mission= trans_tbl[ indx ][ 2 ];
    targs[k].function= trans_tbl[ indx ][ 3 ];
  }
  return ;
}; // Block: 88
//+++++

```

float Mission_::CR_calc(int t_vn, int t_k, float yld)

```

{
  float R=2.0 ;
  float a, vn1, p1, Rn,dltA ;
//
//          Determine Ground Shock For Cratering Targets
//
  a= float(t_k)/float(10);
  dltA=999.0;
  while( dltA>0.001 )
  {
    Rn = 1.0- a+ a*pow(R,0.5)*pow( (20.0/yld), 0.333 );
    dltA=abs( Rn-R);
    R= Rn;
  }
  vn1= 5.485 * log( R ) +float( t_vn );
  p1 = 1.1216 * pow( 1.2,vn1 );
  return( p1 );
}; // Block: 89
//+++++

```

float Mission_::DP_calc(int t_vn, int t_k, float yld)

```

{
  float R=2.0 ;
  float a, vn1, p1, Rn,dltA ;
//
//          Determine the Overpressure
//
  a= float(t_k)/float(10);
  dltA=999.0;
  while( dltA>0.001 )
  {
    Rn = 1.0- a+ a*pow(R,0.3333)*pow( (20.0/yld), 0.333 );
    dltA=abs( Rn-R);
    R= Rn;
  }
  vn1= 2.742 * log( R ) +float( t_vn );

```

```

    p1 = 0.02893 * pow( 1.44,vn1 );
    return( p1 );
};    // Block: 90
//+++++

void Mission_::Dump_Mission_( ostream& out )

{
//
//      This Routine Provides Information Related To Mission And A
//      Dump Capability To Be Used In Debug, And Potentially Restart
//      Function.
//
    out << "---- :: Mission_ Dump (bgn) :: ---- " << obj_name << "\n";
    out << "---- :: Mission_ Dump (---) :: ---- "
        << "Storage Para : (max_targets, max_lps, pts_curve)\n";
    out << "---- :: Mission_ Dump (---) :: ---- " << max_targets << "\t" << max_lps
        << "\t" << pts_curve << "\n";

    out << "---- :: Mission_ Dump (---) :: ---- "
        << "Convergence Parameters : (overpressure, dynamicpressure, g_shock, wr0
    )\n";
    out << "---- :: Mission_ Dump (---) :: ---- " << overpressure << "\t"
        << dynamicpressure << "\t" << g_shock << "\t" << wr0 << "\n";

    if( DUMP )
    {
        out << "---- :: Mission_ Dump (---) :: ---- Target Information : \n";
        for( int k=0; k<max_targets; k++ )
            out << "---- :: Mission_ Dump (---) :: ---- " << targs[k] << "\n";
    }
    out << "---- :: Mission_ Dump (end) :: ---- \n";
    return ;
};    // Block: 91
//+++++

float Mission_::Gr_coupling(float hob, float yld)

{
    float H_scale,HOB_coupling=1.0;
//
//      The Following Is The DNA Equation For A Cool Spectrum
//      ( Scaled Meters/mt )
//
    H_scale= pow( (yld/1000.0),0.333 )*hob/ft2meters;
    HOB_coupling= 1.75/pow((H_scale+1),1.3)+4.5*pow(H_scale,2.77)/

```

```

        pow((H_scale+1.8),3.6);
//
//      The New Dna Correlation (5/92) Is For A Hot Spectrum
//      ( Scaled Meters/kt )
//
H_scale= hob*ft2meters / pow( yld,0.33333 );
HOB_coupling= 0.75 * pow( H_scale,-0.106 );

return( HOB_coupling );
};    // Block: 92
//+++++

float Mission_::Relative_dist( int i, int j )

{
//
//      Calculate Relative Distance Between Dia's And Targets For Use
//      In Target-dia Assignment
//
float r_earth=3958.76;
float angle, cos_angle;
float distance;
//
//      Insert Alorythm For Estimating Distance Between Targets
//
cos_angle= sin( lps[i].lat*deg2rad )*sin( targs[j].lat*deg2rad )*
cos( (lps[i].lon-targs[j].lon)*deg2rad ) +
cos( lps[i].lat*deg2rad )*cos( targs[j].lat*deg2rad );
angle= acos( fabs(cos_angle) );
distance= r_earth*angle;
return( distance );
};    // Block: 93
//+++++

float Mission_::OP_calc( int t_vn, int t_k, float yld )

{
float R=2.0 ;
float a, vn1, p1, Rn,dltA ;
//
//      Determine the Overpressure
//
a= float(t_k)/float(10);
dltA=999.0;
while( dltA>0.001 )
{

```

```

Rn = 1.0- a+ a*pow(R,0.5)*pow( (20.0/yld), 0.333 );
dlt=abs( Rn-R);
R= Rn;
}
vn1= 5.485 * log( R ) + float(t_vn);
p1 = 1.1216 * pow( 1.2,vn1 );
return( p1 );
}; // Block: 94
//+++++

```

void Mission_::OP_contour(float yld)

```

{
//
//      Define the overpressure damage contour
//      Note: This curve corresponds to a Weapon Radius (WR)
//      if sigma(d) is zero WR = R50
//
int i,j;
float cube_root;
float rmin =4050.0;
float rmax = 50.0;
float eps = 0.001;
float pr[3],r_pr[3], x,z,angle,angle_rad,delta;

cube_root= pow( yld, 0.33333 );
delta = 90.0 / (pts_curve-1);

for( angle=0.0,j=0; j<pts_curve; angle+=delta,j++ )
{
r_pr[2] = rmin;
r_pr[1] = ( rmin + rmax ) / 2.0;
r_pr[0] = rmax;
pr[2] = Brode_84( rmin,0.0 );
pr[0] = Brode_84( 0.0,rmax );
angle_rad = angle*deg2rad;
while( (pr[0]-pr[2]) > eps* (overpressure) )
{
for( i=0; i<3; i++ )
{
x = r_pr[i] * cos( angle_rad );
z = r_pr[i] * sin( angle_rad );
pr[i] = Brode_84( x,z );
}
if( (overpressure) > pr[1] )
r_pr[2] = r_pr[1];
}
}

```



```

else
    r_pr[0] = r_pr[1];

    r_pr[1] = ( r_pr[0] + r_pr[2] ) / 2.0;
}

rr[j] = cube_root * r_pr[1] * cos( angle_rad );
hh[j] = cube_root * r_pr[1] * sin( angle_rad );
//    rr[j] = r_pr[1] * cos( angle_rad );
//    hh[j] = r_pr[1] * sin( angle_rad );
}
return;
}; // Block: 95
//+++++

void Mission_::Vul_Setup( int targ_indx, float yld )

{
//
//      This Routine Sets The Sigma Values And Defines The Weapon
//      Radius For Targets Based On The VNTK System.
//
float t_sigma[15]={ 0.1,0.3,0.4,0.5,0.2, 0.3,0.1,0.2,0.4,0.5, 0.5,0.4,0.2,0.1,0.3 };
float sigma, adj_vn ;
float HOB=0.0;
//
//      Note: Adding 1 To Indx For Each Offset From L, Q, & V For
//      P, Q & Z Type Targets. The Vector t_sigma Contains The Sigma
//      Values For Each T Of The VNTK Classification System.
//
int avn,at,ak;
char atc;
avn= targs[ targ_indx ].vn;
atc = targs[ targ_indx ].tc;
ak = targs[ targ_indx ].k;

int indx=0;
switch( atc )
{
case 'G' :
    wr0= WR_gvn( targ_indx, yld, HOB );
    break;
case 'P' :
    indx = indx+1;
case 'O' :
    indx = indx+1;

```

```

case 'N' :
    indx = indx+1;
case 'M' :
    indx = indx+1;
case 'L' :
    sigma= t_sigma[ indx ];
    overpressure= OP_calc( avn,ak,yld );
    wr0= WR_op( yld, sigma, HOB );
    break;
case 'U' :
    indx = indx+1;
case 'T' :
    indx = indx+1;
case 'S' :
    indx = indx+1;
case 'R' :
    indx = indx+1;
case 'Q' :
    indx = indx+5;
    sigma= t_sigma[ indx ];
    dynamicpressure= DP_calc( avn,ak,yld );
    wr0= WR_dp( yld, sigma, HOB );
    break;
case 'Z' :
    indx = indx+1;
case 'Y' :
    indx = indx+1;
case 'X' :
    indx = indx+1;
case 'W' :
    indx = indx+1;
case 'V' :
    indx = indx+10;
    sigma= t_sigma[ indx ];
    g_shock= CR_calc( avn,ak,yld );
    wr0= WR_cr( yld, sigma, HOB );
    break;
default:
    cout << "\n ERROR in vntk coding tc = " << atc << "\n" << flush;
}
return ;
}; // Block: 96
//+++++

```

float Mission_::WR_cr(float yld, float sig, float HOB)

```
{
//
//      Define the overpressure damage contour
//      Note: This curve corresponds to a Weapon Radius (WR)
//      if sigma(d) is zero WR = R50
//
float wr_appx, r50 ;
float rmin =4050.0;
float rmax = 50.0;
float eps  = 0.001;
float pr[3],r_pr[3], x,z ;

    r_pr[2] = rmin;
    r_pr[1] = ( rmin + rmax ) / 2.0;
    r_pr[0] = rmax;
    pr[2] = Brode_84( rmin,0.0 );
    pr[0] = Brode_84( 0.0,rmax );
    while( (pr[0]-pr[2]) > eps* (g_shock) )
    {
        for( int i=0; i<3; i++ )
        {
            x = r_pr[i] ;
            z = HOB ;
            pr[i] = Brode_84( x,z );
        }
        if( (g_shock) > pr[1] )
            r_pr[2] = r_pr[1];
        else
            r_pr[0] = r_pr[1];

        r_pr[1] = ( r_pr[0] + r_pr[2] ) / 2.0;
        r50 = pow( yld, 0.33333 ) * r_pr[1] ;
    }
//
//      Correlation between r50 and WR
//
wr_appx= r50/( 1.0 - sig*sig );
return( wr_appx );
}; // Block: 97
//+++++
```

float Mission_::WR_dp(float yld, float sig, float HOB)

