

**ADVANCED DISTRIBUTED
SIMULATION TECHNOLOGY II
(ADST II)**

**High Level Architecture
Constructive Performance Model
(HLA CPM)**

**DO #0091
CDRL AB02
FINAL REPORT**



19991115 093

For:

United States Army
Simulation, Training, and Instrumentation Command
12350 Research Parkway
Orlando, Florida 32826-3224

By:

Science Applications International
Corporation
12479 Research parkway
Orlando, FL 32826-3248



Lockheed Martin
Information Systems Company
12506 Lake Underhill Road
Orlando, FL 32825

LOCKHEED MARTIN



DTIC QUALITY INSPECTED 4

Approved for public release; distribution unlimited.
UNCLASSIFIED

REPORT DOCUMENTATION PAGE*Form Approved*
OMB No. 074-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing this collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503

1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE 30 JUN 1999	3. REPORT TYPE AND DATES COVERED final	
4. TITLE AND SUBTITLE Advanced Distributed Simulation Technology II (ADST-II) High Level Architecture Constructive Performance Model (HLA CPM)			5. FUNDING NUMBERS N61339-96-D-0002	
6. AUTHOR(S)				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Lockheed Martin Information Systems ADST-II P.O. Box 780217 Orlando FL 32878-0217			8. PERFORMING ORGANIZATION REPORT NUMBER ADST-II-CDRL-HLACPM-9900181	
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) NAWCTSD/STRICOM 12350 Research Parkway Orlando, FL 32328-3224			10. SPONSORING / MONITORING AGENCY REPORT NUMBER	
11. SUPPLEMENTARY NOTES				
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for Public release; distribution is unlimited				12b. DISTRIBUTION CODE
13. ABSTRACT (<i>Maximum 200 Words</i>) The objective of the High Level Architecture (HLA) Constructive Performance Modeling (CPM) project was to develop a performance model that can be used to investigate the ability of the HLA Run-Time Infrastructure (RTI) to support large-scale constructive simulations in training battlefield commanders and their staff. Specifically, the data-driven model that was developed is targeted toward decision support, sensitivity analyses, and evaluating candidate system configurations that use the RTI. The project goal was accomplished. The CPM can be used to support decisions and evaluate candidate configurations that would support a specific RTI scenario. The results of the HLA Constructive Performance Model project demonstrate that a large scale constructive simulation can be modeled in advance of the exercise date. CPM techniques are much more cost effective than a "try it and see if it runs" approach. The detailed model of the RTI that was developed demonstrates the behaviors that an exercise controller should expect to see for a specific simulation architecture.				
14. SUBJECT TERMS STRICOM, ADST-II, HLA, simulation, , Training,				15. NUMBER OF PAGES 411
				16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED		20. LIMITATION OF ABSTRACT

Document Control Information

Revision	Revision History	Date
	Original release	6/30/99
-		
-		
-		
-		
-		
-		
-		
-		
-		
-		
-		
-		
-		
-		

TABLE OF CONTENTS

EXECUTIVE SUMMARY	2
1. INTRODUCTION	3
1.1 PURPOSE	3
1.2 CONTRACT OVERVIEW	3
1.3 TECHNICAL OVERVIEW	3
1.4 REPORT OVERVIEW	4
2. APPLICABLE DOCUMENTS.....	4
2.1 GOVERNMENT	4
2.2 NON-GOVERNMENT	4
3. SYSTEM DESCRIPTION	5
3.1 CPM ARCHITECTURE	5
3.2 CPM TEST SCENARIO	6
3.2.1 <i>Entity Distribution</i>	7
3.2.2 <i>Interest Management</i>	8
3.2.3 <i>CPM Software</i>	9
3.3 DESCRIPTION OF SYSTEM COMPONENTS	11
3.3.1 <i>Network Representation</i>	11
3.3.2 <i>RTI Model</i>	11
3.3.2.1 RTI Management Services Model	11
3.3.2.2 Service Performance Measures.....	12
3.3.2.3 Time Management	13
3.3.4 <i>SIM Model</i>	14
4. MODELS VERIFICATION	16
5. UTILIZATION EXAMPLE	18
6. CONCLUSIONS AND FUTURE RESEARCH	20
7. ACRONYMS.....	21
APPENDIX A – INDEX TO METHODS BY FILES AND INDEX TO FILES BY METHODS.....	A-1
APPENDIX B – CPM CODE.....	B-1

List of Figures

FIGURE 1. CPM MAJOR COMPONENTS.....	5
FIGURE 2. HLA CPM TEST FEDERATION ARCHITECTURE.....	6
FIGURE 3. SCENARIO FORCE LAYDOWN STRUCTURE	7
FIGURE 4. FEDERATE DISTRIBUTION	8
FIGURE 5. THE STRUCTURE OF THE CPM SOFTWARE	10
FIGURE 6. RTI SERVICES REPRESENTED IN CPM.....	12
FIGURE 7. SIMULATION MODEL COMMAND AND CONTROL STRUCTURE	15
FIGURE 8. CPM VERIFICATION OUTPUT DISPLAY.....	17
FIGURE 9. EXAMPLE SHOWING TIME MANAGEMENT.....	17
FIGURE 10. CPU UTILIZATION FOR EXAMPLE SCENARIO.....	18
FIGURE 11. CPU USAGE STARTUP TRANSIENT EXPANDED.....	19

List of Tables

TABLE 1. OBJECT UPDATES AND INTERACTIONS BY REGION	9
TABLE 2. INDEX TO CPM SOURCE FILES	10
TABLE 3. PRAIRIE WARRIOR 94 DATA	16
TABLE 4. DISTRIBUTION OF ENTITIES ACROSS NODES	20

EXECUTIVE SUMMARY

The objective of the High Level Architecture (HLA) Constructive Performance Modeling (CPM) project was to develop a performance model that can be used to investigate the ability of the HLA Run-Time Infrastructure (RTI) to support large-scale constructive simulations in training battlefield commanders and their staff. Specifically, the data-driven model that was developed is targeted toward decision support, sensitivity analyses, and evaluating candidate system configurations that use the RTI. The project goal was accomplished. The CPM can be used to support decisions and evaluate candidate configurations that would support a specific RTI scenario.

The results of the HLA Constructive Performance Model project demonstrate that a large scale constructive simulation can be modeled in advance of the exercise date. CPM techniques are much more cost effective than a "try it and see if it runs" approach. The detailed model of the RTI that was developed demonstrates the behaviors that an exercise controller should expect to see for a specific simulation architecture.

This Final Report includes a top-level description of the Constructive Performance Model that was developed and discusses its major system components. It discusses the initial scenario that was used to exercise the model and presents preliminary simulation data from executing this scenario.

The initial testing scenario was a federation of 27,800 entities distributed among 10 federates. The federates were each represented as a computational node with supporting federate and RTI ambassadors. All of the federates were modeled as time-regulating and time-constrained [3], with the simulation models responsible for resolving all causality errors [2]. The number of entities run on each federate was allocated based on forward edge of battle boundaries and was not split below the battalion level. The static data distribution management (DDM) test case (i.e., slow moving ground units scenario [1]) was employed and the static DDM regions were allocated based on the entity activity level and force ratios.

The model was constructed focusing on several key performance measures. These measures include CPU utilization, network latencies, the amount of real time required to perform time advance grants, latency of an update/reflect operation with respect to simulation time, and the amount of simulation time to send and receive interactions.

1. INTRODUCTION

1.1 Purpose

The purpose of this final report is to document the ADST II effort that developed the HLA Constructive Performance Model and to provide insight into the construction of the model. The report includes a top-level description of the CPM, discusses the major system components, and presents preliminary simulation data from executing the initial scenario.

1.2 Contract Overview

The CPM HLA project was performed as Delivery Order (DO) #0091 under the ADST II contract with STRICOM. The purpose of this DO was to provide a focused effort for the analysis and evaluation of the performance of the RTI as it might be used by a large distributed constructive simulation. Such an effort was needed because prior performance modeling done by other researchers had focused on "characteristic HLA federations" rather than on specific implementations. A performance model was needed that could address large distributed simulations using the RTI as the infrastructure. The goal of the resulting performance model was to capture the relevant performance characteristics of a large constructive simulation, allowing federation designers to model and study the federation performance characteristics prior to constructing the federation. It was also deemed critical that performance modeling support the continuous evolution of interoperability standards and processes.

1.3 Technical Overview

The High Level Architecture Constructive Performance Modeling delivery order developed a constructive performance model to assist in evaluating the ability of the HLA Run-Time Infrastructure to support a large-scale, distributed simulation that addresses a training audience composed of battlefield commanders and their staff. The data-driven model that was developed is targeted toward evaluating candidate system configurations that use the HLA RTI.

Another objective of the CPM project was that the resulting model could be used to conduct sensitivity analyses to identify potential performance constraints of the HLA RTI paradigm. An initial test scenario was used to verify the constructive performance model and to identify directions for improvement of the CPM model representations (i.e., RTI, network, and simulation models).

During a training exercise that has a large training audience that interacts with the constructive simulation, the question "Will the simulation keep up with wall clock time?" often comes up. The answer to the question depends on how the simulation models are distributed among the available hardware, the speed of the network connections between the hardware, the data distribution method, and the time management method that is employed. The constructive performance model that was developed addresses each of these issues. The model assumes that the constructive simulation models are event driven and operate in simulation time that must remain near to wall clock time. 'Near' was considered to be within 30 seconds. 'Near' should correspond to the update rate of any organic C4I equipment that is used by the training audience.

The technical approach to the Constructive Performance Model consisted of building a scenario generation tool, an entity distribution tool, an algorithmically correct HLA RTI model, and an entity simulation engine. All of the tools were integrated into a single executable that operates in

two phases. The first phase constructs the scenario and distributes the two opposing military forces to the number of defined regions, by percent of force for each region. The regions consist of warfighter regions and support regions. The regions are allocated among the defined number of federates. Each federate is viewed as being a single CPU node in a network. The second phase executes the model for a specified number of simulation seconds. The startup transients of object publication, subscription, creation, and discovery occur at the beginning of this period. No data was collected about the transient portion. The remainder of the execution phase represents the execution of the simulation. This is when performance data was collected. The average frequency of occurrence of events (sense, move, fire, and report) that the model used was based on WARSIM Simulation of the Architecture research, and on a Prairie Warrior 94 training exercise.

1.4 Report Overview

The next section lists the government and non-government document references applicable to this project. Section 3 contains descriptions of the overall CPM architecture, the test scenario, and the system components. Model verification is addressed in Section 4. A utilization example is presented in Section 5. Section 6 summarizes the conclusions and addressed potential future work. Appendix A provides indices to the files and methods within the CPM source code. Appendix B contains a listing of the CPM source code.

2. APPLICABLE DOCUMENTS

2.1 Government

Defense Modeling and Simulation Office "Run-Time Infrastructure (RTI) 2.0- Software Design Document (Version 2)." Contract # N61339-97-C-0073.

2.2 Non-Government

[1] Cohen, D. and Kemkes, A. "Applying User-Level Measurements to the RTI 1.3 Release." Spring Simulation Interoperability Workshop, March 1998.

[2] Merritt, W. "Bounding CPU Utilization as a Part of the Model Design and the Scenario Design of a Large-Scale Military Training Simulation." 1998 Winter Simulation Conference, Washington, D.C.

[3] Defense Modeling and Simulation Office. "Standard for Modeling and Simulation (M&S), High Level Architecture (HLA) - Federate Interface Specification - Runtime Interface (RTI 1.3), Draft 1," April 1998.

[4]. Bachinsky, S. "RTI 2.0 Architecture." Spring Simulation Interoperability Workshop, March 1998. 98S-SIW-150.

[5] Defense Modeling and Simulation Office "DoD Modeling and Simulation Master Plan," <http://www.dmsi.mil/docslib/mspolicy/msmp>, October 1995. DoD 5000.59-P.