```
{
```

```

//
//      Define the dynamic pressure weapon radius
//      Note: This curve is based on a correlation from Brode & Speicher.
//      Ref. Air Blast from Nuclear Bursts-analytic approximations;
//      PSR Report 1419-1; H. Brode, S. Speicher
//
float r50,wr_appx, scaled_hob ;
float a,b,c,d, psi, xmin, xi, xip1 ;
scaled_hob = (HOB/1000.0)/pow( yld,0.333 ) ;
psi = scaled_hob + 0.09;
xmin = 0.914*pow( psi,2.5 ) + 170.0*psi / ( 1+337.0*pow( psi,0.25 ) ) ;
//
//      Perform An Iteration To Define The Ground Range For Specific
//      Dynamic Pressure
//
float rmin =150.0;
float rmax = xmin;
float eps = 0.001;
float pr[3],r_pr[3], x,z ;

r_pr[2] = rmin;
r_pr[1] = ( rmin + rmax ) / 2.0;
r_pr[0] = rmax;
pr[2] = Brode_DP( rmin,scaled_hob );
pr[0] = Brode_DP( rmax,scaled_hob );
while( (pr[0]-pr[2]) > eps* (dynamicpressure) )
{
    for( int i=0; i<3; i++ )
    {
        x = r_pr[i] ;
        pr[i] = Brode_DP( x,scaled_hob ) ;
    }
    if( (dynamicpressure) > pr[1] )
        r_pr[2] = r_pr[1];
    else
        r_pr[0] = r_pr[1];

    r_pr[1] = ( r_pr[0] + r_pr[2] ) / 2.0;
    r50 = 1000.0 * r_pr[1] * pow( yld, 0.33333 ) ;
}
//
//      Correlation between r50 and WR
//
wr_appx= r50/( 1.0 - sig*sig );
return( wr_appx );
}; // Block: 98

```

```

//+++++
float Mission_::WR_gvn( int targ_indx, float yld, float HOB )
{
//
//      Define the weapon radius for GVN type targets
//
char baseC='A' ;
float slope[22]={ 0.1, 0.2, 0.25, 0.3, 0.325, 0.35, 0.375, 0.4, 0.425, 0.45, 0.475,
    0.5, 0.525, 0.550, 0.6, 0.65, 0.7, 0.75, 0.8, 0.85, 0.9, 1.0 };

int gvn1, gslop1, gvn2, gslop2, sl_indx1, sl_indx2 ;
float sig ;
gvn1= targ[ targ_indx ].gvn1;
gvn2= targ[ targ_indx ].gvn2;
sl_indx1= targ[ targ_indx ].tg1 - baseC ;
sl_indx2= targ[ targ_indx ].tg2 - baseC ;
//
//      Next Begin The Depth-to-Effect (DTE) Calculations.
//
float dte1,dte2,dte, Yover, eff_yld ;
if( HOB >=0.5 )
{
    eff_yld = yld * Gr_coupling( HOB, yld ) ;
}
else
    eff_yld= yld;

Yover = 20.0*pow( 1.07,(( gvn2-gvn1)/(slope[sl_indx2]-slope[sl_indx1])) ) ;
dte1 = (pow(eff_yld/20.0, slope[sl_indx1]) )*( 4000.0/pow(1.07,gvn1) ) ;
dte2 = (pow(eff_yld/20.0, slope[sl_indx2]) )*( 4000.0/pow(1.07,gvn2) ) ;
//
//      Now Choose Based On The Effective Yield Relative To The
//      Cross Over Yield.
//
if( eff_yld <= Yover )
{
    sig= float( targ[ targ_indx ].dsig1)/ 10.0;
    dte = dte1 ;
}
else
{
    sig= float( targ[ targ_indx ].dsig2)/ 10.0;
    dte = dte2 ;
}
}

```

```

//
//      The Next Calculation Estimates A Vertical Weapon Radius Which
//      Is Used To Approximate The Horizontal Wr.
//
float wr_appx, vert_wr;
vert_wr = dte / ( 1 - sig*sig ) ;
//
//      Also if the depth to target is greter than the depth to effect we need
//      to force conditons which will yield a Pk of 0.0
//
if( abs(targs[ targ_indx ].targ_depth) >= vert_wr )
    vert_wr = 0.0001;
//
//      (Appears That The Assumption Is That The Damage Contour Or
//      DTE Is Spherical And A root(2) Approximation Is Used.)
//
wr_appx = 1.414213562 * vert_wr ;
return( wr_appx );
};    // Block: 99
//+++++

```

float Mission_::WR_op(float yld, float sig, float HOB)

```

{
//
//      Define the overpressure damage contour
//      Note: This curve corresponds to a Weapon Radius (WR)
//      if sigma(d) is zero WR = R50
//
float wr_appx, r50 ;
float rmin =4050.0;
float rmax = 50.0;
float eps = 0.001;
float pr[3],r_pr[3], x,z ;

    r_pr[2] = rmin;
    r_pr[1] = ( rmin + rmax ) / 2.0;
    r_pr[0] = rmax;
    pr[2] = Brode_84( rmin,0.0 );
    pr[0] = Brode_84( 0.0,rmax );
    while( (pr[0]-pr[2]) > eps* (overpressure) )
    {
        for( int i=0; i<3; i++ )
        {
            x = r_pr[i] ;
            z = HOB ;

```

```

        pr[i] = Brode_84( x,z );
    }
    if( (overpressure) > pr[1] )
        r_pr[2] = r_pr[1];
    else
        r_pr[0] = r_pr[1];

    r_pr[1] = ( r_pr[0] + r_pr[2] ) / 2.0;
    r50 = pow( yld, 0.33333 ) * r_pr[1] ;
}
//
//          Correlation between r50 and WR
//
wr_appx= r50/( 1.0 - sig*sig );
return( wr_appx );
};    // Block: 100

```

```
//  +++++ Random Number operations ( randint ) +++++
```

```
#include <limits.h>
```

```
class randint
```

```
{  
// -----  
//           Object for generating random numbers  
//           gauss is used to draw from a normal distribution  
// -----  
}
```

```
long randx;
```

```
void getx()
```

```
{ randx = randx * 1103515245 + 12345; }
```

```
public:
```

```
int e_siz;
```

```
int **e_dist;
```

```
randint( long s = 1000)
```

```
{  
randx = s;  
e_siz=0;  
//           e_dist= new int*[2];  
//           e_dist[0]= new int[ s ];  
//           e_dist[1]= new int[ s ];  
}
```

```
void seed( long s )
```

```
{ randx = s; }
```

```
long get_seed( )
```

```
{ return randx ; }
```

```
int draw()
```

```
{  
getx();  
return randx & LONG_MAX;  
}
```


float fdraw()

```
{  
getx();  
return ( randx & LONG_MAX ) / (float) LONG_MAX;  
}
```

int exp_dist()

```
{  
int valu=0.0 ;  
int indx2;  
indx2= int( fdraw()*e_siz ) ;  
valu= e_dist[ 1 ][ indx2 ] ;  
//  
//          Draw Without Replacement  
//  
for( int k=indx2; k<e_siz-1; k++ )  
    e_dist[ 1 ][ k ]= e_dist[ 1 ][ k+1 ] ;  
e_siz -=1;  
  
return (valu ) ;  
}
```

void exp_dist_Rep(int tmp)

```
{  
e_dist[ 1 ][ e_siz ]= tmp;  
e_siz +=1;  
return ;  
}
```

void Setup_exp_dist(int* hist, int siz)

```
{  
float valu=0.0 ;  
e_siz= siz;  
e_dist = new int*[ 2 ];  
e_dist[ 0]= new int[ e_siz ];  
e_dist[ 1]= new int[ e_siz ];  
  
for( int k=0; k< siz; k++ )  
{  
    e_dist[ 0 ][k] = k ;  
    e_dist[ 1 ][k] = hist[ k ] ;  
}
```

```
return ;  
}
```

void CleanUp_dist()

```
{  
    e_siz=0;  
    delete e_dist;  
    return;  
};    // Block: 101
```

void Swap(float* a, float* b)

```
{  
    float temp;  
    temp= *a;  
    *a=*b;  
    *b=temp;  
    return;  
};    // Block: 102
```

void Sort(float* A, int sz)

```
{  
    int top, search;  
    for( top=0; top<sz-1; top++ )  
        for( search=top+1; search<sz; search++ )  
            if( A[search] > A[top] )  
                Swap( &A[search], &A[top] );  
    return;  
};    // Block: 103
```

float gauss()

```
{  
    static int iset=0;  
    static float gset;  
    float fac,r,v1,v2;  
    if( iset == 0 )  
    {  
        do  
        {  
            v1 = 2.0* randint::fdraw() - 1.0;  
            v2 = 2.0* randint::fdraw() - 1.0;  
            r = v1*v1 + v2*v2;  
        }  
    }
```

```
    } while( r>=1.0 );

    fac = sqrt( -2.0*log(r)/r );
    gset = v1*fac;
    iset = 1;
    return v2*fac;
  }
else
{
  iset=0;
  return gset;
}
}; // Block: 104

}; // Block: 105
```

```
//      ++++++
//      ++++++
//      Controlling Object For The WEAPON_ Function
//      ++++++
//      ++++++
```

class WEAPON_

```
{
public:
//
//      Define addiitonal model parameters:
//
int exists;
char* obj_name;

float yield, fallout_free ;
int inventory, allocated ;
int base_fz_opt, new_fz_opt, total_fz_opt ;
//
//      NOTE: System Response Characteristics Are Being Defined By
//      A Gamma Distribution.
//
float alpha, beta, gamma ;
float *ceps, *hobs, *relia ;
int *opt_inv ;
int Yield_opt, Inv_opt ;

WEAPON_();          // Constructor
//
//      overload the input and output operators
//
friend istream& operator>>( istream& in, WEAPON_& );
friend ostream& operator<<( ostream& out, WEAPON_& );
//
//      Define addiitonal member functions:
//
WEAPON_& operator=( WEAPON_& );
void WEAPON_Load( istream& );
void WEAPON_Setup( int ) ;
}; // Block: 106
```

WEAPON_::WEAPON_()

```
{
obj_name = new char[10];
```

```

exists=FALSE;
strcpy( obj_name,"None" );

Yield_opt= TRUE ;
Inv_opt= TRUE ;
yield=10.0;
inventory=0;
base_fz_opt=0;
new_fz_opt=0;
total_fz_opt=base_fz_opt+new_fz_opt;
alpha=2.0;
beta=1.0;
gamma=-1.0;
allocated=0;
}; // Block: 107

```

istream& operator>>(istream& in, WEAPON_& t)

```

{
char tmp_in[15];
char tmp_file[15];
t.exists=TRUE;
in >> t.yield >> t.inventory >> t.base_fz_opt >> t.new_fz_opt ;
t.total_fz_opt= t.base_fz_opt + t.new_fz_opt;
t.ceps= new float[ t.total_fz_opt ];
t.hobs= new float[ t.total_fz_opt ];
t.relia= new float[ t.total_fz_opt ];
t.opt_inv= new int[ t.total_fz_opt+1 ];
for( int k=0; k< t.total_fz_opt; k++ )
    in >> t.ceps[k] >> t.hobs[k] >> t.relia[k] ;
in >> t.alpha >> t.beta >> t.gamma ;
//
//          Note: Define The Fallout Free HOB To Be The Following
//
t.fallout_free = 180.0*pow( t.yield,0.3333 ) ;
return in;
}; // Block: 108
//          ++++++

```

ostream& operator<<(ostream& out, WEAPON_& t)

```

{
//
//          provide coding to represent the ouput desired for this object
//

```

```

out << "\t---- :: Weapon_ Dump (bgn) :: ---- " << t.obj_name << "\n";
out << "\t---- :: Weapon_ Dump (---) :: ---- Yield : " << t.yield
    << "\t Inventory : " << t.inventory << "\t Allocated : " << t.allocated << "\n";

out << "\t---- :: Weapon_ Dump (---) :: ---- Total Fuze Options : "
    << t.total_fz_opt << "\n";

out << "\t---- :: Weapon_ Dump (---) :: ----";
for( int k=0; k< t.total_fz_opt; k++ )
    out << "\t" << t.ceps[k] << "\t" << t.hobs[k] << "\t" << t.relia[k] << "\t" << t.opt_inv[k];

out << "\n\t---- :: Weapon_ Dump (---) :: ---- Delivery Time : \t"
    << t.alpha << "\t" << t.beta << "\t" << t.gamma << "\n";

out << "\t---- :: Weapon_ Dump (end) :: ----\n";
return out;
}; // Block: 109

```

void WEAPON_::WEAPON_Load(istream& in)

```

{
    char tmp_in[15];
    char tmp_file[15];
    exists=TRUE;
    int tmp;
    in >> tmp_in;
    while( strcmp( tmp_in,"eoi" ) !=0 )
    {
        //
        //          Output The Weapon_ Class Keywords As They Are Loaded
        //
        cout << "----- WEAPON_ Keyword : " << tmp_in << "\n" << flush;
        if( strcmp( tmp_in,"Yield" ) == 0 )
        {
            in >> yield;
            Yield_opt=FALSE;
        }
        //
        //          Note: Define The Fallout Free HOB To Be The Following
        //
        fallout_free = 180.0*pow( yield,0.3333 ) ;
    }
    else if( strcmp( tmp_in,"Inventory" ) == 0 )
    {
        in >> inventory;
        Inv_opt=FALSE;
    }
}

```

```

else if( strcmp( tmp_in,"Options" ) == 0 )
{
    in >> base_fz_opt >> new_fz_opt;
    total_fz_opt=base_fz_opt+new_fz_opt;
    ceps= new float[ total_fz_opt ];
    hobs= new float[ total_fz_opt ];
    relia= new float[ total_fz_opt ];
    opt_inv= new int[ total_fz_opt+1 ];
//
//          Default Hobs:  Imply Variable Hob Settings
//
//          Default relia : 1.0
//
    for( int k1=0; k1<total_fz_opt; k1++ )
    {
        hobs[k1] = 99999.0 ;
        relia[k1] = 1.0 ;
        opt_inv[k1] = 0 ;
    }
}
else if( strcmp( tmp_in,"CEPs" ) == 0 )
{
    for( int k=0; k< base_fz_opt; k++ )
        in >> ceps[k] ;
}
else if( strcmp( tmp_in,"HOBs" ) == 0 )
{
    for( int k=0; k< base_fz_opt; k++ )
        in >> hobs[k] ;
}
else if( strcmp( tmp_in,"SysRelia" ) == 0 )
{
    float tmp_relia ;
    in >> tmp_relia ;
    for( int k=0; k< base_fz_opt; k++ )
        relia[k] = tmp_relia ;
}
else if( strcmp( tmp_in,"Reliability" ) == 0 )
{
    for( int k=0; k< base_fz_opt; k++ )
        in >> relia[k] ;
}
else if( strcmp( tmp_in,"Response" ) == 0 )
{
    in >> alpha >> beta >> gamma ;
}

```

```

else
    cerr << "Error: (WEAPON_ Input) unacceptable input option : " << tmp_in
    << "\n" << flush;

    in >> tmp_in;
};    // Block: 110

return;
};    // Block: 111

```

void WEAPON_::WEAPON_Setup(int num_opts)

```

{
//
//          Set Space For Fuzing Options Of A Weapon System
//
    ceps= new float[ num_opts ];
    hobs= new float[ num_opts ];
    relia= new float[ num_opts ];
    opt_inv= new int[ num_opts+1 ];
    return;
};    // Block: 112

```

WEAPON_ & WEAPON_::operator=(WEAPON_ & aWeapon)

```

{
//
//          Equating One Weapon To Another " A = B "
//
    yield= aWeapon.yield ;
    inventory= aWeapon.inventory ;
    base_fz_opt= aWeapon.base_fz_opt ;
    new_fz_opt= aWeapon.new_fz_opt ;
    total_fz_opt= aWeapon.total_fz_opt ;
    for( int i=0; i<aWeapon.total_fz_opt; i++ )
    {
        ceps[i] = aWeapon.ceps[i];
        hobs[i] = aWeapon.hobs[i];
        relia[i] = aWeapon.relia[i];
        opt_inv[i] = aWeapon.opt_inv[i];
    }
    alpha = aWeapon.alpha ;
    beta = aWeapon.beta ;
    gamma = aWeapon.gamma ;

    Yield_opt= aWeapon.Yield_opt;

```



```
Inv_opt= aWeapon.Inv_opt;  
allocated= aWeapon.allocated;
```

```
return *this;  
}; // Block: 113
```

APPENDIX K Code listing for fuzzy logic algorithms.

```
//                   ++++++ Main Fuzzy Analysis ++++++
```

```
#include "header.h"
```

```
#include "FzSet.h"
```

```
#include "FzHedge.h"
```

```
#include "FzVariable.h"
```

```
#include "FzRule.h"
```

```
#include "FzIssue.h"
```

```
#include "FzDecision.h"
```

```
    int main( int argc, char *argv[])
```

```
{
```

```
    cout << "+++++++ Begin Assessment ++++++\n";
```

```
    FzDecision bs1;
```

```
    bs1.Load_Model( argv[1] );
```

```
    bs1.Dump_Model( "z_info" );
```

```
    bs1.Run_Model( "z_tracking" );
```

```
    bs1.Dump_Results( "z_result" );
```

```
    cout << "+++++++ End Assessment ++++++\n";
```

```
};    // Block: 114
```



```

//
//      The degree of truth of each preposition is ascertained and then a
//      Zadah type AND_ function is activated to determine the level to
//      associate with the then part of the proposition. As a result of the
//      2 rules the heavy surface is defined from which defuzzification will
//      produce a quantitative assessment of the targeting importance.
//
//      The Issue_ construct of these algorithms permits a structured
//      approach for answering multiple issues as well as coupling a
//      number of variables into a global model.
//
//      Input Format:
//
//      Load_Model() controls input of all elements of the fuzzy model, it
//      may transfer control to another function but the call is initiated in
//      FzDecision member functions.
//
//      Input Keywords include:
//      rule, variables, issues, defuzzify, runs, & eoi for termination.
//
//      ++++++
public:

    int De_Fuz;
    int max_runs;
    FzVariable* var;
    FzVariable* last_var;

    Rules_* first_rule;
    Rules_* last_rule;
    Rules_* current_rule;

    Issue_* first_issue;
    Issue_* last_issue;
    Issue_* current_issue;

    struct Parsed_
    {
//
//      Note: more space has been allocated(4)than allowed(3) for
//      rule atoms to mitigate a problem that surfaced during initial
//      development. The problem surface in the function which parses
//      the rules. no idea 10/30/94
//
        char ID[5];
        int n_pred;
    }

```

```

FzVariable* Vr[4];
Hedge_ * Hd[4];
FzSet* St[4];
int Opij[4];
int Aggregation_Op ;
Parsed_ *next;
Parsed_( Parsed_ *p )
    { next=p; n_pred=3; }
}; // Block: 115

```

```

Parsed_ * pi_r;
Parsed_ * pi_first;

```

```

double final[VECMAX];
double wrk[VECMAX];

```

```

//
//          Define Addiitonal Member Functions:
//

```

```

                FzDecision();
void            Assess_Issues( float*, float* );
void            DeFuzzify( int );
void            Dump_Model( char* );
void            Dump_Results( char* );
FzVariable*     Find_FzVariable( char* );
void            Issue_ID( );
void            Load_Model( char* );
void            Parse_Rule();
void            Reset_Var();
void            Rule_Aggregation( FzVariable*, int );
void            Run_Model( char* );
void            Store_Results( int, ostream& out );
void            Z_AND( );
void            Z_OR( );
int             Z_Truth( );

```

```

}; // Block: 116

```

```

//          ++++++
//          NOTE: The integers in the constructor must be adjusted to reflect
//          the number of variables and arguments in the model

```

FzDecision::FzDecision()

```

{
//          Define default conditions
De_Fuz=CENTROID;

first_rule=0;

```

```

last_rule=0;

first_issue=0;
last_issue=0;

last_var=0;
pi_first=0;

max_runs=1;
}; // Block: 117
//      ++++++
//      Function That Runs The Fuzzy Logic Algorithms for an issue
//      assuming that the algorithm is buried in a code that can set
//      case scler values on the fly.

void FzDecision::Assess_Issues(float* info_in, float* info_out)

{
  FzVariable* V[3];
  Hedge_* H[3];
  FzSet* S[3];
  double tr_V;
  double final_truth;
  int i,j,i_if,i_then, aggr_op;
  Issue_::RO_* loc_ro;
  int issue_index=0;
//
//      Set Outer Outer Loop To Handle Multiple Issues In
//      The Decision Space.
//
//      Issues correlate to finding consequent fuzzy sets, e.g. A problem is
//      searching for priorities of a process as well as trying to determine
//      overall risk, in this case we are dealing with two basic issues.
//
  for( current_issue= first_issue; current_issue != 0; current_issue= current_issue-
>next )
  {
    int j=0;
    {
      for(i=0; i<VECMAX; i++)
        final[i]=0;
//
//      Each Issue Is Comprised Of A Number Of Rules, 1-Many.
//
    for( loc_ro= current_issue->rl1; loc_ro != 0; loc_ro= loc_ro->next )
      {

```