[6] Agrawal, D. and Agre, J. "Replicated Objects in the Time Warp Simulations." Proceedings of the 1992 Winter Simulation Conference.

[7] Fujimoto, R. "Parallel Discrete Event Simulation." Communications of the ACM, Vol. 33 (70):30-53, 1990.

[8] Steinman, J. "Breathing Time Warp." Proceedings of the 1993 Workshop on Parallel and

Distributed Simulation, p.109-118.

[9] Defense Modeling and Simulation Office "Run-Time Infrastructure (RTI) 2.0 – Software Design Document (Version 2)." Contract # N61339-97-C-0073.

[10] Mattern, F. "Efficient Algorithms for Distributed Snapshots and Global Virtual Time Approximation." Journal of Parallel and Distributed Computing, 1993.

[11] Dey, A. "An Executable Model of the RTI." Spring Simulation Interoperability Workshop, March 1998. 98S-SIW-246.

[12] Masakazu, F. "Performance Evaluation Model of HLA-RTI and Evaluation of result of eRTI." Fall Simulation Interoperability Workshop, 1997. 97F-SIW-187.

[13] Morse, K. "Issues in the Relationship Between HLA's Declaration Management and Data Distribution Management Services." Fall Simulation Interoperability Workshop, 1997. 97F-SIW-083.

[14] McCormack, J., Weckenmann, C., Lowe, G., and Merritt, W. "Development of a HLA Constructive Performance Model." Spring Simulation Interoperability Workshop, March 1999. 99S-SIW-146.

3. SYSTEM DESCRIPTION

3.1 CPM Architecture

The HLA CPM architecture includes three major components: the network model, the simulation model, and the RTI model (Figure 1). The network model includes a representation of the network hardware and the data transport modes. The network hardware representation models single CPU nodes with I/O cards. The I/O cards are modeled as operating asynchronously and supporting infinite queues.

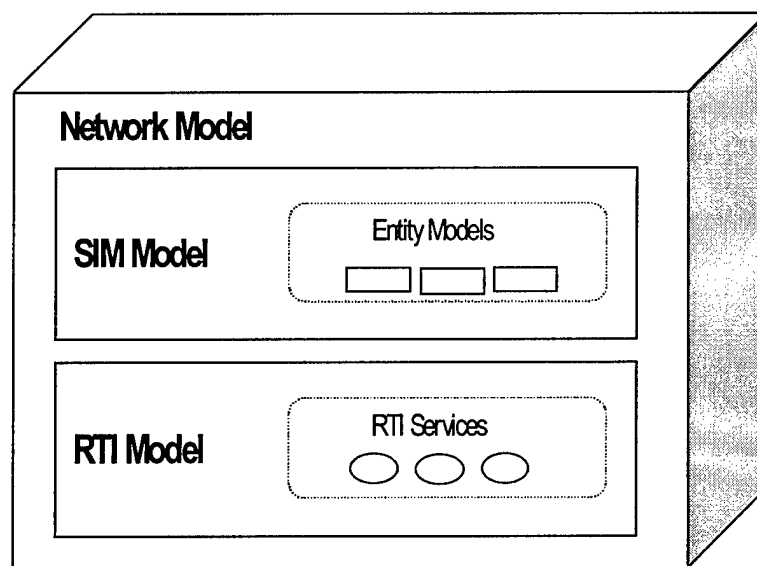


Figure 1. CPM Major Components

The simulation (SIM) model uses a one-second time step, with immediate event resolution based on the time-stamp order (TSO) of the messages received, when computing time advance grants. The SIM model contains a collection of entity models. The entity models include a representation of the unit hierarchies, the tactical communication network characteristics of the warfighting units, as well as the units' physical and behavioral characteristics. The unit entities of the SIM model employ extended entity states. An extended entity state is one that is valid for a defined interval of simulated time.

The RTI model is based on the HLA Interface Specification Version 1.3 and the RTI Next Generation (NG) design [3,4,9]. The RTI model includes a representation of such RTI services as create, join, publish, subscribe with region, register with region, send interaction with region, update attributes, and time advance request. The RTI services provide the interface protocol between the SIM model and the RTI model.

3.2 CPM Test Scenario

The initial testing scenario for the HLA CPM consisted of a federation of 10 federates (Figure 2). Each federate was represented as a computational node with supporting federate and RTI ambassadors. A computational node may represent a computer or a coupled system of computers that communicate to the rest of the federation through its RTI interface. The RTI Executive and the Federation Execution (Fedex) processes were contained within Federate 1. All of the federates were time-regulating and time-constrained (TR&C). Federate 1 was configured to be the lower bound time stamp (LBTS) controller for time management operations of the federation.

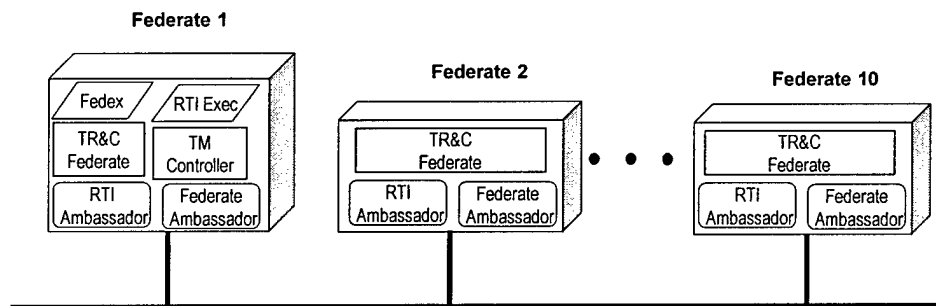


Figure 2. HLA CPM Test Federation Architecture

3.2.1 Entity Distribution

The number of entities run on each federate was allocated based on their activity level. Entities at a high activity level are those located at the forward edge of the battle, while entities at medium/low activity levels perform support functions and rear operations behind the forward forces. Figure 3 presents a graphical representation of the overall force laydown structure.

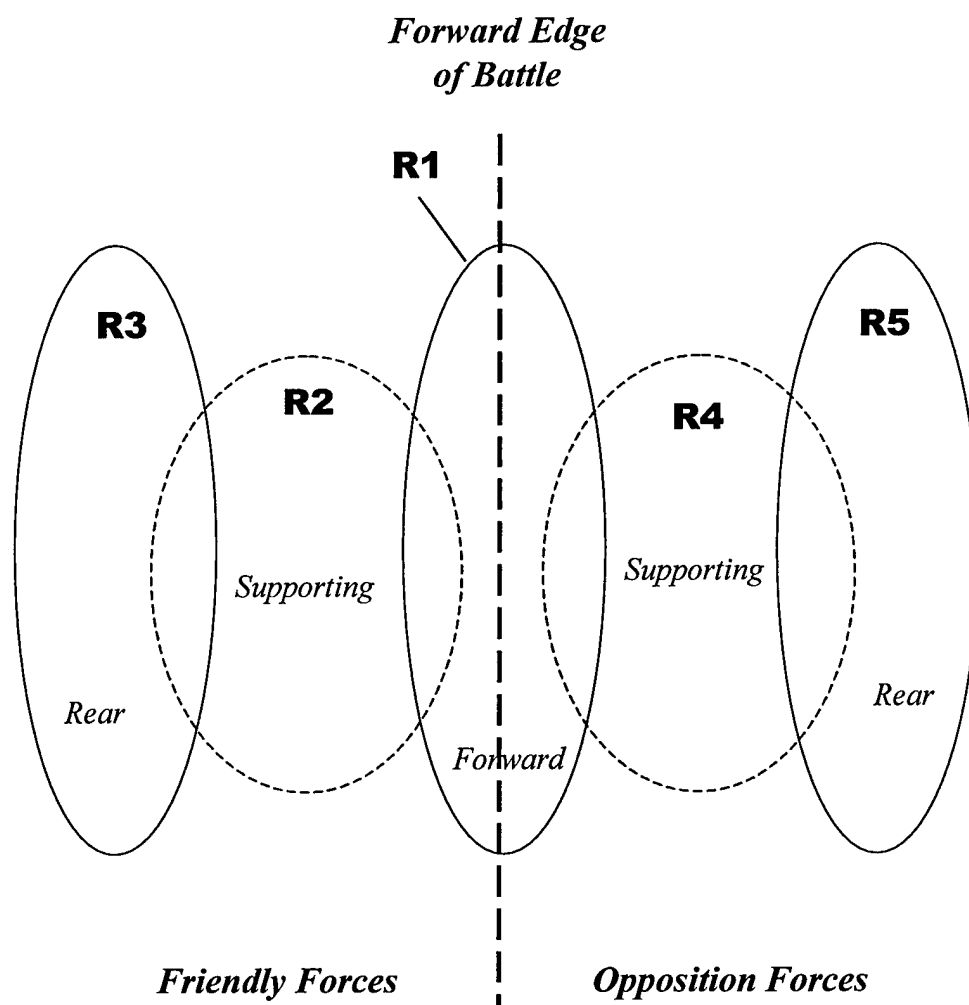


Figure 3. Scenario Force Laydown Structure

Region R1 contains the highly active entities of the forward units and has interactions with the support forces for re-supply activities. Regions R2 and R4 represent supporting forces that transport supplies to the forward units from the rear forces. Regions R3 and R5 represent rear and adjacent forces that generally operate at a low level of activity.

Figure 4 presents the distribution of federates across the battlefield. Federates F1, F2, and F3 contain high and medium activity entities of the front line warfighting units. Federates F4, F5,

and F6 represent the medium and low activity entities of the friendly supporting and rear forces while Federates F7, F8, and F9 represent the corresponding entities for the opposition forces. Federate F10 is held in reserve.

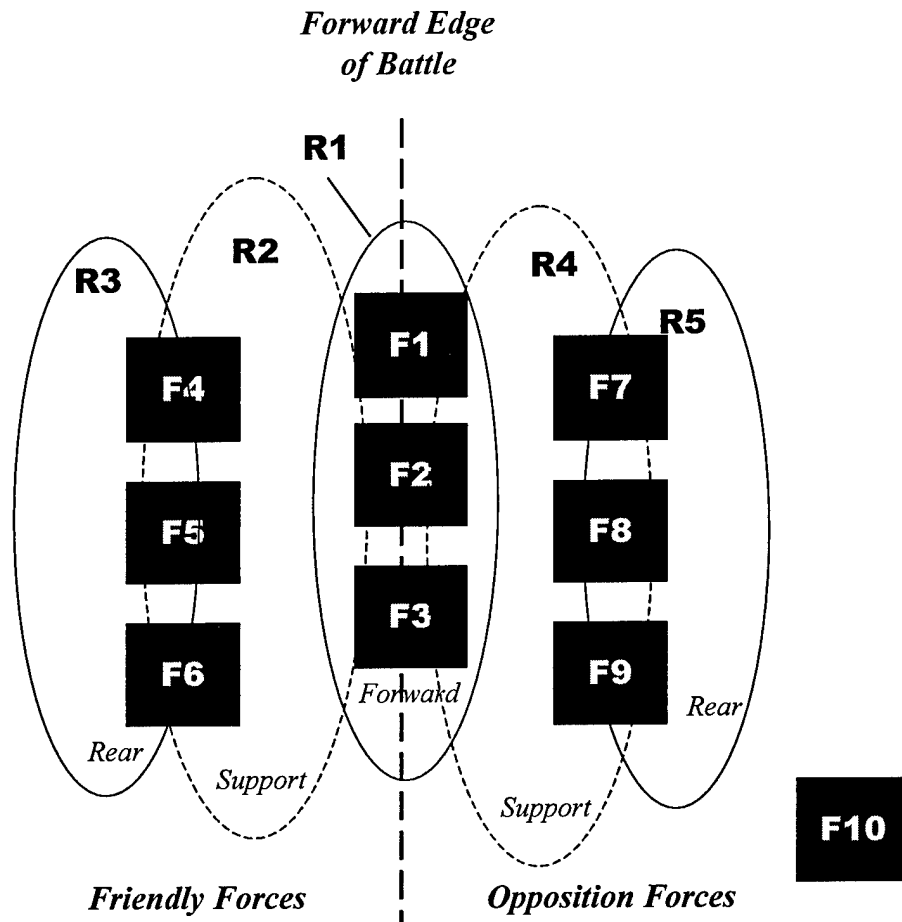


Figure 4. Federate Distribution

3.2.2 Interest Management

A static DDM case (i.e., "slow moving ground units" scenario [1]) was employed for the test scenario and the static DDM regions were allocated based on the entity activity level and force ratios (Figure 3). As previously discussed, five DDM regions were defined, and the number of regions defined per computational node can vary, for example, by having more than one region per node, or by having one region cover more than one computation node. In general, entities with a high activity level will have smaller DDM regions (i.e., with fewer entities) than medium activity entities, and low activity entities will have the largest DDM regions (i.e., more entities contained in a region). This of course implies, for the purposes of this scenario, that terrain will be allocated to each computational node. The terrain was allocated to each node using an algorithm executed during model initialization. Overlapping terrain between nodes is allowed.