```

pi_r= (Parsed_*)loc_ro->add_;
Reset_Var();
//
//      Identify All Variables In The Rule, All Hedges And All Fuzzy Sets
//
for( i=0; i<pi_r->n_pred; i++ )
{
    V[i]= (pi_r->Vr[i]);
    H[i]= (pi_r->Hd[i]);
    S[i]= (pi_r->St[i]);
}
//
//      Perform All Hedge Operations
//
for( i=0; i<pi_r->n_pred; i++ )
if( H[i] != 0 )
H[i]->Tran( *S[i] );
//
//      This Identifies The Proposition Predicate Portion Of The Rule And
//      The Consequent Portion Of The Rule Under Evaluation.
//
i_if=i_then=0;
for( i=0; i<pi_r->n_pred; i++ )
{
    if( pi_r->Opj[i] == IF_ ) i_if=i;
    if( pi_r->Opj[i] == THEN_ ) i_then=i;
}
//
//      Assess The Degree Of Truth For The If Predicates. Each
//      proposition predicate exhibits a degree of truth based on the value
//      of the scalers used in the current run. These values are used to
//      condition the consequent.
//
final_truth=1.0;
for( i=i_if; i<i_then; i++ )
{
    V[i]->scaler[j] = info_in[ i ];
    switch ( pi_r->Opj[i] )
    {
        case IF_ :
        case AND_ :
            tr_V= S[i]->Degree_of_Truth( V[i]->scaler[j] );
            if( tr_V < final_truth )
                final_truth= tr_V;
            break;
        case OR_ :

```



```

        tr_V= S[i]->Degree_of_Truth( V[i]->scaler[j] );
        if( tr_V > final_truth )
            final_truth= tr_V;
        break;
    default :
        break;
    }
}

//
//          Work The Then Side Of The Predicates;assumes A Zadah Type And
//
for( int k=0; k<VECMAX; k++ )
if( final_truth < S[i_then]->mu_wrk[k] )
    wrk[k]= final_truth;
else
    wrk[k]= S[i_then]->mu_wrk[k];

//
//          Final rule(s) combinatorics
//
if( i_then <= 0 )
    aggr_op=MIN_MIN;
else
    aggr_op=pi_r->Aggregation_Op ;
Rule_Aggregation( V[i_then], aggr_op );
}

//
// Com 36      De-fuzzification Must Now Be Performed
//
    DeFuzzify( j );
}

//          -----
V[i_then]->result_var = FALSE;
info_out[ issue_index ] = V[i_then]->scaler[j] ;
issue_index +=1;
//          -----
}
return;
};    // Block: 118
//          ++++++
//Fuction which de fuzzifies the solution variable after a model run

void FzDecision::DeFuzzify( int ijk )

{
    double result=0;
    FzSet* fs_tmp;

```

```

Reset_Var();

var=last_var;
while( var!=0 )
{
    if( var->result_var == TRUE )
    {
        fs_tmp = var->fs_result;
        if( var->num_scaler <= 0 )
        {
            var->num_scaler= max_runs;
            var->scaler= new double[ max_runs ];
            var->D_o_T = new double[ max_runs ];
        }
        switch ( De_Fuz )
        {
            case CENTROID :
                result= fs_tmp->Centroid();
                break;
            case AVGMAXIMUM :
                result= fs_tmp->Ave_Max();
                break;
            case MAXIMUM :
                result= fs_tmp->Max_Edge();
                break;
            default :
                result=1.0e9;
                break;
        }
        var->scaler[ijk]= result;
        var->D_o_T[ijk]= fs_tmp->Degree_of_Truth(result);
    }
    var=var->next;
}

return;
}; // Block: 119
//      ++++++

void FzDecision::Dump_Model( char* O_file )

{
    ofstream out( O_file );

    for( current_rule= first_rule; current_rule !=0; current_rule= current_rule->next )

```

```

    {
        out << "\nRule: " << (*current_rule).ID << "\n";
        current_rule->Dump_rule( out );
    }
    for( var= last_var; var !=0; var= var->next )
        var->Variable_Output( out );
    return;
}; // Block: 120
//      ++++++

```

void FzDecision::Dump_Results(char* O_file)

```

{
    ofstream out( O_file );
    int i;
    var=last_var;
    while( var!=0 )
    {
        if( var->result_ == TRUE )
        {
            out << "\n Fuzzy Variable:" << var->Fz_Var_ID << "\n";
            for( i=0; i<var->num_scaler; i++ )
                out << " " << var->Fz_Var_ID << "[ " << i+1 << " ] "
                    << var->scaler[i] << " DoT " << var->D_o_T[i] << "\n";
//
//      Print Out Again For Use With Plotting Programs
//
            out << "\n\n Fuzzy Variable:" << var->Fz_Var_ID << "(Scaler Values)\n";
            for( i=0; i<var->num_scaler; i++ )
                out << " " << var->scaler[i];
            out << "\n\n Fuzzy Variable:" << var->Fz_Var_ID << "(Deg of Truth)\n";
            for( i=0; i<var->num_scaler; i++ )
                out << " " << var->D_o_T[i];
        }
        var=var->next;
    }
//
//      for( var= last_var; var !=0; var= var->next )
//      var->Variable_Output( out );
    return;
}; // Block: 121
//      ++++++

```

FzVariable* FzDecision::Find_FzVariable(char* v_nam)

```

{

```

```

FzVariable* rtn_v;
rtn_v=0;
var=last_var;
while( var!=0 )
{
    if( strcmp( var->Fz_Var_ID, v_nam ) == 0 )
        rtn_v=var;
    var=var->next;
}

return( rtn_v );
}; // Block: 122
//      ++++++
// a then logic function

void FzDecision::Issue_ID( )

{
    Issue_::RO_* loc_ro;

    for( current_issue= first_issue; current_issue != 0; current_issue= current_issue-
>next )
    {
        loc_ro=(current_issue->r11);
        while (1)
        {
//
//      Now compare the rule with a parsed rule for a series of matches
//
            pi_r= pi_first;
            while( pi_r != 0 )
            {
                if( strcmp( loc_ro->r_ID, pi_r->ID ) == 0 )
                    loc_ro->add_ = pi_r;
                pi_r= pi_r->next;
            }
            if( loc_ro == (current_issue->rln) )
                break;
            loc_ro= loc_ro->next;
        }
    }
    return;
}; // Block: 123
//      ++++++

```

void FzDecision::Load_Model(char* l_file)

```
{
ifstream in( l_file );
char tmp_in[15];
in >> tmp_in;
while( strcmp( tmp_in,"eoi" ) !=0 )
{
cout << "----- FzDecision Keyword : " << tmp_in << "\n" << flush;
if( strcmp(tmp_in,"rule") ==0 )
{
current_rule = new Rules_( last_rule );
last_rule= current_rule;
current_rule->Load_rule( in );
in >> tmp_in;
}
else if( strcmp(tmp_in,"variables") ==0 )
{
var = new FzVariable( last_var );
last_var= var;
var->Variable_Setup( in );
in >> tmp_in;
}
else if( strcmp(tmp_in,"issues") ==0 )
{
current_issue = new Issue_( last_issue );
last_issue= current_issue;
current_issue->Load_issue( in );
in >> tmp_in;
}
else if( strcmp(tmp_in,"defuzzify") ==0 )
{
in >> tmp_in;
if( strcmp( tmp_in, "CENTROID" ) == 0 )
De_Fuz=CENTROID;
else if( strcmp( tmp_in, "AVGMAXIMUM" ) == 0 )
De_Fuz=AVGMAXIMUM;
else if( strcmp( tmp_in, "MAXIMUM" ) == 0 )
De_Fuz=MAXIMUM;
else
cout << "\nERROR: Not a developed defuzzification routine" << flush;
in >> tmp_in;
}
else if( strcmp(tmp_in,"runs") ==0 )
{
in >> max_runs;
}
```

```

        in >> tmp_in;
    }
    else
    {
        cerr << "Error: (FzDecision Input) unacceptable input option : " << tmp_in
        << "\n" << flush;
    }
};

//
//          Identify the first rule in the decision space
//
for( current_rule= last_rule; current_rule->prev != 0;
    current_rule= current_rule->prev )
{
    (current_rule->prev)->next= current_rule;
}
first_rule= current_rule;
//
// Com 37      Execute The Function Which Parses The Rules;
//              Rules And Variables Must Be Loaded
//
Parse_Rule();
//
// Com 38      Identify The First Issue In The Decision Space
//
for( current_issue= last_issue; current_issue->prev != 0; current_issue=
current_issue->prev )
{
    (current_issue->prev)->next= current_issue;
}
first_issue= current_issue;
//
// Com 39      Identify & Associate The Address Of The Parsed Rule With An Issue
//
Issue_ID();

return;
}; // Block: 124
//          ++++++

void FzDecision::Parse_Rule( )

{
    FzSet* tmp_St;
    Hedge_* tmp_Hd;
    FzVariable* tmp_Vr;

```

```

    int cnt;
//
// Com 40    Parse The Rules In Order To Prepare For Execution
//
    for( current_rule= last_rule; current_rule != 0; current_rule= current_rule->prev )
    {
        pi_r = new Parsed_( pi_first );
        pi_first= pi_r;
//
//          Zero Out Elements In The Parsed Structure
//
        for(int i=0; i<pi_r->n_pred; i++)
        {
            pi_r->Vr[i]=0;
            pi_r->Hd[i]=0;
            pi_r->St[i]=0;
            pi_r->Op[i]=999;
        }

        strcpy( pi_r->ID, current_rule->ID );
        pi_r->Aggregation_Op=current_rule->Aggregation_Op;
        current_rule->at=current_rule->at1;
        cnt=0;
        while( current_rule->at != 0 )
        {
//
//          Check On Next Atom Being A Variable
//
            tmp_Vr = Find_FzVariable( (current_rule->at)->atom );
            if( tmp_Vr != 0 )
            {
                pi_r->Vr[cnt]= tmp_Vr;
                current_rule->at= (current_rule->at)->np1;
//
//          If We Found A Variable, Check On Likelihood Of A Hedge
//
                tmp_Hd= tmp_Vr->Find_FzHedge((current_rule->at)->atom );
                if( tmp_Hd != 0 )
                {
                    pi_r->Hd[cnt]= tmp_Hd;
                    current_rule->at= (current_rule->at)->np1;
                }
//
//          if we found a variable, check on likelihood of a fuzzy set
//
                tmp_St= tmp_Vr->Find_FzSet((current_rule->at)->atom );

```