Table 1 presents each of the five regions and contains a list of the supported objects and interactions for the initial scenario. Registration for object updates and interactions is performed at the federate level by regions. For example, a warfighter platoon located on Federate 1 in Region 1 publishes and subscribes to all objects. The platoon will also subscribe to all interactions and will publish all interactions except supplies.

Table 1. Object Updates and Interactions by Region

Region	Object Updates	Interactions
R1	Move(PS), Mission(PS) Unit Type (PS) Plan(PS) Health(PS)	Fire (PS) Orders(PS) Reports(PS) Sense(PS) Supply(S)
R2	Move(PS) Mission(S) Health(PS)	Orders(S) Reports(P) Supply(PS)
R3	Move(PS) Mission(S) Health(PS)	Orders(S) Reports(P) Supply(PS)
R4	Move(PS) Mission(S) Health(PS)	Orders(S) Reports(P) Supply(PS)

3.2.3 CPM Software

The CPM software (Figure 5) contains four principle files: Sim.c, EventManager.c, rti_manager.c, and SimModel.c. Sim.c is the main and from it GrowArmy.c routines are called to build a force for the scenario. In GrowArmy.c the regions are created and the simulated military units and entities are allocated to the regions. The allocation and region management routines are contained in the file Regions.c. Also from GrowArmy.c the federate nodes are created and the regions are allocated to the nodes. FedNodes.c contains these methods. The final federate setup operations of declaring the intent to publish and the subscription requirements are completed from methods in FedNodes.c. The federation creation and the joining of the federates are accomplished by direct calls in the body of Sim.c.

EventManager.c is the object of the main loop of Sim.c. The specific method used is that EventManager controls the selection of the next model event. There are three categories of events: an IO operation, a RTI service, and an Entity simulation event. The IO operations are handled in the body of EventManager. The method rtingr_RTIEvent from file rti_manager is called to handle both incoming and outgoing RTI service effects. An object attribute update is an outgoing operation that becomes a reflect with an incoming update to the receiving federate [3]. The Entity simulation event category addresses the events that occur from one entity to another entity, or between an entity and the incoming RTI federate services. The method SimModel in SimModel.c handles the state of the entites.

Table 2 lists the CPM source files and provides a brief description of each.

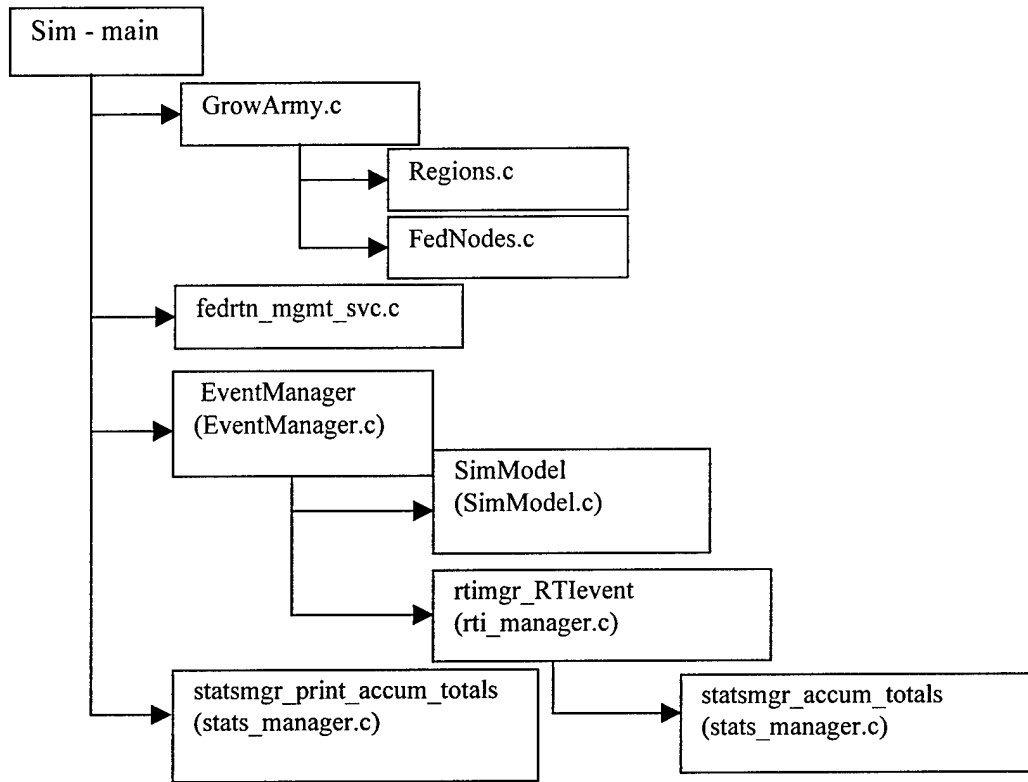


Figure 5. The Structure of the CPM Software

Table 2. Index to CPM Source Files

File name	Description	Appendix page #
EventManager.c	Model event manager	B- 7
event_manager.c	RTI event manager	B- 198
FedNodes.c	Creates nodes and distributes the regions to nodes	B- 53
Regions.c	Creates regions and assigns units to regions	B- 111
ReadOrdr.c	Creates unit to unit orders if needed	B- 101
SOAcreat.c	Creators for the data structures used	B- 140
SOAdestr.c	Destructors for the data structures used	B- 151
InterestGroups.c	Develops associations between specific units on opposing forces	B- 98
FilterUnits.c	Filters used to group units for adds, joins, or removals from Regions	B- 74
GrowArmy.c	Generates an army given a number of battalions and ratios of forces	B- 83
init_rti.c	Initializes the RTI model data types	B- 219
rti_manager.c	Manages the RTI service execution	B- 267
io_manager.c	Manages the RTI to network interface	B- 222
stats_manager.c	Manages the statistics collection	B- 305

object_mgmt_svc.c	Object management	B-	241
data_distrib_mgmt_svc.c	RTI Data Distribution Management	B-	188
declar_mgmt_svc.c	RTI Declaration Management	B-	191
fedrtn_mgmt_svc.c	RTI Federation Management	B-	205
SimModel.c	Simulation Model for the two military forces	B-	158
UnitEcheleon.c	Supports definition and display of hierarchical organization of the forces	B-	177
UnitCharFile.c	Supports definition and display of unit characteristics	B-	172
Sim.c - main	The place to start. Initializes scenario and calls EventManager for the execution of the model. This also controls the number of replicates of the experiments	B-	153
time_mgmt_svc.c	Time Management services	B-	323
mood.c	Xwindow utilities	B-	226

3.3 Description of System Components

3.3.1 Network Representation

The network representation used was a model of an Ethernet local area network (LAN) with maximum throughput of 100 Mbps. Switches were represented as delays fixed at 1 millisecond. This is a best case value that was used to simplify model complexities and support verification of interactions. The reliable transport communication mode was used for relaying information (e.g., entity state and event interactions) between computational nodes.

3.3.2 RTI Model

The model of the RTI implemented for this project references the RTI 1.3 Interface Specification [3] but also contains the concepts outlined in the more recent RTI 1.3NG implementation [4, 9]. However, the RTI model is configurable in its representation of RTI implementations. Most configuration items defined in the RTI Initialization Data (RID) and Federation Object Model (FOM) are parameterized in the model. In the RTI model, the configuration items are set to a distributed federation execution and to Ethernet network type communications.

3.3.2.1 RTI Management Services Model

The RTI model employs all of the RTI Specification's management groups (Figure 6) except the Ownership Management. Services implemented per management group were limited to those that are most often used during a federation's existence and are most applicable to evaluating the performance of the RTI. Services marked with an asterisk in Figure 6 indicate those services for which data were collected for the initial test scenario.

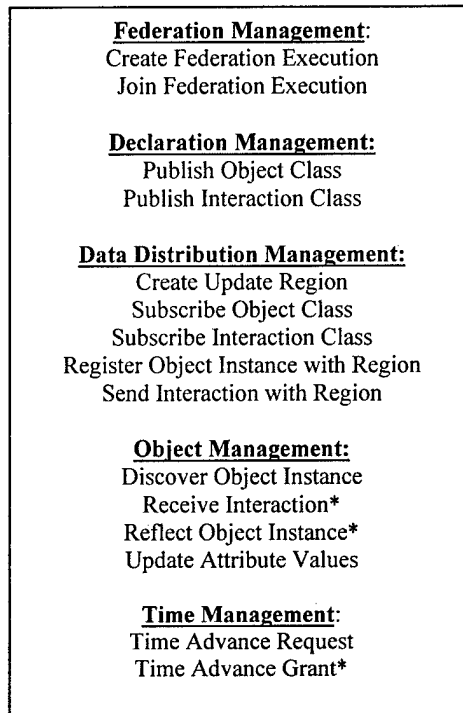


Figure 6. RTI Services Represented in CPM

Object classes, attribute groups, interactions and regions are represented within the RTI model. For the initial version of the RTI model, object attributes were abstracted into object attribute groups where each object class is representative of an object class *plus* a group of attributes. To declare additional attributes for the same object class, another object class is created. Interactions and parameter groups were represented in a similar manner.

The RTI model does not directly implement the aspect of routing spaces for DDM since routing spaces are more indicative of multicast groups. Instead, the RTI model contains a single default routing space and uses regions as subdividers. For simplicity, a region is defined as a fixed area referenced by a number, which abstracts the notation of a dimensioned region in the Interface Specification [3]. Overlapping regions are represented as fixed region numbers. Dynamic regions are currently not supported by the RTI model.

3.3.2.2 Service Performance Measures

During execution, the RTI model collects statistics on the performance of each of the services during the Federation's operation, providing evaluation information on a service's frequency and average execution time per federate. These statistics include the average time to update instance attributes (T_{UPDATE}), the average time to reflect attribute values ($T_{REFLECT}$), and the average total time spent in the system (TOTAL). These statistics are gathered through the use of complexity formulas that consider various factors that occur in the execution of a service. A subset of the formulas used in the RTI model is listed below.

UPDATE INSTANCE ATTRIBUTES service:

$$T_{UPDATE} = V + L_{FOM\_DB} + L_{FEDEX\_DB} + L_{INSTANCE\_DB} + (N_f * L_{SUBSCRIBED\_DB}) + (N_f * S_{REFLECT\_EVENTS})$$

REFLECT ATTRIBUTE VALUES service:

$$T_{\text{REFLECT}} = V + X_{\text{NTSO}}$$

Total Time in System:

$$\text{TOTAL} = T_{\text{UPDATE}} + T_{\text{I/O}} + \text{MAX}_{\text{FEDJ}}(T_{\text{REFLECT}})$$

Where:

T = resource CPU execution time

V = service criteria verification

L = lookup/search time in database; within either the FOM_DB, FEDEX_DB, or INSTANCE_DB.

N_r = number of items in a region resulting from a lookup in the INSTANCE_DB.

N_f = number of federate nodes resulting from a lookup in the SUBSCRIBED_DB.

S = setup of REFLECT events to other nodes

X = transfer time from I/O input to the federate's TSO event queue.

The formulas are used in determining the time for a service to complete execution. Computational times result from accumulation of the factors specific to the execution of each service. The factors that are most variable are those that entail the setup of data and retrieval of database information, since their cost is directly dependent on the existing database size and retrieval methods. Therefore, the more items (e.g., number of federates, regions, and object/instance subscriptions) there are in the database, the more CPU time is needed for the computation. Likewise, the type of search algorithm employed (e.g., hash table, linear search, or binary search) will also affect the amount of CPU time needed for the computation. For example, hash table searches usually find an item more quickly than linear searches. The current RTI model assumes the use of a hash table with searches that execute at a factor of O(n), where n is the number of searches.

3.3.2.3 Time Management

The HLA CPM implements a distributed snapshot algorithm for determining time advances in the time stepped simulation. This LBTS algorithm employs the design from the RTI 1.3 Next Generation (NG) implementation that evolved out of the various Global Virtual Time (GVT) algorithms [2, 6, 7], but more specifically from the work of Mattern's distributed snapshot algorithm [9, 10]. GVT is a property of an instantaneous snapshot of a distributed system. Within the LBTS computation, federate messages and states are color-coded (i.e., by attaching a 'color' tag), permitting capture and accumulation of time stamped messages for the LBTS computation. The LBTS information is transmitted from each of the federation member nodes to the federate acting as the LBTS controller by use of a reduction network configuration as in the RTI 1.3NG design. (In the initial test scenario Federate 1 was designated as the LBTS controller. However, the RTI model allows any federate to be designated as the LBTS controller). A global minima (LBTS) is determined from the collected LBTS information at the LBTS controller. The LBTS controller grants time advances when it determines that the federation is in a stable state where all transient messages have been accounted for.

3.3.4 SIM Model

The HLA CPM simulation model or SIM model assumes that the simulation is part of a single cohesive distributed simulation that is using the HLA RTI to provide an infrastructure. A large single cohesive distributed simulation differs from small standalone simulations in that temporal anomalies, or causality errors, cannot be tolerated. Large distributed simulations focus on commander and staff training, while small disparate simulations focus on equipment training. The simulation must tolerate the latency of distribution, respond in a realistic way to operator commands, and execute operations with enough detail to adequately implement the commander's plans.

Such a simulation is currently being designed for JSIMS, so we are in a discovery phase. Currently, no such simulation exists. CPM permits examination of the quantitative constraints that must be met by such an infrastructure to accomplish interoperability and scalability in an affordable open systems architecture [4]. The HLA CPM was developed to help assess the performance constraints that an infrastructure would need to meet in order to provide this scalability.

The DDM services provide the greatest contribution to achieving this scalability goal by enabling regions to be defined [13]. It is therefore important to represent regions in the SIM model. The SIM model must register instances of objects and interactions with specific regions, and subscribe to class attributes and interactions by regions. In addition, the SIM model represents military behavior of entity models as a function of CPU processing load, network needs in data distribution, and virtual time management. Fundamental military behavior can be grouped into the basic categories of sense, move, shoot, and communicate.

Distributed simulations require data to be distributed in a timely manner to advance simulation time at a realistic rate. The tradeoff between using the computational resource to (1) perform work that contributes to the advance of simulation time, (2) distribute data, and (3) manage time advance, directly affects the scalability of a distributed simulation [6].

A simulation model must include certain factors to adequately investigate a solution space for any specific scenario. Entity models consume some of the computational resources of a node. The frequency at which events and attribute updates must be resolved drives the update rate. Both CPU consumption and update frequency impact the data distribution management. Since the simulation must be coherent (i.e., without causality errors) the representative model must continually track simulation time with respect to real time. In the SIM model the value used for an entity model's CPU consumption is 2 milliseconds per update. The update frequency for events and attributes was based on Prairie Warrior 94 training exercise data.

It is important that the organization of the military force is appropriately represented within the SIM model. The organization affects the extent of a behavior: a 'move' to a battalion size force has a much different computational requirement than a 'move' to a platoon. Therefore, the hierarchy of command and operation is maintained in the SIM model. The representation of command hierarchy in the simulation model is battalion, company, and platoon. The brigade level is the level of the training audience. Whole companies were required to be allocated to a specific federate (i.e., cannot be split across federates). Battalions may be allocated across multiple federates. Figure 7 depicts the hierarchical organization and the communication structure of the units as they are represented in the simulation model.

30 June 1999

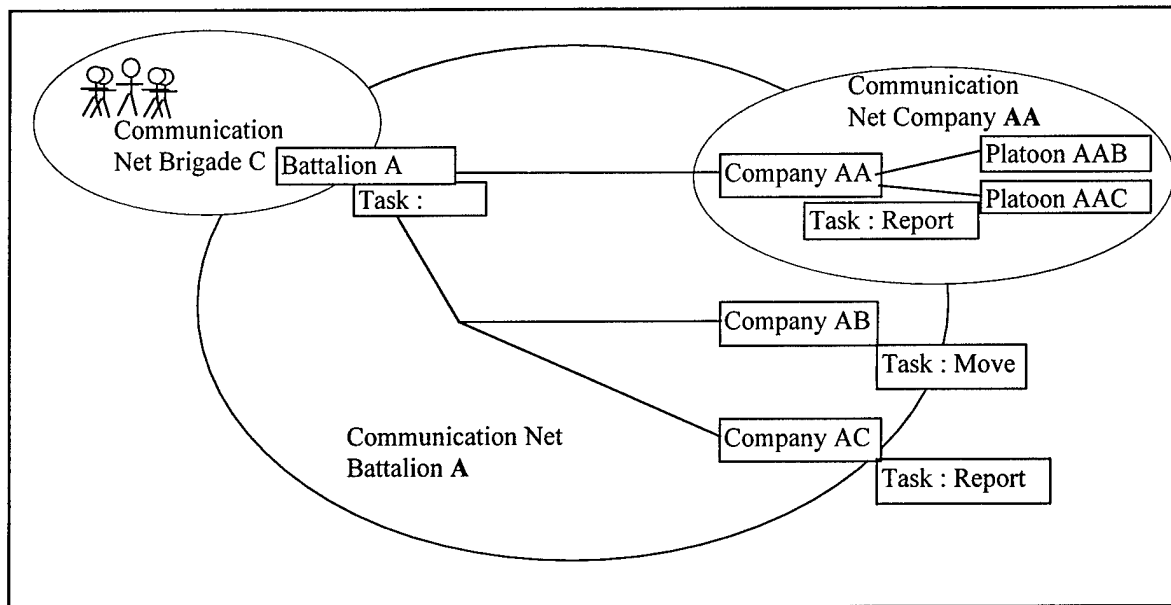


Figure 7. Simulation Model Command and Control Structure

In addition, the SIM model is data driven from input parameter files that are read during initialization. No direct input is required from the user. In current distributed training exercises such as Prairie Warrior, the interface media to the training audience consists of reports. This makes it feasible to use extended entity states in the constructive simulation. An extended entity state is a statement about the activity of an entity that is valid for a specific interval of simulation time. The use of extended entity states is possible because the training objectives align with commander and staff planning and with team building needs, not with equipment training. No joystick type control of vehicles is needed for these types of simulations, and visual displays do not need to be out the window type displays.

The units in such simulations have independent entities or platforms that interact without constant human control. For example, a vehicle following a route is on that route at a specific location at a specific time. If the route is known and distributed through update services, no further calculation is needed related to that entity until an interaction occurs. Interactions would need to be of a nature that could require the entity to change its state. These interactions are delivered as discrete events to the affected units in time stamp order (TSO). In the CPM, entity states are checked once a second to determine if an extended entity state has expired and needs to be updated.

Table 3 contains a summary of the sense, move, shoot, and communicate behaviors from the Prairie Warrior 1994 training exercise. The table contains a listing of the type of behavior, the frequency of the behavior (per minute) and the average number of entities affected by the behavior. The shaded boxes represent average values. This data was used as a basis for the frequency of entity behaviors in the HLA CPM simulation model.

Table 3. Prairie Warrior 94 Data

Category	Frequency/ Minute	Average # Entities Affected
<i>Sensor</i>	0.636	253
<i>Ground Unit Move</i>	1.196	271.5
<i>Unit Company Move</i>	0.360	65.6727
<i>Average Move</i>	0.778	100.4
<i>Explicit Unit Fire</i>	1.115	183
<i>Ground Attack</i>	2.49	170.5
<i>Average Fire</i>	1.8025	176.75
<i>Order</i>	1.476	130
<i>Report</i>	1.409	196.5
<i>Average C2</i>	1.4425	163.25

Every platoon of the initial scenario is allowed to have up to five attribute groups, and is allowed to cause up to five interactions for subscription and registration purposes. Each platoon consists of four platforms. Because of the complexity of large distributed systems and the fact that resources to support a large-scale military training simulation may vary significantly from one exercise scenario to another, the simulation model has a scenario generation and distribution capability. From an input file, the scenario generator is provided with the number of regions of confrontation, the force ratio, number of support regions, and the number of battalions of the forces. The simulation model then uses this information to build the representative armies. The simulation model contains a set of rules that can be easily altered or augmented to establish additional regions for subscription and registration of instances with regions. Assignment to the number of desired federates is carried out by an algorithm that does not allow a company to be divided across multiple nodes.

4. MODELS VERIFICATION

The data generated by any model are only as good as the representative models and the input data used to drive them. In developing a CPM of the HLA RTI, verification of the developed models is a critical first step toward evaluating the usefulness of the generated results. Toward this end we performed preliminary verification of the correctness of the RTI and SIM model algorithms.

To aid in verification of the simulation relationships between the SIM, the RTI model, and the network model, a monitor of the queue management for the federation was developed as shown in Figure 8. Each service time is represented by a line segment color-coded to identify the portion of the model that is using that resource for that time. Figure 9 is an expanded view showing an example of the simulation of Time Management. The call-outs in Figure 9 describe the steps in the time advance algorithm [3]. The simulation model execution is shown in red; the RTI model execution is shown in green. Time requests are shown in blue and time grants are shown in magenta. During initial experimentation and testing of the HLA CPM, the verification display was used extensively to examine the sensitivity of the scenario to factors such as time advance grants.