```

        if( tmp_St != 0 )
        {
            pi_r->St[cnt]= tmp_St;
            current_rule->at= (current_rule->at)->np1;
        }
        else
            pi_r->St[cnt]= tmp_Vr->fs_result;
        cnt++;
    }
    else
//
//      chect on likelihood of a operator
//
    {
        if( strcmp( (current_rule->at)->atom, "AND" ) == 0 ||
            strcmp( (current_rule->at)->atom, "and" ) == 0 )
            pi_r->Opj[cnt]= AND_;
        else if( strcmp( (current_rule->at)->atom, "OR" ) == 0 ||
            strcmp( (current_rule->at)->atom, "or" ) == 0 )
            pi_r->Opj[cnt]= OR_;
        else if( strcmp( (current_rule->at)->atom, "THEN" ) == 0 ||
            strcmp( (current_rule->at)->atom, "then" ) == 0 )
            pi_r->Opj[cnt]= THEN_;
        else if( strcmp( (current_rule->at)->atom, "IF" ) == 0 ||
            strcmp( (current_rule->at)->atom, "if" ) == 0 )
            pi_r->Opj[cnt]= IF_;
        else
            cerr << "ERROR: could not find an operator for RULE ID "
                << current_rule->ID << flush;
            current_rule->at= (current_rule->at)->np1;
        }
    }
}
return;
}; // Block: 125
//      ++++++

```

void FzDecision::Reset_Var()

```

{
    var=last_var;
    while( var !=0 )
    {
        var->Reset_FS();
        var= var->next;
    }
}

```



```

for(int i=0; i<VECMAX; i++)
{
    wrk[i]=0;
}
return;
};    // Block: 126
//      ++++++
//
//      A "Then" Logic Function. This function provides the mechanism for
//      combining sets of rule consequents. Given a set of rules in which
//      RISK is a consequent variable, the results must be convolved to
//      generate the final consequent fuzzy set.
//

```

void FzDecision::Rule_Aggregation(FzVariable* A, int aggr_op)

```

{
    FzSet* s_tmp;
    int j;
//
//      If the "solution fuzzy set" (sfs) is zero everywhere, then use a
//      MIN_MAX operator to define a default concequent fuzzy set.
//      (used in conjunction with unconditional propositions)
//
    if( Z_Truth() == FALSE )
        aggr_op = MIN_MAX;

    switch( aggr_op )
    {
        case MIN_MAX :
//
//      This is te rule combinatorics associated with precedents and
//      consequents using the Zadah intersection operaor. (Z_AND)
//      (Cox, page 226, The Fuzzy Systems Handbook )
//
            for( j=0; j<VECMAX; j++ )
                if( wrk[j] > final[j] )
                    final[j]= wrk[j];
            break;
        case MIN_MIN :
//
//      Used for unconditional consequents. E.g. then RISK is HIGH
//      This is basically the Zadah Union operator (Z_OR)
//      (Cox, page 239, The Fuzzy Systems Handbook )
//

```

```

        for( j=0; j<VECMAX; j++ )
            if( wrk[j] < final[j] )
                final[j]= wrk[j];
        break;
    case ADDITIVE :
//
//          This rule aggregation mechanism is best used for situations
//          building evidence. (See Cox, The Fuzzy Systems Handbook,
//          page 227)
//
        for( j=0; j<VECMAX; j++ )
            {
                final[j]+= wrk[j];
                if( final[j] > 1.0 )
                    final[j]=1.0;
            }
        break;
    default:
        for( j=0; j<VECMAX; j++ )
            final[j]=0.0;
    }

    s_tmp = A->fs_result;

    if( A->result_var != TRUE )
    {
        A->result_var=TRUE;
        A->result_ =TRUE;
        s_tmp->domain[0]= (A->last_fs)->domain[0];
        s_tmp->domain[1]= (A->last_fs)->domain[1];
    }
    for( j=0; j<VECMAX; j++ )
        s_tmp->mu[j]= final[j];

    return;
};    // Block: 127
//          ++++++
// Com 41    Function That Runs The Fuzzy Logic Algorithms

```

void FzDecision::Run_Model(char* O_file)

```

{
    ofstream tr_out( O_file );
    FzVariable* V[3];
    Hedge_* H[3];
    FzSet* S[3];

```

```

double tr_V;
double final_truth;
int i,j,i_if,i_then, aggr_op;
Issue_::RO_* loc_ro;
ofstream out_store( "z_store" );
//
//      Set Outer Outer Loop To Handle Multiple Issues In
//      The Decision Space.
//
//      Issues correlate to finding consequent fuzzy sets, e.g. A problem is
//      searching for priorities of a process as well as trying to determine
//      overall risk, in this case we are dealing with two basic issues.
//
for( current_issue= first_issue; current_issue != 0; current_issue= current_issue-
>next )
{
    tr_out << "\n1---> Processing " << current_issue->ID << " issues...." << flush;
//
//      Set Outer Loop To Handle Multiple Variable Value Runs;
//      basically what value do the consequent fuzzy variables become
//      given a set of MODEL variable settings.
//
for( j=0; j<max_runs; j++ )
{
    tr_out << "\n2-----> Processing case " << j << " ...." << flush;
//      -----
    for(i=0; i<VECMAX; i++)
        final[i]=0;
//
//      Each Issue Is Comprised Of A Number Of Rules, 1-Many.
//
for( loc_ro= current_issue->r1; loc_ro != 0; loc_ro= loc_ro->next )
{
    tr_out << "\n3-----> Processing rule " << loc_ro->r_ID << " ...." << flush;
    pi_r= (Parsed_*)loc_ro->add_;
    Reset_Var();
//
//      Identify All Variables In The Rule, All Hedges And All Fuzzy Sets
//
for( i=0; i<pi_r->n_pred; i++ )
{
    V[i]= (pi_r->Vr[i]);
    H[i]= (pi_r->Hd[i]);
    S[i]= (pi_r->St[i]);
}
//

```

```

//          Perform All Hedge Operations
//
for( i=0; i<pi_r->n_pred; i++ )
if( H[i] != 0 )
H[i]->Tran( *S[i] );

//
//          This Identifies The Proposition Predicate Portion Of The Rule And
//          The Consequent Portion Of The Rule Under Evaluation.
//
i_if=i_then=0;
for( i=0; i<pi_r->n_pred; i++ )
{
if( pi_r->Op[i] == IF_ ) i_if=i;
if( pi_r->Op[i] == THEN_ ) i_then=i;
}

//
//          Assess The Degree Of Truth For The If Predicates. Each
//          proposition predicate exhibits a degree of truth based on the value
//          of the scalars used in the current run. These values are used to
//          condition the consequent.
//

final_truth=1.0;
for( i=i_if; i<i_then; i++ )
{
switch ( pi_r->Op[i] )
{
case IF_ :
case AND_ :
tr_V= S[i]->Degree_of_Truth( V[i]->scaler[j] );
if( tr_V < final_truth )
final_truth= tr_V;
break;
case OR_ :
tr_V= S[i]->Degree_of_Truth( V[i]->scaler[j] );
if( tr_V > final_truth )
final_truth= tr_V;
break;
default :
break;
}
}

//
//          Work The Then Side Of The Predicates; assumes A Zadah Type And
//
for( int k=0; k<VECMAX; k++ )
if( final_truth < S[i_then]->mu_wrk[k] )

```

```

        wrk[k]= final_truth;
    else
        wrk[k]= S[i_then]->mu_wrk[k];
//
//          Final rule(s) combinatorics
//
        if( i_then <= 0 )
            aggr_op=MIN_MIN;
        else
            aggr_op=pi_r->Aggregation_Op ;
        Rule_Aggregation( V[i_then], aggr_op );
    }
//
// Com 42      De-fuzzification Must Now Be Performed
//
        DeFuzzify( j );
        Store_Results( j, out_store );
    }
//          -----
//          V[i_then]->result_var = FALSE;
//          -----
    }
    return;
}; // Block: 128
//          ++++++

```

void FzDecision::Store_Results(int case_num, ostream& out)

```

{
    FzSet* fs_tmp;
    int i;

    var=last_var;
    while( var!=0 )
    {
        if( var->result_var == TRUE )
        {
            fs_tmp = var->Find_FzSet( "RESULT_" );
            out << "\n Case Number:" << case_num+1 << "";
            out << " Fuzzy Variable:" << var->Fz_Var_ID << "\n";
            out << *fs_tmp;
        }
//
//          Print Out Again For Use With Plotting Programs
//
        out << "\n\n Fuzzy Variable:" << var->Fz_Var_ID << "(Scaler Values)\n";
    }
}

```

```

    for( i=0; i<var->num_scaler; i++ )
        out << " " << var->scaler[i];
    out << "\n\n Fuzzy Variable:" << var->Fz_Var_ID << "(Deg of Truth)\n";
    for( i=0; i<var->num_scaler; i++ )
        out << " " << var->D_o_T[i];
    var=var->next;
}

//
//          for( var= last_var; var !=0; var= var->next )
//          var->Variable_Output( out );
return;
}; // Block: 129
//          ++++++
//
//          Zadah and function
//

```

void FzDecision::Z_AND()

```

{
    for( int j=0; j<VECMAX; j++ )
        if( wrk[j] < final[j] )
            final[j]= wrk[j];
    return;
}; // Block: 130
//          ++++++
//
//          Zadah or function
//

```

void FzDecision::Z_OR()

```

{
    for( int j=0; j<VECMAX; j++ )
        if( wrk[j] > final[j] )
            final[j]= wrk[j];
    return;
}; // Block: 131
//          ++++++
//          Need a function to check the degree of truth of the "final" vector
//          for use with the setting of default consequent fuzzy sets.
//

```

int FzDecision::Z_Truth()

```

{
    int z_truth=FALSE;

```

```
for( int j=0; j<VECMAX; j++ )  
    if( final[j] > 0.001 )  
    {  
        z_truth= TRUE ;  
        return z_truth ;  
    }  
return z_truth ;  
};    // Block: 132
```

```

//      ++++++ Fuzzy Hedge Algorithms ( Hedge_ ) ++++++

class Hedge_

{
//      ++++++
//
// Com 43      Class Defines The characteristics Of Hedges.
//      Hedge_ are modifiers to members of a fuzzy variable's set. They
//      are elements of the FzVariable class. They are stored as linked
//      lists.
//
//      Hedges can be adders, factors or employ a power law.
//
//      Member functions include:
//
//      Overloaded input and output operators; operator>> & operator<<
//      and a transformation function; Tran which is used to perform the
//      fuzzy level modification.
//
//      Input format is as follows:
//
//      hedge
//      SIGNIFICANTLY power 2.0
//
//      Keyword is "hedge " (keyed on in Variable_Setup of FzVariable)
//      The semantic modifier "SIGNIFICANTLY" uses a power law with
//      the exponent set to 2.0.
//      ++++++
public:
//
//      Assume all hedges are public...
//
char *ID;
double adder, factor, expon;
Hedge_ *next;