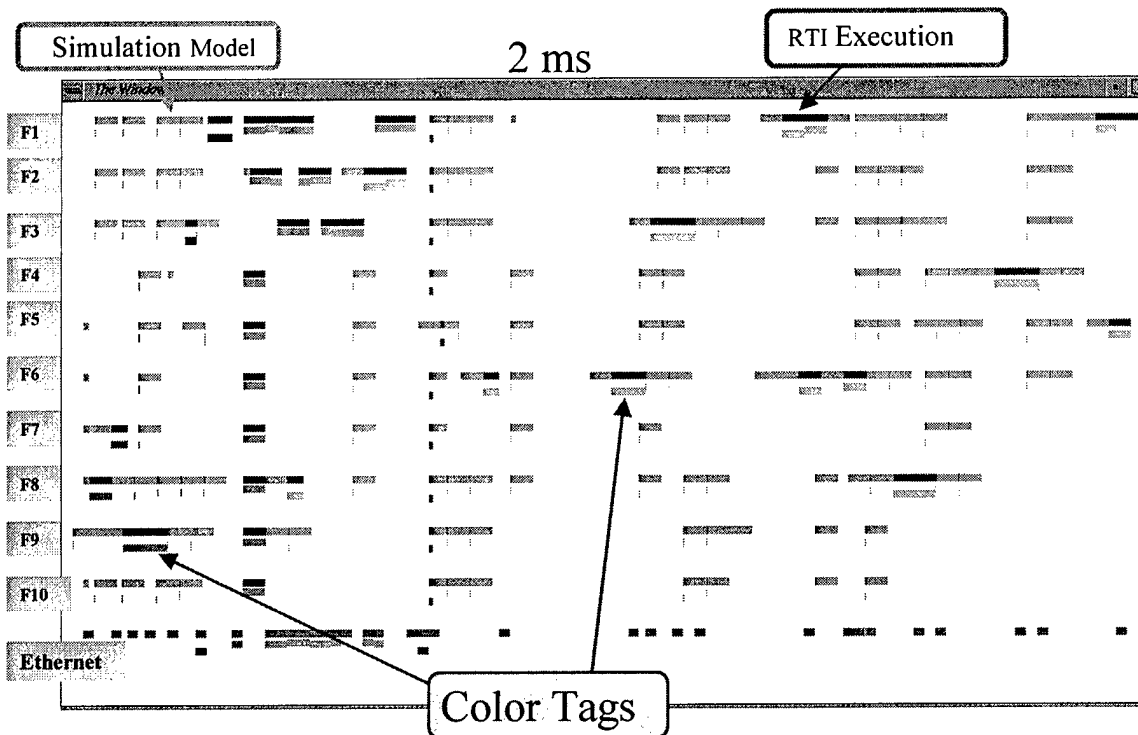


Figure 8. CPM Verification Output Display

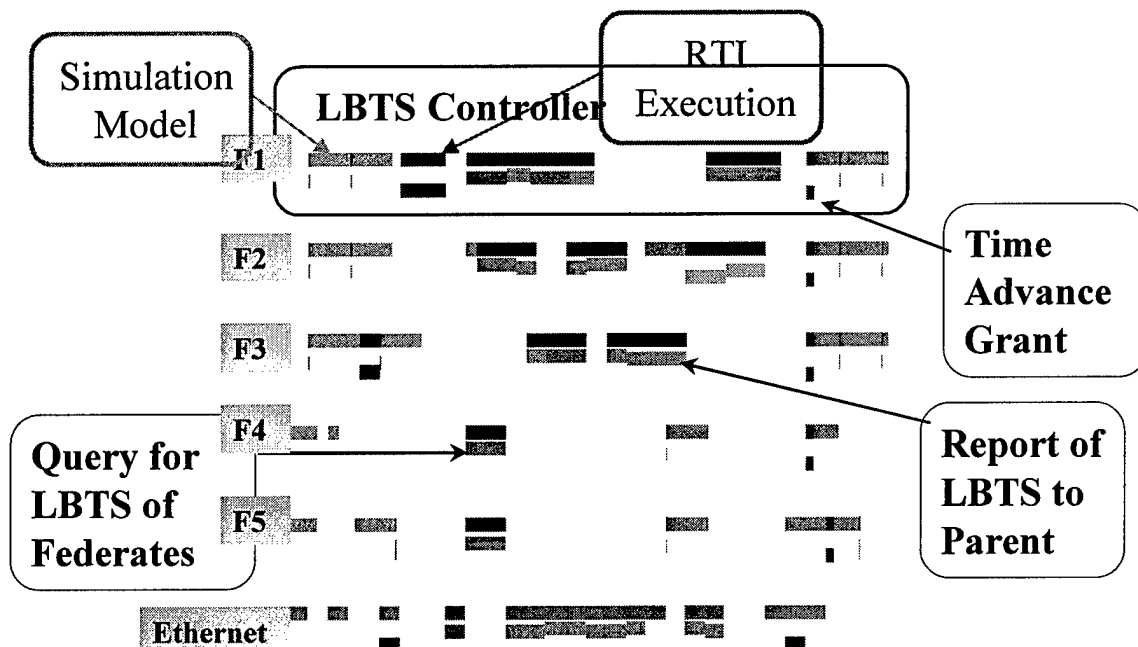


Figure 9. Example Showing Time Management

5. UTILIZATION EXAMPLE

Figures 10 and 11 are provided to demonstrate the CPU utilization on each node. The scenario for this run had time regulation disabled. The distribution of entities to federates for this example is as shown in Table 4. Figure 10 shows the startup transition to a steady state of the CPU utilization. The first 120 seconds are required to work through the initialization events. As can be seen, the initialization period is congested. Investigation of the model behavior showed that the congestion was due to the initial scheduling of events. After 120 seconds, the system reached a condition where there is ample CPU time to execute the simulation. This example was used to demonstrate the complex effects that can be seen by using simulation. If this utilization profile were not acceptable, the initialization portion of the scenario would have to be changed. Thus we have shown the CPM can be used as a decision support tool in planning an exercise. Figure 11 is provided to show a more detailed view of the data sample.

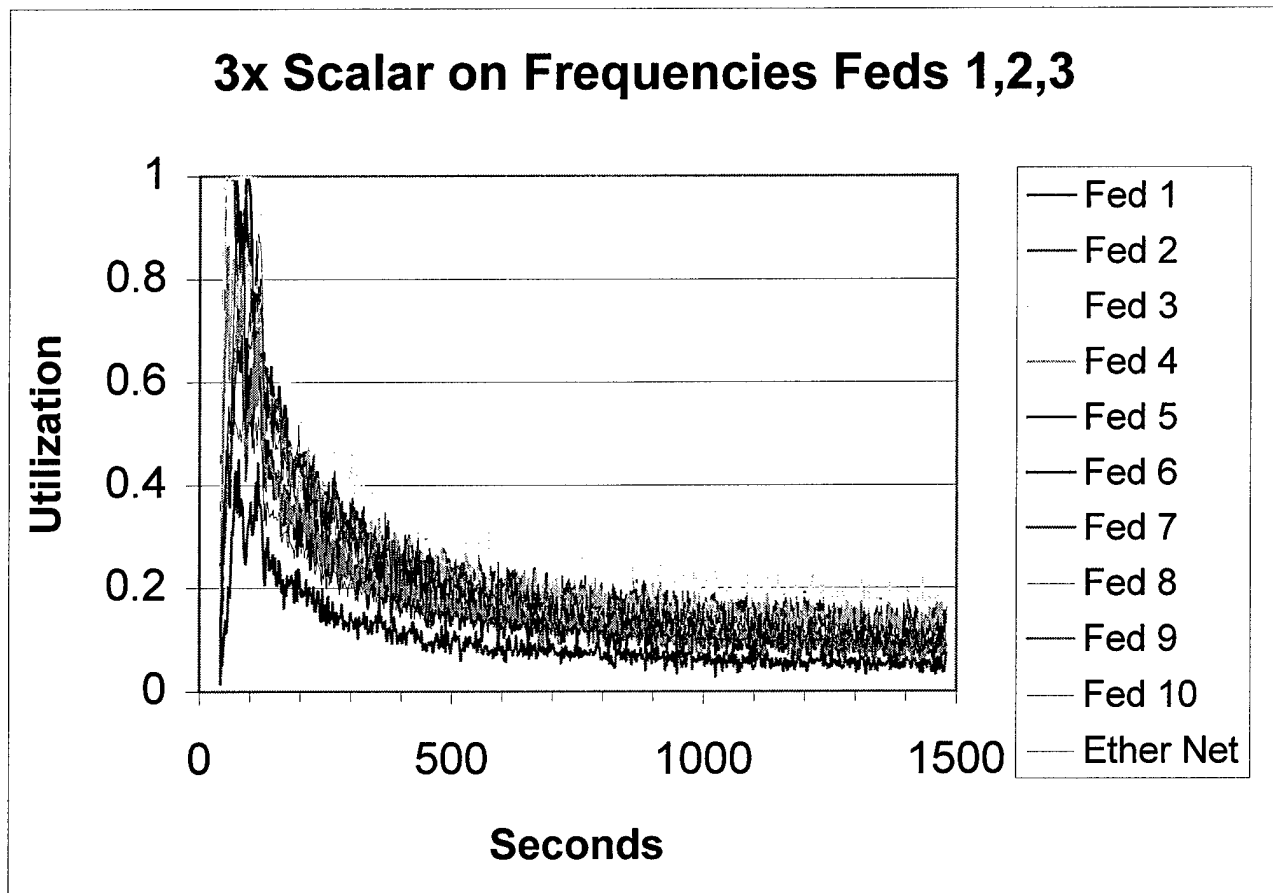


Figure 10. CPU Usage for Example Scenario

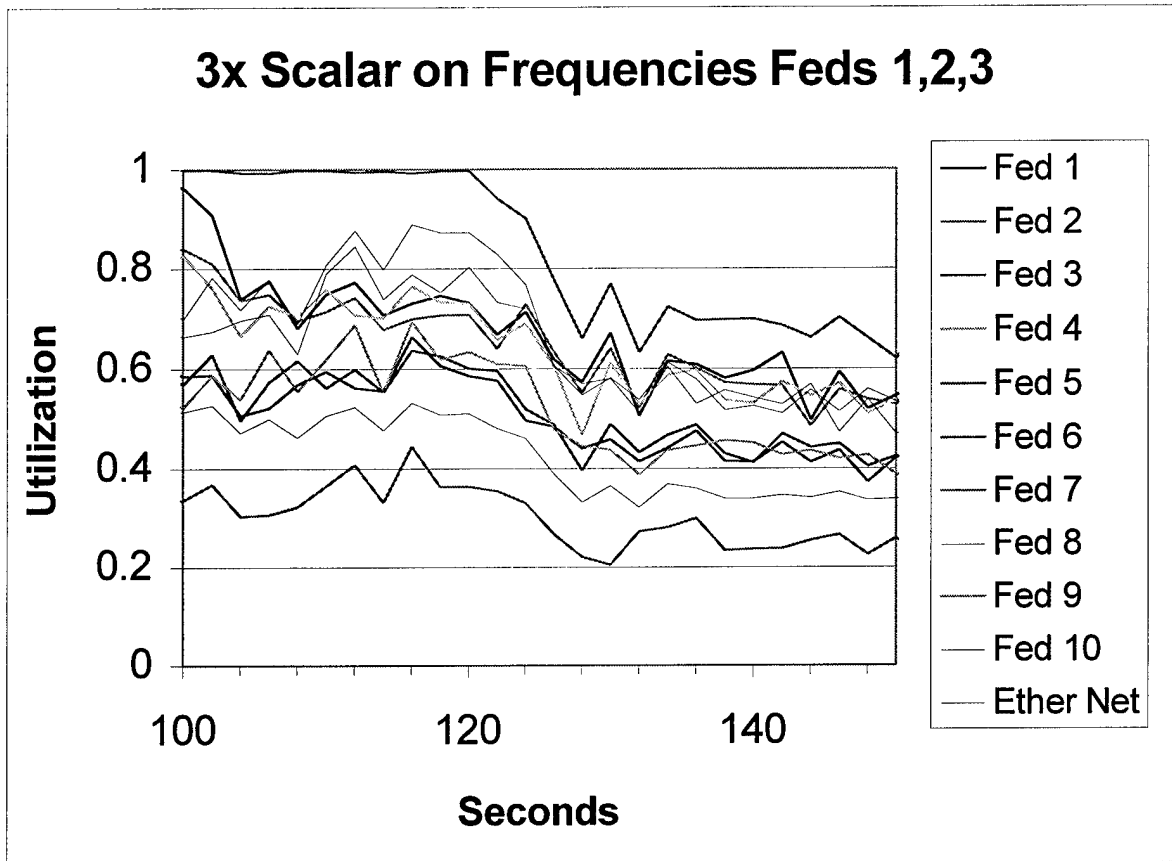


Figure 11. CPU Usage Startup Example Expanded

Table 4. Distribution of Entities Across Nodes

Federate Node	Blue Force	Opposing Force
1	932	1152
2	920	1168
3	972	1188
4	2684	0
5	2688	0
6	2692	0
7	2684	0
8	316	0
9	0	0
10	0	0
Totals	13888	13936

6. CONCLUSIONS AND FUTURE RESEARCH

This final report has documented the initial development of a constructive performance model using the HLA RTI. The data-driven model that was developed is targeted toward decision support, sensitivity analyses, and evaluating candidate system configurations that use the RTI. The project goal was accomplished. The CPM can be used to support decisions and evaluate candidate configurations that would support a specific RTI scenario.

The results of the HLA Constructive Performance Model project demonstrate that a large scale constructive simulation can be modeled in advance of the exercise date. CPM techniques are much more cost effective than a "try it and see if it runs" approach. The detailed model of the RTI that was developed demonstrates the behaviors that an exercise controller should expect to see for a specific simulation architecture. From observing the CPU utilization profile for the scenario, the implementor can decide whether or not sufficient system resources are available.

Potential sensitivity analyses using the HLA CPM could address questions such as the following:

- How many DDM regions and what types of regions are optimal for the desired scenario?
- What rationale should be used to map entities to regions?
- How many CPU nodes are needed in the network?
- Does a small change in any aspect of the simulation create a large impact on the ability of the simulation to keep up with wall clock time?
- When does "run it and see" carry too high a risk when developing a training exercise?

In the current implementation of the RTI model the object attributes are abstracted together with

the object classes. In an enhanced version of the RTI mode, it would be more desirable to represent attributes separately from their object class.

The initial performance measures focused on three RTI services: Receive Interaction, Reflect Object Instance, Time Advance Grant. The choice of these measurements was driven in part by the data that was available from the Prairie Warrior 94 training exercise. Additional performance measurements could be collected if reliable input data could be obtained to drive the model.

7. ACRONYMS

CPM	Constructive Performance Model
CPU	Central Processor Unit
DDM	Data Distribution Management
DO	Delivery Order
DMSO	Defense Modeling and Simulation Office
FED	Federation Execution Data
FOM	Federation Object Model
GVT	Global Virtual Time
HLA	High Level Architecture
IO	Input/Output
LAN	Local Area Network
LBTS	Lower Bound Time Stamp
Mbps	Megabits per second
NG	Next Generation
RID	RTI Initialization Data
RTI	Run-Time Infrastructure
SIM	Simulation
STRICOM	US Army Simulation, Training, and Instrumentation Command
TR&C	Time-Regulating and Time-Constrained
TSO	Time-Stamp Order

APPENDIX A - INDEX TO METHODS BY FILES AND INDEX TO FILES BY METHODS

METHODS BY FILES

#	File name	Method name	Appendix & pg #
1	data_distrib_mgmt_svc.c	ddm_associate_update_region	B- 190
2	data_distrib_mgmt_svc.c	ddm_create_update_region	B- 189
3	declar_mgmt_svc.c	*create_subscribed_node	B- 194
4	declar_mgmt_svc.c	add_federate_to_regions_table	B- 195
5	declar_mgmt_svc.c	dm_publish_interact_class	B- 193
6	declar_mgmt_svc.c	dm_publish_objclass	B- 192
7	declar_mgmt_svc.c	dm_subscribe_interact_class	B- 197
8	declar_mgmt_svc.c	dm_subscribe_objclass	B- 196
9	event_manager.c	eventmgr_change_processing_mode	B- 200
10	event_manager.c	eventmgr_determine_events_LBTS	B- 201
11	event_manager.c	eventmgr_get_destination	B- 204
12	event_manager.c	eventmgr_get_parent_name	B- 203
13	event_manager.c	eventmgr_process_event	B- 199
14	event_manager.c	eventmgr_retrieve_LBTS_info	B- 202
15	EventManager.c	*GetLowestTimeMessage	B- 31
16	EventManager.c	*SelectMsg	B- 32
17	EventManager.c	*SetEventMessage	B- 46
18	EventManager.c	*SetExtendEventMessage	B- 47
19	EventManager.c	AddEvent	B- 17
20	EventManager.c	AddEventToQueue	B- 19
21	EventManager.c	AddPriorityEvent	B- 14
22	EventManager.c	AnyPotentialMessagesToReceiveWithOldColorTag	B- 11
23	EventManager.c	ChangeFederateColorTag	B- 9
24	EventManager.c	CntMsg	B- 32
25	EventManager.c	ColorRTIService	B- 12
26	EventManager.c	ColorSMinRTI	B- 12
27	EventManager.c	ColorTag	B- 12
28	EventManager.c	CountRtiColors	B- 10
29	EventManager.c	CreateRTIreport	B- 51
30	EventManager.c	CurrentFederateTime	B- 9
31	EventManager.c	CurrentFederationColor	B- 9
32	EventManager.c	CurrentPhysicalTime	B- 9
33	EventManager.c	EventManager	B- 21
34	EventManager.c	GetColorTag	B- 12
35	EventManager.c	GetLBTSfromFederate	B- 9
36	EventManager.c	NextEndTime	B- 27
37	EventManager.c	PrintEventsInSystem	B- 39
38	EventManager.c	PrintEventsProcessed	B- 40
39	EventManager.c	PrintFedTime	B- 45
40	EventManager.c	PrintQueue	B- 38
41	EventManager.c	PrintQueueHistory	B- 34
42	EventManager.c	PrintUtilizationResourceTime	B- 41

#	File name	Method name	Appendix & pg #
43	EventManager.c	QueueColorCompare	B- 36
44	EventManager.c	QueueCompare	B- 36
45	EventManager.c	QueueEnd	B- 35
46	EventManager.c	QueuesInitialize	B- 33
47	EventManager.c	QueuesPrint	B- 43
48	EventManager.c	QueuesTest	B- 39
49	EventManager.c	ResourceVerification	B- 48
50	EventManager.c	rtmgr_RTleventTEST	B- 49
51	EventManager.c	SetBaseResourceTime	B- 41
52	EventManager.c	SetRtiColors	B- 10
53	EventManager.c	SimModelTEST	B- 50
54	EventManager.c	StartLBTSCalculation	B- 9
55	EventManager.c	TestForDiffColor	B- 37
56	EventManager.c	TSOBoundMessage	B- 12
57	EventManager.c	XQueuesPrint	B- 41
58	FedNodes.c	AddNodeLinksToRegionsByUnit	B- 64
59	FedNodes.c	AddRatioUnitsToNode	B- 58
60	FedNodes.c	AddRegionToNode	B- 66
61	FedNodes.c	AddSupportUnitsToNode	B- 60
62	FedNodes.c	AddToNode	B- 62
63	FedNodes.c	CreateNodes	B- 55
64	FedNodes.c	GetUnitsByCoTypeForRegion	B- 54
65	FedNodes.c	PrintNodesOfFed	B- 68
66	FedNodes.c	PrintUnitsOfFed	B- 69
67	FedNodes.c	PublishByFederate	B- 72
68	FedNodes.c	RemoveNodeLinksToRegionsByUnit	B- 63
69	FedNodes.c	struct	B- 67
70	FedNodes.c	SubscribeByFederate	B- 70
71	fedrtn_mgmt_svc.c	add_federate_to_feds_saving_list	B- 212
72	fedrtn_mgmt_svc.c	fm_create_fedrtn_execution	B- 206
73	fedrtn_mgmt_svc.c	fm_federate_save_achieved	B- 214
74	fedrtn_mgmt_svc.c	fm_federate_save_begun	B- 213
75	fedrtn_mgmt_svc.c	fm_fedrtn_save_achieved	B- 217
76	fedrtn_mgmt_svc.c	fm_initialize_federate	B- 207
77	fedrtn_mgmt_svc.c	fm_initiate_federate_save	B- 210
78	fedrtn_mgmt_svc.c	fm_is_fedrtn_saved	B- 216
79	fedrtn_mgmt_svc.c	fm_join_fedrtn_execution	B- 208
80	fedrtn_mgmt_svc.c	fm_request_fedrtn_save	B- 209
81	fedrtn_mgmt_svc.c	fm_setup_fedrtn_complete_events	B- 218
82	fedrtn_mgmt_svc.c	fm_setup_initiate_federate_save_events	B- 211
83	fedrtn_mgmt_svc.c	is_federate_in_fed_saved_list	B- 215
84	FilterUnits.c	*FilterByEcheleon	B- 77
85	FilterUnits.c	*FilterNotAssignedToFed	B- 79
86	FilterUnits.c	*FilterNotInRegion	B- 80
87	FilterUnits.c	AddToFilterSubordinates	B- 75
88	FilterUnits.c	EquipInList	B- 76
89	FilterUnits.c	MergeFilterList	B- 81
90	FilterUnits.c	PrintFilterList	B- 82
91	GrowArmy.c	*Grow_Unit_Characteristics	B- 85
92	GrowArmy.c	*Grow_Unit_List	B- 87

#	File name	Method name	Appendix & pg #
93	GrowArmy.c	Grow_Echeleon	B- 88
94	GrowArmy.c	GrowInitArmy	B- 90
95	GrowArmy.c	imax	B- 84
96	GrowArmy.c	imin	B- 84
97	GrowArmy.c	PickSome	B- 84
98	GrowArmy.c	triangle	B- 97
99	init_rti.c	Initialize_RTI	B- 221
100	init_rti.c	RIDdb_init	B- 220
101	InterestGroups.c	GrowInterestGroup	B- 99
102	InterestGroups.c	Print_InterestGroup	B- 100
103	io_manager.c	iomgr_determine_iochannel	B- 223
104	io_manager.c	iomgr_send_ioevent	B- 224
105	mood.c	draw_graphics	B- 232
106	mood.c	draw_text	B- 230
107	mood.c	get_GC	B- 228
108	mood.c	LalaClear	B- 236
109	mood.c	LalaColor	B- 237
110	mood.c	LalaDraw	B- 238
111	mood.c	LalaDrawLink	B- 239
112	mood.c	LalaFinished	B- 240
113	mood.c	LalaInit	B- 234
114	mood.c	LalaPlace	B- 237
115	mood.c	LalaTimeQueue	B- 239
116	mood.c	LalaUpdate	B- 237
117	mood.c	load_font	B- 229
118	mood.c	TooSmall	B- 233
119	object_mgmt_svc.c	*create_obj_instance	B- 243
120	object_mgmt_svc.c	*create_regions_node	B- 242
121	object_mgmt_svc.c	*om_create_destinations_element	B- 253
122	object_mgmt_svc.c	add_obj_to_federates_table	B- 244
123	object_mgmt_svc.c	om_discover_object	B- 254
124	object_mgmt_svc.c	om_instance_exists	B- 246
125	object_mgmt_svc.c	om_is_class_published	B- 245
126	object_mgmt_svc.c	om_is_object_registered	B- 249
127	object_mgmt_svc.c	om_provide_attrib_value_update	B- 263
128	object_mgmt_svc.c	om_receive_interaction	B- 260
129	object_mgmt_svc.c	om_reflect_attrib_values	B- 257
130	object_mgmt_svc.c	om_register_instance	B- 247
131	object_mgmt_svc.c	om_request_attrib_value_update	B- 252
132	object_mgmt_svc.c	om_send_interaction	B- 251
133	object_mgmt_svc.c	om_setup_discover_events	B- 255
134	object_mgmt_svc.c	om_setup_receive_interaction_events	B- 261
135	object_mgmt_svc.c	om_setup_reflect_events	B- 258
136	object_mgmt_svc.c	om_update_attrib_values	B- 250
137	ReadOrdr.c	*DuplicateOrder	B- 104
138	ReadOrdr.c	*MakeOrder	B- 103
139	ReadOrdr.c	Free_OrderQue	B- 109
140	ReadOrdr.c	MsgIdTag	B- 105
141	ReadOrdr.c	Print_Lotl	B- 105
142	ReadOrdr.c	Print_NewOrder	B- 110