      Hedge_( Hedge_ *p );
void      Tran( FzSet&);
friend istream&      operator>>(istream& in, Hedge_ & t);
friend ostream&      operator<<(ostream& out, Hedge_ & t);
}; // Block: 133
//      ++++++

```



```

Hedge_::Hedge_( Hedge_ *p )

{ next=p; adder=0; factor=1; expon= 1; ID= new char[15]; }

//      ++++++

void Hedge_::Tran( FzSet& bs )

{
for(int i=0; i<VECMAX; i++ )
{
bs.mu_wrk[i]= (adder+bs.mu[i])*factor;
if( bs.mu_wrk[i] > 1.0 ) bs.mu_wrk[i]=1.0;
bs.mu_wrk[i]= pow( bs.mu_wrk[i],expon );
}
}; // Block: 134

//      ++++++

istream& operator>>(istream& in, Hedge_& t)

{
char tmp[15];
in >> t.ID;
in >> tmp;
if( strcmp( tmp,"add" ) == 0 )
in >> t.adder;
else if( strcmp( tmp,"multiply" ) == 0 )
in >> t.factor;
else if( strcmp( tmp,"power" ) == 0 )
in >> t.expon;
else
cerr << "Error: (Hedge_ Input) not an input option : " << tmp << "\n" << flush;
return in;
}; // Block: 135

//      ++++++

ostream& operator<<(ostream& out, Hedge_& t)

{
out << " " << t.ID << " Add: " << t.adder<< " Prod: " << t.factor << " Exp: "
<< t.expon << "\n";
return out;
}; // Block: 136

```

```

//          ++++++ Fuzzy Issue Control ( Issue_ ) ++++++

class Issue_

{
//          ++++++
//
// Com 44    Defines Characteristics And Function Of Fuzzy Issues.
//
//          Issue_ consist of blocks of rules associated with an issue.
//          Issues are part of the "FzDecision class" variable list and are
//          elements of a linked list.
//
//          Member functions for rules define into and output of a issues.
//          i.e. Load_issue & Dump_issue
//
//          Issue format is as follows:
//
//          issues risk a1 a2 a3 a4 a5 eoi
//
//          Keyword is "issues " (keyed on in Load_Model of FzDecision)
//          The variable following the keyword is an ID term descriptive of the
//          issue at hand. The five(5) a1...a5 are the ID's associated with a
//          rule.
//
//          RO_ is the structure internal to ISSUE_ which contains the set
//          of rules comprising an issue. Detail usage is discussed in
//          FzDecision.
//          ++++++
public:
char ID[40];
Issue_ *next;
Issue_ *prev;

    struct RO_

    {

        char* r_ID;
        void* add_;
        RO_ *prev;
        RO_ *next;
        RO_( RO_ *p )
        {prev=p; next=0; r_ID= new char[5]; }
    };    // Block: 137

    RO_ *rln;

```

```

RO_ *rl1;
        Issue_( Issue_ *p );
void      Load_issue(istream& in);
void      Dump_issue(ostream& out);
}; // Block: 138
//      ++++++

Issue_::Issue_( Issue_ *p )

{ prev=p; next=0; rl1=0; rln=0; }
//      ++++++

void Issue_::Load_issue(istream& in)

{
char tmp[15];
RO_ *rl;

in >> ID;
in >> tmp;
while( strcmp( tmp,"eoi" ) != 0 )
{
    rl= new RO_( rln );
    rln= rl;
    strcpy( rl->r_ID, tmp );
    in >> tmp;
};

for( rl =rln; rl->prev !=0; rl =rl->prev )
    (rl->prev)->next =rl;
rl1= rl;
return;
}; // Block: 139
//      ++++++

void Issue_::Dump_issue(ostream& out)

{
RO_ *rl;
for( rl=rl1; rl !=0; rl=rl->next )
    out << " " << rl->r_ID << " ";
return;
}; // Block: 140

```

```

//      ++++++ Fuzzy Rule Operations ( Rules_ ) ++++++

class Rules_

{
//      ++++++
//
// Com 45      Class defines the structure of a rule.
//      Rules_ consist of "rule atoms" RA_ ie the parts of a rule
//      Rules which are part of the "FzDecision class" variable list are
//      elements of a linked list. The rule atoms makeing up a rule are
//      also structured as linked lists.
//
//      Member functions for rules define into and output of a rule sets.
//      i.e. Load_rule & Dump_rule
//
//      Rule format is as follows:
//
//      rule c1 < if SERVCIE_LIFE is LONG then MATERIALS is EXOTIC >
//
//      Keyword input is "rule " (keyed on in Load_Model of FzDecision)
//      follwed by the rule enclosed in angle brackets "< >"
//      if, is, then, and : are the logic elements of the rule
//      ++++++
public:
char ID[5];
int Aggregation_Op ;
Rules_ *next;
Rules_ *prev;

struct RA_

{
char *atom;
RA_ *nm1;
RA_ *np1;
RA_( RA_ *p )
{nm1=p; np1=0; atom= new char[15];}
friend istream& operator>>(istream& in, RA_& t)
{
in >> t.atom;
return in;
}
friend ostream& operator<<(ostream& out, RA_& t)
{

```

```

        out << " & " << t.atom << " & ";
        return out;
    }
}; // Block: 141

```

```

RA_ *at1;
RA_ *atn;
RA_ *at;
Rules_( Rules_ *p );
void Load_rule(istream& in);
void Dump_rule(ostream& out);
}; // Block: 142

```

```

//      ++++++

```

```

Rules_::Rules_( Rules_ *p )

```

```

{prev=p; next=0; at1=0; atn=0; Aggregation_Op=MIN_MAX; }

```

```

//      ++++++

```

```

void Rules_::Load_rule(istream& in)

```

```

{
    char tmp[15];
    in >> ID;
    in >> tmp;
    while( strcmp( tmp,">" ) != 0 )
    {
        if( strcmp( tmp,"Op" ) ==0 )
        {
            in >> tmp;
            if( strcmp( tmp,"MIN_MAX" ) )
                Aggregation_Op=MIN_MAX;
            else if( strcmp( tmp,"MIN_MIN" ) )
                Aggregation_Op =MIN_MIN;
            else if( strcmp( tmp,"ADDITIVE" ) )
                Aggregation_Op =ADDITIVE;
            else
                Aggregation_Op=MIN_MAX;

            in >> tmp;
        }
        else if( strcmp( tmp,"<" ) ==0 )
            in >> tmp;
        else if( strcmp( tmp,"is" ) == 0 || strcmp( tmp,"IS" ) == 0 )

```

```

        in >> tmp;
    else
    {
        at= new RA_( atn );
        atn= at;
        strcpy( at->atom, tmp );
        in >> tmp;
    }

};

//
//      Next go through and define next atom np1
//
for( at =atn; at->nm1 !=0; at =at->nm1 )
    (at->nm1)->np1 =at;
at1= at;
return;
};    // Block: 143

//      ++++++

void Rules_::Dump_rule(ostream& out)

{
    for( at=at1; at !=0; at=at->np1 )
        out << " " << at->atom << " ";
    return;
};    // Block: 144

```

```
//      ++++++ Fuzzy Sets Operations ( FzSet ) ++++++
```

```
class FzSet
```

```
{
//      ++++++
//
// Com 46      Define The Characteristics Of Fuzzy Sets
//
//      Fuzzy sets capture the levels associated with a semantic variable.
//      They are stored as linked lists. Fuzzy sets capture aspects of a
//      semantic or qualitative problem by defining levels to be associated
//      with a variable. Fuzzy sets are members of the FzVariable class.
//
//      Member functions include:
//
//      A significant number of functions such as; Setup_Beta( ),
//      Setup_Custom( ), Setup_Gauss( ), Setup_Linear( ), Setup_Pi( ),
//      Setup_Sigmoid( ), Setup_Tri( ), Setup_Uniform( ) are used to
//      define the shape of the membership function. Note: not all are
//      defined at this point in time(2/27/98).
//
//      The second block of functions define defuzzification methods;
//      Centroid(), Ave_Max() & Max_Edge()
//
//      The remaining functions address the following functions:
//
//      Alfacut_Above sets values associated with an element to zero
//      beyond a specified level.
//      Degree_of_Truth
//      Fz_Initialize simply sets values to the membership function over
//      the range specified in domain.
//      Get_Max
//      Normalize_FZS
//      Reset_Wrk resets the working vector of membership levels back to
//      the original level.
//
//      Input and output is acheived through the overloaded redirection
//      operators.
//
//      Input format is as follows:
//
//      fuzzy_set
//      HIGH domain 0 50 func_form sigmoid 5 25 45 increase
//      eoi
```

```

//
//      Keyword is "fuzzy_set" ( keyed on in Variable_Setup of FzVariable)
//      The ID for member of the set follows the keyword, "HIGH" in this
//      case. "domain" defines the range of the variable for the set element,
//      while "func_form" triggers input of the shape of the member
//      element. In this case the shape is an increasing sigmoid function
//      with low, mid, and high points defined to be 5, 25, & 45.
//      The levels associated with the shape function are stored in
//      the variable parms
//
//      ++++++
public:
    int exists;
//
//      Fuzzy Set Descriptor Block (FSDB)
//
    char* ID;                // FZS id or name
    char* C_type;            // FZS curve type
    int conv_trend;          // Converted trend for linear & sigmoid func
    int set_stat;            // FZS status, set or not set
    int fz_order;            // FZS order ?
    double domain[2];        // FZS min and max domain values
    double parms[4];
    double alfacut;          // FZS alfacut
    double mu[VECMAX];       // FZS membership array
    double mu_wrk[VECMAX];   // FZS membership working array
    FzSet* next;             // Linked list, last fuzzy set was...

//
//      overload the input and output operators
//
    friend istream& operator>>( istream& in, FzSet&);
    friend ostream& operator<<( ostream& out, FzSet&);
//
//      Define additional member functions:
//
    FzSet( FzSet* );
    void Help();
    void Alfacut_Above( double );
    double Centroid();
    double Ave_Max();
    double Max_Edge();
    double Degree_of_Truth( double );
    void Fz_Initialize( );
    double Get_Max();
    void Normalize_FZS( );

```



```

void      Reset_Wrk( );
void      Setup_Beta( );
void      Setup_Custom( );
void      Setup_Gauss( );
void      Setup_Linear( );
void      Setup_Pi( );
void      Setup_Sigmoid( );
void      Setup_Tri( );
void      Setup_Uniform( );
}; // Block: 145

```

```

//      ++++++
//      NOTE:The integers in the constructor must be adjusted to reflect
//      the number of variables and arguments in the model

```

FzSet::FzSet(FzSet *p)

```

{
exists=FALSE;
ID = new char[15];
C_typ = new char[15];
set_stat=FALSE;
next= p;
}; // Block: 146

```

```

//      ++++++
//      Note:Provide output to the terminal to assist in defining the object
//      model and the information required

```

void FzSet::Help()

```

{
return;
}; // Block: 147

```

```

//      ++++++

```

istream& operator>>(istream& in, FzSet& adum)

```

{
char tmp_in[15];
char trend[15];

adum.exists=TRUE;
in >> adum.ID;
in >> tmp_in;
while( strcmp( tmp_in,"eoi" ) !=0 )

```

```

{
if( strcmp( tmp_in,"domain" ) == 0 )
    in >> adum.domain[0] >> adum.domain[1];
else if( strcmp( tmp_in,"alfa_cut" ) == 0 )
    in >> adum.alfacut;
else if( strcmp( tmp_in,"func_form" ) == 0 )
{
    in >> adum.C_typ;
    if( strcmp( adum.C_typ,"gauss" ) == 0 )
        in >> adum.parms[0] >> adum.parms[1];
    else if( strcmp( adum.C_typ,"linear" )== 0 )
    {
        in >> adum.parms[0] >> adum.parms[1];
        in >> trend;
    }
    else if( strcmp( adum.C_typ,"sigmoid" )== 0 )
    {
        in >> adum.parms[0] >> adum.parms[1] >> adum.parms[2];
        in >> trend;
    }
    else if( strcmp( adum.C_typ,"triangular" )== 0 )
        in >> adum.parms[0] >> adum.parms[1] >> adum.parms[2];
    else if( strcmp( adum.C_typ,"uniform" )== 0 )
        in >> adum.parms[0] >> adum.parms[1];
    else
        cerr << "Error: (FzSet Input) unacceptable form option : " << tmp_in
        << "\n" << flush;
    }
else
{
    cerr << "Error: (FzSet Input) unacceptable input option : " << tmp_in
    << "\n" << flush;
}
in >> tmp_in;
}

//
//      Initialize The Fuzzy Set
//
if( strcmp( adum.C_typ,"linear" )== 0 || strcmp( adum.C_typ,"sigmoid" )== 0 )
{
    if( strcmp( trend, "increase" )== 0 )
        adum.conv_trend= INCREASE;
    else if( strcmp( trend, "decrease" )== 0 )
        adum.conv_trend= DECREASE;
    else
        cerr << "Error: (FzSet Input) not a trend option : " << trend << "\n" << flush;
}

```

```

    }
    adum.Fz_Initialize( );
    return in;
}; // Block: 148

//      ++++++
ostream& operator<<( ostream& out, FzSet& adum )

{
//
//      Provide Coding To Represent The Ouput Desired For This Object
//
    out << adum.ID << "\n";
    double domain_width;
    double member_value;
    int i;
    domain_width= adum.domain[1]-adum.domain[0];
    for( i=0; i<VECMAX; i++)
    {
        member_value = adum.domain[0] + (float)i * domain_width / VECMAX;
        out << " " << member_value << " " << adum.mu[i] << " "
        << adum.mu_wrk[i]
        << "\n";
    }
    return out;
}; // Block: 149

//      ++++++
//      Note:Modify fuzzy set with an alfa cut
//

void FzSet::Alfacut_Above( double cut_val )

{
    int i;
    for( i=0; i<VECMAX; i++ )
        if( mu[i] < cut_val ) mu[i] = 0.0;
    return;
}; // Block: 150

//      ++++++
//      Note:determine the maximum membership value
//

double FzSet::Centroid( )

{
    int i;

```

```

double domain_width;
double member_value;
double s1=0.0;
double s2=0.0;
domain_width= domain[1]-domain[0];
for( i=0; i<VECMAX; i++)
{
    member_value = domain[0] + (float)i * domain_width / VECMAX;
    s1+= mu_wrk[i];
    s2+= mu_wrk[i]*member_value;
}
if( s1 == 0 ) s1 =999999.9;
return( s2/s1 );
};    // Block: 151
//      ++++++
//      Note:determine the maximum membership value
//

```

double FzSet::Ave_Max()

```

{
    int i;
    double mx_mem;
    mx_mem= Get_Max();
    double domain_width;
    double member_value;
    double s1=0.0;
    double s2=0.0;
    domain_width= domain[1]-domain[0];
    for( i=0; i<VECMAX; i++)
    {
        member_value = domain[0] + (float)i * domain_width / VECMAX;
        if( mu_wrk[i] >= mx_mem )
        {
            s1+= mu_wrk[i];
            s2+= mu_wrk[i]*member_value;
        }
    }
    if( s1 == 0 ) s1 =999999.9;
    return( s2/s1 );
};    // Block: 152
//      ++++++
//      Note:determine The Maximum Membership Value
//

```

double FzSet::Max_Edge()

```
{
  int i;
  double mx_mem;
  mx_mem= Get_Max();
  double domain_width;
  double member_value;
  domain_width= domain[1]-domain[0];
  i=0;
  member_value = domain[0];
  while( mu_wrk[i] < mx_mem )
  {
    member_value = domain[0] + (float)i * domain_width / VECMAX;
    i++;
  }
  return( member_value );
}; // Block: 153
//      ++++++
//      Note:assess The Degree Of Truth Given A Scaler Value
//
```

double FzSet::Degree_of_Truth(double scaler)

```
{
  double domain_width;
  double member_value;
  double truth_val;
  int i;
  truth_val= 0.0;
  domain_width= domain[1]-domain[0];
  for( i=0; i<VECMAX; i++)
  {
    member_value = domain[0] + (float)i * domain_width / VECMAX;
    if( member_value > scaler )
      return( truth_val );
    truth_val= mu_wrk[i];
  }
  return( truth_val );
}; // Block: 154
//      ++++++
//      Note: Determine The Maximum Membership Value
//
```

void FzSet::Fz_Initialize()

```
{
  if( strcmp( C_typ,"gauss" ) == 0 )
    Setup_Gauss();
  else if( strcmp( C_typ,"linear" ) == 0 )
    Setup_Linear();
  else if( strcmp( C_typ,"sigmoid" ) == 0 )
    Setup_Sigmoid();
  else if( strcmp( C_typ,"triangular" ) == 0 )
    Setup_Tri();
  else if( strcmp( C_typ,"uniform" ) == 0 )
    Setup_Uniform();
  else
    cerr << "Error: (FzSet Input) unacceptable form option : " << C_typ << "\n"
    << flush;
  return;
}; // Block: 155
//      ++++++
//      Note:determine the maximum membership value
//
```

double FzSet::Get_Max()

```
{
  int i;
  double max_mem=0.0;
  for( i=0; i<VECMAX; i++ )
    if( mu_wrk[i] > max_mem ) max_mem = mu_wrk[i];
  return( max_mem );
}; // Block: 156
//      ++++++
//      Note:renormalize the fuzzy set
//
```

void FzSet::Normalize_FZS()

```
{
  int i;
  double norm_val;
  norm_val = FzSet::Get_Max();
  for( i=0; i<VECMAX; i++ )
    mu_wrk[i] = mu[i]/norm_val;
  return;
}; // Block: 157
//      ++++++
```

```
//      Note:renormalize the fuzzy set
//
```

void FzSet::Reset_Wrk()

```
{
  int i;
  for( i=0; i<VECMAX; i++ )
    mu_wrk[i] = mu[i];
  return;
}; // Block: 158
//      ++++++
//      Note:intiialize membership function to a beta function
//
```

void FzSet::Setup_Beta()

```
{
  return;
}; // Block: 159
//      ++++++
//      Note:intiialize membership function to a customized input function
//
```

void FzSet::Setup_Custom()

```
{
  return;
}; // Block: 160
//      ++++++
//      Note:intiialize membership function to a gauss function
//
```

void FzSet::Setup_Gauss()

```
{
  double domain_width;
  double thiscaler,gausspt;
  double center, sigma, tmp;
  int i;
  center= parms[0];
  sigma = parms[1];
  tmp = sigma*pow(2*PI,0.5);
  domain_width= domain[1]-domain[0];
  for( i=0; i<VECMAX; i++)
  {
    thiscaler= domain[0] + (float)i * domain_width / (VECMAX-1);
```

```

        gausspt= -pow( ((center-thiscaler)/sigma),2);
        mu[i]= exp( gausspt/2 );
    }
    return;
};    // Block: 161
//      ++++++
//      Note:intiialize membership function to a linear function;also provides
//      for the capability of puting sholders on the linear functions.

```

void FzSet::Setup_Linear()

```

{
    double slope_width;
    double domain_width;
    double member_value;
    double lo, hi;
    int i;
    lo= parms[0];
    hi= parms[1];
    slope_width = hi-lo;
    domain_width= domain[1]-domain[0];
    for( i=0; i<VECMAX; i++)
    {
        member_value = domain[0] + (float)i * domain_width / VECMAX;

        if( member_value > hi )
            mu[i]= 1.0;
        else if( member_value > lo && member_value <= hi )
            mu[i]= (member_value - lo) / slope_width;
        else
            mu[i]= 0.0;
    }
    if( conv_trend == DECREASE )
        for( i=0; i<VECMAX; i++ )
            mu[i]= 1.0 - mu[i];
    return;
};    // Block: 162
//      ++++++
//      Note:intiialize membership function to a pi function
//

```

void FzSet::Setup_Pi()

```

{

```



```

return;
}; // Block: 163

```

```

//          ++++++
//          Note:initialize membership function to a sigmoid function
//

```

void FzSet::Setup_Sigmoid()

```

{
double slope_width;
double domain_width;
double member_value;
double tmp1, left, flexpoint, right;
int i;

left= parms[0];
flexpoint= parms[1];
right= parms[2];
slope_width = right-left;
domain_width= domain[1]-domain[0];

for( i=0; i<VECMAX; i++ )
{
member_value = domain[0] + (double)i * domain_width / VECMAX;
if( member_value >= right )
mu[i]=1.0;
else if( member_value > flexpoint && member_value < right )
{
tmp1 = (member_value-right)/slope_width;
mu[i]= 1-(2*(pow(tmp1,2)));
}
else if( member_value >= left && member_value <= flexpoint )
{
tmp1 = (member_value-left)/slope_width;
mu[i]= (2*(pow(tmp1,2)));
}
else
mu[i]=0.0;
}

if( conv_trend == DECREASE )
for( i=0; i<VECMAX; i++ )
mu[i]= 1.0 - mu[i];

```

```

return;
}; // Block: 164
//      ++++++
//      Note:initialize membership function to a triangular function
//

void FzSet::Setup_Tri( )