#	File name	Method name	Appendix & pg #
143	ReadOrdr.c	Print_Order	B- 107
144	ReadOrdr.c	Print_OrderQue	B- 108
145	ReadOrdr.c	Print_Route	B- 106
146	ReadOrdr.c	ReadOrder	B- 102
147	ReadOrdr.c	SetMsgDest	B- 105
148	ReadOrdr.c	SetMsgOrig	B- 105
149	Regions.c	*CmdUnitNotInRegion	B- 124
150	Regions.c	*FillRegion	B- 125
151	Regions.c	*FindRegion	B- 127
152	Regions.c	*PutAllInRegBySideInFilterList	B- 129
153	Regions.c	*PutColInRegionInFilterList	B- 128
154	Regions.c	*PutNumInRegBySideInFilterList	B- 130
155	Regions.c	*PutRegionInFilterList	B- 131
156	Regions.c	AddCommandRegions	B- 117
157	Regions.c	AddNewRegion	B- 119
158	Regions.c	AddRegionReference	B- 121
159	Regions.c	AddToRegion	B- 122
160	Regions.c	AddToRegionElements	B- 123
161	Regions.c	CreateRegions	B- 114
162	Regions.c	PrintOneRegion	B- 137
163	Regions.c	PrintRegionElements	B- 134
164	Regions.c	PrintRegions	B- 136
165	Regions.c	PrintRegionsNodes	B- 135
166	Regions.c	RegisterRegions	B- 138
167	Regions.c	RemoveRegionReference	B- 132
168	rti_manager.c	rtimgr_clear	B- 300
169	rti_manager.c	rtimgr_clear_reduction_network_info	B- 273
170	rti_manager.c	rtimgr_compute_elapsed_time_statistic	B- 281
171	rti_manager.c	rtimgr_criteria_compare	B- 294
172	rti_manager.c	rtimgr_criteria_create	B- 294
173	rti_manager.c	rtimgr_federate_is_parent	B- 278
174	rti_manager.c	rtimgr_federate_processed_initial_counts	B- 277
175	rti_manager.c	rtimgr_FedExdb_init	B- 274
176	rti_manager.c	rtimgr_final_cleanup	B- 301
177	rti_manager.c	rtimgr_FOMdb_init	B- 271
178	rti_manager.c	rtimgr_get_Fed_ambsvc_time	B- 299
179	rti_manager.c	rtimgr_get_RTI_ambsvc_time	B- 295
180	rti_manager.c	rtimgr_init	B- 276
181	rti_manager.c	rtimgr_is_fedamb_svc	B- 290
182	rti_manager.c	rtimgr_printsvc_stat	B- 280
183	rti_manager.c	rtimgr_process_fedamb_svc	B- 287
184	rti_manager.c	rtimgr_process_rtiamb_svc	B- 282
185	rti_manager.c	rtimgr_retrieve_svctblinfo	B- 293
186	rti_manager.c	rtimgr_RTlevent	B- 291
187	rti_manager.c	rtimgr_update_fedrtn_state_status	B- 279
188	rti_utils.c	change_strgpeices_to_onestrng	B- 266
189	rti_utils.c	get_string_peices	B- 265
190	Sim.c	main	B- 154
191	SimModel.c	CreateInteraction	B- 165
192	SimModel.c	InitSimModel	B- 166

#	File name	Method name	Appendix & pg #
193	SimModel.c	InitSimModelTest	B- 170
194	SimModel.c	SimModel	B- 159
195	SimModel.c	UpdateEntity	B- 164
196	SOAcreat.c	*c_Comm_Net_Association	B- 141
197	SOAcreat.c	*c_Comm_Net_List	B- 141
198	SOAcreat.c	*c_Duplicate_Event_Message	B- 149
199	SOAcreat.c	*c_Event_Message	B- 148
200	SOAcreat.c	*c_Federate_Destination	B- 148
201	SOAcreat.c	*c_Filter_Unit_List	B- 148
202	SOAcreat.c	*c_InterestList	B- 143
203	SOAcreat.c	*c_Nodes_of_Fed_List	B- 146
204	SOAcreat.c	*c_Nodes_wrt_Region_List	B- 146
205	SOAcreat.c	*c_Order_Packet	B- 143
206	SOAcreat.c	*c_Region_Element_List	B- 147
207	SOAcreat.c	*c_Region_List	B- 147
208	SOAcreat.c	*c_Region_Node_Handle	B- 145
209	SOAcreat.c	*c_Task_List	B- 144
210	SOAcreat.c	*c_Unit_Region_List	B- 148
211	SOAcreat.c	*c_Units_on_Node_List	B- 145
212	SOAcreat.c	*c_Truth_Group_List	B- 141
213	SOAcreat.c	*c_Node_Table_Def	B- 142
214	SOAcreat.c	*c_Node_List	B- 142
215	SOAcreat.c	*c_Unit_List	B- 143
216	SOAcreat.c	*c_Serv_Characteristics	B- 144
217	SOAcreat.c	*c_Serv_List	B- 144
218	SOAcreat.c	*c_Serv_Stack	B- 145
219	SOAcreat.c	*c_Region_Definition	B- 145
220	SOAdestr.c	d_Comm_Net_Association	B- 151
221	SOAdestr.c	d_Comm_Net_List	B- 151
222	SOAdestr.c	d_Event_Message	B- 151
223	SOAdestr.c	d_Federate_Destination	B- 152
224	SOAdestr.c	d_Filter_Unit_List	B- 151
225	SOAdestr.c	d_Node_List	B- 151
226	SOAdestr.c	d_Node_Table_Def	B- 151
227	SOAdestr.c	d_Order_Packet	B- 151
228	SOAdestr.c	d_Region_Element_List	B- 152
229	SOAdestr.c	d_Serv_Characteristics	B- 152
230	SOAdestr.c	d_Serv_List	B- 152
231	SOAdestr.c	d_Serv_Stack	B- 152
232	SOAdestr.c	d_Task_List	B- 152
233	SOAdestr.c	d_Truth_Group_List	B- 151
234	SOAdestr.c	d_Unit_Characteristics	B- 151
235	SOAdestr.c	d_Unit_List	B- 151
236	SOAdestr.c	d_Unit_Region_List	B- 152
237	stats_manager.c	initialize_statistic	B- 305
238	stats_manager.c	statsmgr_accum_totals	B- 317
239	stats_manager.c	statsmgr_cleanup	B- 322
240	stats_manager.c	statsmgr_clear_accum_totals	B- 321
241	stats_manager.c	statsmgr_collect_statistic	B- 319
242	stats_manager.c	statsmgr_compute_nbr_nodes	B- 313

30 June 1999

#	File name	Method name	Appendix & pg #
243	stats_manager.c	statsmgr_forward_to_sim_model	B- 314
244	stats_manager.c	statsmgr_get_statsarray_index	B- 316
245	stats_manager.c	statsmgr_init_statruns_file	B- 304
246	stats_manager.c	statsmgr_lookup_in_fedexdb	B- 308
247	stats_manager.c	statsmgr_lookup_in_fomdb	B- 307
248	stats_manager.c	statsmgr_norm_distrib	B- 315
249	stats_manager.c	statsmgr_print_accum_totals	B- 320
250	stats_manager.c	statsmgr_setup_class_in_fedexdb	B- 311
251	stats_manager.c	statsmgr_setup_fed_in_fedexdb	B- 309
252	stats_manager.c	statsmgr_setup_instance_in_fedexdb	B- 312
253	stats_manager.c	statsmgr_setup_region_in_fedexdb	B- 310
254	stats_manager.c	statsmgr_setup_stats_tables	B- 306
255	time_mgmt_svc.c	all_my_subfederates_reported	B- 327
256	time_mgmt_svc.c	tm_clear_LBTS_info	B- 337
257	time_mgmt_svc.c	tm_controller_LBTS_compute	B- 328
258	time_mgmt_svc.c	tm_forward_LBTS_info	B- 332
259	time_mgmt_svc.c	tm_LBTS_requests_setup	B- 324
260	time_mgmt_svc.c	tm_query_fed_LBTS	B- 334
261	time_mgmt_svc.c	tm_time_adv_grant	B- 336
262	time_mgmt_svc.c	tm_time_adv_grant_setup	B- 333
263	time_mgmt_svc.c	tm_time_adv_request	B- 325
264	UnitCharFile.c	CountSubrEquip	B- 178
265	UnitCharFile.c	GetTotalEquip	B- 179
266	UnitCharFile.c	GetTotalEquipByLevel	B- 179
267	UnitCharFile.c	GetTotalPersonByLevel	B- 179
268	UnitCharFile.c	Initialize_Others	B- 173
269	UnitCharFile.c	MaxEcheleon	B- 173
270	UnitCharFile.c	Print_Echeleon	B- 181
271	UnitCharFile.c	Print_EchSummary	B- 180
272	UnitCharFile.c	Print_UnitC	B- 175
273	UnitCharFile.c	Print_UnitC_comma	B- 176
274	UnitCharFile.c	Print_UnitC_File	B- 175
275	UnitCharFile.c	PrintRTIEchelon	B- 182
276	UnitCharFile.c	PrintRTIInstanceEchelon	B- 183
277	UnitCharFile.c	ResetEcheleon	B- 178
278	UnitCharFile.c	TallyClearEch	B- 179
279	UnitCharFile.c	TallyEcheleon	B- 178
280	UnitCharFile.c	TallyPrintEch	B- 179
281	UnitCharFile.c	UnitCharacter	B- 173
282	UnitCharFile.c	ViewEcheleonLeft	B- 185
283	UnitCharFile.c	ViewEcheleonRight	B- 185
284	UnitCharFile.c	ViewNew	B- 185
285	UnitCharFile.c	ViewNext	B- 185
286	UnitCharFile.c	ViewRefresh	B- 186

FILES BY METHODS

#	Method name	Filename	Appendix & pg #
1	*c_Comm_Net_Association	SOAcreat.c	B- 141
2	*c_Comm_Net_List	SOAcreat.c	B- 141
3	*c_Duplicate_Event_Message	SOAcreat.c	B- 149
4	*c_Event_Message	SOAcreat.c	B- 148
5	*c_Federate_Destination	SOAcreat.c	B- 148
6	*c_Filter_Unit_List	SOAcreat.c	B- 148
7	*c_InterestList	SOAcreat.c	B- 143
8	*c_Node_List	SOAcreat.c	B- 142
9	*c_Node_Table_Def	SOAcreat.c	B- 142
10	*c_Nodes_of_Fed_List	SOAcreat.c	B- 146
11	*c_Nodes_wrt_Region_List	SOAcreat.c	B- 146
12	*c_Order_Packet	SOAcreat.c	B- 143
13	*c_Region_Definition	SOAcreat.c	B- 145
14	*c_Region_Element_List	SOAcreat.c	B- 147
15	*c_Region_List	SOAcreat.c	B- 147
16	*c_Region_Node_Handle	SOAcreat.c	B- 145
17	*c_Serv_Characteristics	SOAcreat.c	B- 144
18	*c_Serv_List	SOAcreat.c	B- 144
19	*c_Serv_Stack	SOAcreat.c	B- 145
20	*c_Task_List	SOAcreat.c	B- 144
21	*c_Truth_Group_List	SOAcreat.c	B- 141
22	*c_Unit_List	SOAcreat.c	B- 143
23	*c_Unit_Region_List	SOAcreat.c	B- 148
24	*c_Units_on_Node_List	SOAcreat.c	B- 145
25	*CmdUnitNotInRegion	Regions.c	B- 124
26	*create_obj_instance	object_mgmt_svc.c	B- 243
27	*create_regions_node	object_mgmt_svc.c	B- 242
28	*create_subscribed_node	declar_mgmt_svc.c	B- 194
29	*DuplicateOrder	ReadOrdr.c	B- 104
30	*FillRegion	Regions.c	B- 125
31	*FilterByEcheleon	FilterUnits.c	B- 77
32	*FilterNotAssignedToFed	FilterUnits.c	B- 79
33	*FilterNotInRegion	FilterUnits.c	B- 80
34	*FindRegion	Regions.c	B- 127
35	*GetLowestTimeMessage	EventManager.c	B- 31