{
double domain_width;
double member_value;
double tmp1,tmp2, left, flexpoint, right;
int i;

left= parms[0];
flexpoint= parms[1];
right= parms[2];
tmp1 = right-flexpoint;
tmp2 = flexpoint-left;
domain_width= domain[1]-domain[0];
for( i=0; i<VECMAX; i++)
{
member_value = domain[0] + (float)i * domain_width / VECMAX;
if( member_value > left&& member_value <= flexpoint )
mu[i]= (member_value - left) / tmp2;
else if( member_value > flexpoint&& member_value < right )
mu[i]= 1.0 - (member_value - flexpoint) / tmp1;
else
mu[i]= 0.0;
}
return;
}; // Block: 165
//      ++++++
//      Note:initialize membership function to a uniform distribution function
//

```

```

void FzSet::Setup_Uniform( )

{
double domain_width;
double member_value;
double lo, hi;
int i;
lo= parms[0];
hi= parms[1];
domain_width= domain[1]-domain[0];

```

```
for( i=0; i<VECMAX; i++ )
{
    member_value = domain[0] + (float)i * domain_width / VECMAX;
    mu[i]= 0.0;
    if( member_value <= hi && member_value >= lo )
        mu[i]= 1.0;
}
return;
};    // Block: 166
```

```
//  +++++ Fuzzy Variable Operations ( FzVariable ) +++++
```

```
class FzVariable
```

```
{
//      +++++
//
// Com 47   Defines The Characteristics of Fuzzy Variables
//
//          Fuzzy variables, FzVariable, are the descriptors for the model
//          being developed. They may represent time, temperature, or
//          service life. Fuzzy variables are typically used in the "FzDecision
//          class" and are defined as a linked list. Fuzzy variables are
//          associated with fuzzy sets described by the "FxSet class".
//
// Com 48   Member functions include:
//          Find_FzHedge, Find_FzSet which are used to locate
//          characteristics for hedges and fuzzy sets used in the rule model.
//          Reset_FS
//          Variable_Output, & Variable_Setup, which provide functions
//          for loading and printing fuzzy variables.
//
//          Fuzzy variables represent a characteristic of the problem. We
//          normally recognize that the real set of numbers are members of
//          the set temperature we do not think in terms of sets. In the case
//          of fuzzy variables we also must define the space and membership
//          associated with a variable. For this reason we define the
//          levels/descritization of the variable through fuzzy sets, identifying
//          a range of applicability, modifiers or hedges, and in this specific
//          implementation, the values used in the exercise of the system
//          model.
//          Loading of data is controlled or orchestrated by this class.
//
// Com 49   Input format is as follows:
//
//          variables
//          MISSION_PROFILE
//          scaler 6 10 30 50 70 90 70
//          fuzzy_set
//          COMPLEX domain 0 100 func_form linear 0 100 increase
//          eoi
//          fuzzy_set
//          ROUTINE domain 0 100 func_form linear 0 100 decrease
//          eoi
//          hedge
```

```

//          VERY power 2.0
//          eoi
//
// Com 50    Keyword is "variables" (keyed on in Load_Model of FzDecision)
//           In this case "MISSION_PROFILE" is the variable name. "scaler"
//           is a local input keyword indicating that 6 test runs will be conducted
//           and the quantitative values to be assigned to MISSION_PROFILE
//           will be, 10 30 50 70 90 70.
//
// Com 51    The next local keyword, "fuzzy_set" triggers input of set information
//           and is described in FxSet class headers. Suffice to say COMPLEX
//           and ROUTINE represent linguistic levels associated with the
//           variable "MISSION_PROFILE".
//
// Com 52    The last local keyword is "hedge" which triggers input of hedge
//           type information. Hedges act as modifiers in the semantic fuzzy
//           model being developed. In this case the modifier allows us to
//           examine linguistic rules associated with
//           VERY COMPLEX MISSION_PROFILE s.
//
//           ++++++
public:
//
//           Define model parameters:
//
    int result_var;
    int result_;
    char* Fz_Var_ID;
    int num_scaler;
    double* scaler;
    double* D_o_T;

    FzVariable* next;
    FzSet* fs;
    FzSet* last_fs;
    FzSet* fs_result;

    Hedge_*hedge;
    Hedge_*last_hedge;
//
//           Define addiitonal member functions:
//
    FzVariable( FzVariable* );
    Hedge_*    Find_FzHedge( char* );
    FzSet*    Find_FzSet( char* );
    void      Help();

```

```

void      Reset_FS( );
void      Variable_Output( ostream& out );
void      Variable_Setup( istream& in );
}; // Block: 167

//      ++++++
//      NOTE:The integers in the constructor must be adjusted to reflect
//      the number of variables and arguments in the model
//

FzVariable::FzVariable( FzVariable *p )

{
//      Define default conditions
//
result_var=FALSE;
result_=FALSE;
Fz_Var_ID = new char[25];
last_fs=0;
last_hedge= 0;
next= p;
scaler= 0;
D_o_T= 0;
//
//      Setup A Result Fuzzy Set For Use In The De-fuzzification Process
//
fs = new FzSet( last_fs );
last_fs= fs;
fs_result= fs;
strcpy( fs->ID, "RESULT_" );
}; // Block: 168
//      ++++++

```

Hedge_ * FzVariable::Find_FzHedge(char* v_nam)

```

{
Hedge_ * rtn_v;
rtn_v=0;
hedge=last_hedge;
while( hedge!=0 )
{
if( strcmp( hedge->ID, v_nam ) == 0 )
rtn_v=hedge;
hedge=hedge->next;
}
}

```

```

    return( rtn_v );
};    // Block: 169
//      ++++++
FzSet* FzVariable::Find_FzSet( char* v_nam )

{
    FzSet* rtn_v;
    rtn_v=0;
    fs=last_fs;
    while( fs!=0 )
    {
        if( strcmp( fs->ID, v_nam ) == 0 )
            rtn_v=fs;
        fs=fs->next;
    }
    return( rtn_v );
};    // Block: 170
//      ++++++
//      Note:Provide output to the terminal to assist in defining the object
//      model and the information required

```

void FzVariable::Help()

```

{
    return;
};    // Block: 171
//      ++++++

```

void FzVariable::Reset_FS()

```

{
    fs=last_fs;
    while( fs!=0 )
    {
        fs->Reset_Wrk();
        fs=fs->next;
    }
    return;
};    // Block: 172
//      ++++++

```

void FzVariable::Variable_Output(ostream& out)

```

{
    out << "\n Fuzzy Variable:" << Fz_Var_ID;
    for( fs=last_fs; fs != 0; fs=fs->next )

```

```

    {
        out << "\nFuzzy set: ";
        out << *fs;
    }

    out << "\n Fuzzy Variable:" << Fz_Var_ID << "\n";
    for( int i=0; i<num_scaler; i++ )
        out << " " << Fz_Var_ID << "[ " << i+1 << " ] " << scaler[i] << "\n";

    out << "\n Fuzzy Variable:" << Fz_Var_ID;
    for( hedge=last_hedge; hedge != 0; hedge=hedge->next )
    {
        out << "\n Fuzzy Hedges: ";
        out << *hedge;
    }
    return;
}; // Block: 173
//      ++++++

```

void FzVariable::Variable_Setup(istream& in)

```

{
    char tmp_in[15];
    in >> Fz_Var_ID;
    in >> tmp_in;
    while( strcmp( tmp_in,"eoi" ) !=0 )
    {
        if( strcmp( tmp_in,"scaler" ) == 0 )
        {
            in >> num_scaler;
            scaler = new double[num_scaler];
            D_o_T= new double[num_scaler];
            int icas=0;
            in >> tmp_in;
            while( strcmp( tmp_in,"eoi" ) !=0 )
            {
                scaler[icas]= double( atof(tmp_in) ) ;
                icas++;
                in >> tmp_in;
                if( icas > num_scaler )
                    cerr << "Error: (Scaler Input) too many values : " << icas << "\n" << flush;
            }
        }
        else if( strcmp( tmp_in,"fuzzy_set" ) == 0 )
        {
            fs = new FzSet( last_fs );
        }
    }
}

```



```

    in >> *fs;
    last_fs= fs;
}
else if( strcmp( tmp_in,"hedge" ) == 0 )
{
    hedge = new Hedge_( last_hedge );
    in >> *hedge;
    last_hedge= hedge;
}
else
{
    cerr << "Error: (FzVariable Input) unacceptable input option : " << tmp_in
    << "\n" << flush;
}
in >> tmp_in;
}
return;
}; // Block: 174

```

DISTRIBUTION:

1	Michael Maglich, SP282 Strategic Systems Programs 1931 Jefferson Davis Hwy. Arlington, Va 22241-5362	1	MS1221	J.S. Rottler, 5400
1	Mike Wagner ITTS 1500 Garden of the Gods Rd. PO Box 7463 Colorado Springs, CO 80933	1	MS1221	A.B. Cox, 5405
1	Dwayne Curtiss ITTS 1111 Jefferson Davis Hwy. Suite 700 Arlington, VA 22202	1	MS0417	W.H. Ling, 5413 Attn: D.M. Fordham, 5413
1	Rick Verbanec Lockheed-Martin Missiles & Space PO Box 3504 89-20/157 Sunnyvale, CA 94089-3504	10	MS0455	M.E. Senglaub, 6232
1	Alex Loewenthal Dept. 2514, Bldg. 611 Plant 10 Lockheed Martin Skunkworks 1011 Lockheed Way Palmdale, CA 93599-2514	1	MS9018	Central Technical Files, 8940-2
1	Wallace Louie NSWC DL Code K-44 17320 Dahlgren Rd. Dahlgren, VA 22448	2	MS0899	Technical Library, 4916
1	Doug P. Anson LANL TSA-5 MS F602 Los Alamos, NM 87545	1	MS0619	Review & Approval Desk, 15102 For Doe/OSTI
1	Dr. C. Christopher Reed The Aerospace Corporation PO Box 92957- M4/943 Los Angeles, CA 90009-2957			
1	MS0457	W.J. Tedeschi, 2001		
1	MS0429	J.H. Stichman, 2100		
1	MS0475	R.C. Hartwig, 2105		
1	MS0475	J.B. Godfrey, 2105		
2	MS0481	T.F. Hendrickson, 2167		
1	MS9005	J.B. Wright, 2200		
2	MS0501	M.K. Lau, 2338 Attn: T.E. Owen, 2338		

THIS PAGE INTENTIONALLY LEFT BLANK.