#	Method name	Filename	Appendix & pg #
36	*Grow_Unit_Characteristics	GrowArmy.c	B- 85
37	*Grow_Unit_List	GrowArmy.c	B- 87
38	*MakeOrder	ReadOrdr.c	B- 103
39	*om_create_destinations_element	object_mgmt_svc.c	B- 253
40	*PutAllInRegBySideInFilterList	Regions.c	B- 129
41	*PutColInRegionInFilterList	Regions.c	B- 128
42	*PutNumInRegBySideInFilterList	Regions.c	B- 130
43	*PutRegionInFilterList	Regions.c	B- 131
44	*SelectMsg	EventManager.c	B- 32
45	*SetEventMessage	EventManager.c	B- 46
46	*SetExtendEventMessage	EventManager.c	B- 47
47	add_federate_to_feds_saving_list	fedrtn_mgmt_svc.c	B- 212
48	add_federate_to_regions_table	declar_mgmt_svc.c	B- 195
49	add_obj_to_federates_table	object_mgmt_svc.c	B- 244
50	AddCommandRegions	Regions.c	B- 117
51	AddEvent	EventManager.c	B- 17
52	AddEventToQueue	EventManager.c	B- 19
53	AddNewRegion	Regions.c	B- 119
54	AddNodeLinksToRegionsByUnit	FedNodes.c	B- 64
55	AddPriorityEvent	EventManager.c	B- 14
56	AddRatioUnitsToNode	FedNodes.c	B- 58
57	AddRegionReference	Regions.c	B- 121
58	AddRegionToNode	FedNodes.c	B- 66
59	AddSupportUnitsToNode	FedNodes.c	B- 60
60	AddToFilterSubordinates	FilterUnits.c	B- 75
61	AddToNode	FedNodes.c	B- 62
62	AddToRegion	Regions.c	B- 122
63	AddToRegionElements	Regions.c	B- 123
64	all_my_subfederates_reported	time_mgmt_svc.c	B- 327
65	AnyPotentialMessagesToReceiveWithOldColorTag	EventManager.c	B- 11
66	change_strgpeices_to_onestrng	rti_utils.c	B- 266
67	ChangeFederateColorTag	EventManager.c	B- 9
68	CntMsg	EventManager.c	B- 32
69	ColorRTIService	EventManager.c	B- 12
70	ColorSISMinRTI	EventManager.c	B- 12

#	Method name	Filename	Appendix & pg #
71	ColorTag	EventManager.c	B- 12
72	CountRtiColors	EventManager.c	B- 10
73	CountSubrEquip	UnitCharFile.c	B- 178
74	CreateInteraction	SimModel.c	B- 165
75	CreateNodes	FedNodes.c	B- 55
76	CreateRegions	Regions.c	B- 114
77	CreateRTIreport	EventManager.c	B- 51
78	CurrentFederateTime	EventManager.c	B- 9
79	CurrentFederationColor	EventManager.c	B- 9
80	CurrentPhysicalTime	EventManager.c	B- 9
81	d_Comm_Net_Association	SOAdestr.c	B- 151
82	d_Comm_Net_List	SOAdestr.c	B- 151
83	d_Event_Message	SOAdestr.c	B- 151
84	d_Federate_Destination	SOAdestr.c	B- 152
85	d_Filter_Unit_List	SOAdestr.c	B- 151
86	d_Node_List	SOAdestr.c	B- 151
87	d_Node_Table_Def	SOAdestr.c	B- 151
88	d_Order_Packet	SOAdestr.c	B- 151
89	d_Region_Element_List	SOAdestr.c	B- 152
90	d_Serv_Characteristics	SOAdestr.c	B- 152
91	d_Serv_List	SOAdestr.c	B- 152
92	d_Serv_Stack	SOAdestr.c	B- 152
93	d_Task_List	SOAdestr.c	B- 152
94	d_Truth_Group_List	SOAdestr.c	B- 151
95	d_Unit_Characteristics	SOAdestr.c	B- 151
96	d_Unit_List	SOAdestr.c	B- 151
97	d_Unit_Region_List	SOAdestr.c	B- 152
98	ddm_associate_update_region	data_distrib_mgmt_svc.c	B- 190
99	ddm_create_update_region	data_distrib_mgmt_svc.c	B- 189
100	dm_publish_interact_class	declar_mgmt_svc.c	B- 193
101	dm_publish_objclass	declar_mgmt_svc.c	B- 192
102	dm_subscribe_interact_class	declar_mgmt_svc.c	B- 197
103	dm_subscribe_objclass	declar_mgmt_svc.c	B- 196
104	draw_graphics	mood.c	B- 232
105	draw_text	mood.c	B- 230
106	EquipInList	FilterUnits.c	B- 76
107	EventManager	EventManager.c	B- 21
108	eventmgr_change_processing_mode	event_manager.c	B- 200
109	eventmgr_determine_events_LBTS	event_manager.c	B- 201
110	eventmgr_get_destination	event_manager.c	B- 204
111	eventmgr_get_parent_name	event_manager.c	B- 203

#	Method name	Filename	Appendix & pg #
112	eventmgr_process_event	event_manager.c	B- 199
113	eventmgr_retrieve_LBTS_info	event_manager.c	B- 202
114	fm_create_fedrtn_execution	fedrtn_mgmt_svc.c	B- 206
115	fm_federate_save_achieved	fedrtn_mgmt_svc.c	B- 214
116	fm_federate_save_begun	fedrtn_mgmt_svc.c	B- 213
117	fm_fedrtn_save_achieved	fedrtn_mgmt_svc.c	B- 217
118	fm_initialize_federate	fedrtn_mgmt_svc.c	B- 207
119	fm_initiate_federate_save	fedrtn_mgmt_svc.c	B- 210
120	fm_is_fedrtn_saved	fedrtn_mgmt_svc.c	B- 216
121	fm_join_fedrtn_execution	fedrtn_mgmt_svc.c	B- 208
122	fm_request_fedrtn_save	fedrtn_mgmt_svc.c	B- 209
123	fm_setup_fedrtn_complete_events	fedrtn_mgmt_svc.c	B- 218
124	fm_setup_initiate_federate_save_events	fedrtn_mgmt_svc.c	B- 211
125	Free_OrderQue	ReadOrdr.c	B- 109
126	get_GC	mood.c	B- 228
127	get_string_peices	rti_utils.c	B- 265
128	GetColorTag	EventManager.c	B- 12
129	GetLBTSfromFederate	EventManager.c	B- 9
130	GetTotalEquip	UnitCharFile.c	B- 179
131	GetTotalEquipByLevel	UnitCharFile.c	B- 179
132	GetTotalPersonByLevel	UnitCharFile.c	B- 179
133	GetUnitsByCoTypeForRegion	FedNodes.c	B- 54
134	Grow_Echeleon	GrowArmy.c	B- 88
135	GrowInitArmy	GrowArmy.c	B- 90
136	GrowInterestGroup	InterestGroups.c	B- 99
137	imax	GrowArmy.c	B- 84
138	imin	GrowArmy.c	B- 84
139	Initialize_Others	UnitCharFile.c	B- 173
140	Initialize_RTI	init_rti.c	B- 221
141	initialize_statistic	stats_manager.c	B- 305
142	InitSimModel	SimModel.c	B- 166
143	InitSimModelTest	SimModel.c	B- 170
144	iomgr_determine_iochannel	io_manager.c	B- 223
145	iomgr_send_ioevent	io_manager.c	B- 224
146	is_federate_in_fed_saved_list	fedrtn_mgmt_svc.c	B- 215
147	LalaClear	mood.c	B- 236
148	LalaColor	mood.c	B- 237
149	LalaDraw	mood.c	B- 238
150	LalaDrawLink	mood.c	B- 239

#	Method name	Filename	Appendix & pg #
151	LalaFinished	mood.c	B- 240
152	LalaInit	mood.c	B- 234
153	LalaPlace	mood.c	B- 237
154	LalaTimeQueue	mood.c	B- 239
155	LalaUpdate	mood.c	B- 237
156	load_font	mood.c	B- 229
157	main	Sim.c	B- 154
158	MaxEcheleon	UnitCharFile.c	B- 173
159	MergeFilterList	FilterUnits.c	B- 81
160	MsgIdTag	ReadOrdr.c	B- 105
161	NextEndTime	EventManager.c	B- 27
162	om_discover_object	object_mgmt_svc.c	B- 254
163	om_instance_exists	object_mgmt_svc.c	B- 246
164	om_is_class_published	object_mgmt_svc.c	B- 245
165	om_is_object_registered	object_mgmt_svc.c	B- 249
166	om_provide_attrib_value_update	object_mgmt_svc.c	B- 263
167	om_receive_interaction	object_mgmt_svc.c	B- 260
168	om_reflect_attrib_values	object_mgmt_svc.c	B- 257
169	om_register_instance	object_mgmt_svc.c	B- 247
170	om_request_attrib_value_update	object_mgmt_svc.c	B- 252
171	om_send_interaction	object_mgmt_svc.c	B- 251
172	om_setup_discover_events	object_mgmt_svc.c	B- 255
173	om_setup_receive_interaction_events	object_mgmt_svc.c	B- 261
174	om_setup_reflect_events	object_mgmt_svc.c	B- 258
175	om_update_attrib_values	object_mgmt_svc.c	B- 250
176	PickSome	GrowArmy.c	B- 84
177	Print_Echeleon	UnitCharFile.c	B- 181
178	Print_EchSummary	UnitCharFile.c	B- 180
179	Print_InterestGroup	InterestGroups.c	B- 100
180	Print_Lotl	ReadOrdr.c	B- 105
181	Print_NewOrder	ReadOrdr.c	B- 110
182	Print_Order	ReadOrdr.c	B- 107
183	Print_OrderQue	ReadOrdr.c	B- 108
184	Print_Route	ReadOrdr.c	B- 106
185	Print_UnitC	UnitCharFile.c	B- 175
186	Print_UnitC_comma	UnitCharFile.c	B- 176
187	Print_UnitC_File	UnitCharFile.c	B- 175
188	PrintEventsInSystem	EventManager.c	B- 39
189	PrintEventsProcessed	EventManager.c	B- 40
190	PrintFedTime	EventManager.c	B- 45
191	PrintFilterList	FilterUnits.c	B- 82
192	PrintNodesOfFed	FedNodes.c	B- 68
193	PrintOneRegion	Regions.c	B- 137
194	PrintQueue	EventManager.c	B- 38

#	Method name	Filename	Appendix & pg #
195	PrintQueueHistory	EventManager.c	B- 34
196	PrintRegionElements	Regions.c	B- 134
197	PrintRegions	Regions.c	B- 136
198	PrintRegionsNodes	Regions.c	B- 135
199	PrintRTIEchelon	UnitCharFile.c	B- 182
200	PrintRTIInstanceEchelon	UnitCharFile.c	B- 183
201	PrintUnitsOfFed	FedNodes.c	B- 69
202	PrintUtilizationResourceTime	EventManager.c	B- 41
203	PublishByFederate	FedNodes.c	B- 72
204	QueueColorCompare	EventManager.c	B- 36
205	QueueCompare	EventManager.c	B- 36
206	QueueEnd	EventManager.c	B- 35
207	QueuesInitialize	EventManager.c	B- 33
208	QueuesPrint	EventManager.c	B- 43
209	QueuesTest	EventManager.c	B- 39
210	ReadOrder	ReadOrd.c	B- 102
211	RegisterRegions	Regions.c	B- 138
212	RemoveNodeLinksToRegionsByUnit	FedNodes.c	B- 63
213	RemoveRegionReference	Regions.c	B- 132
214	ResetEchelon	UnitCharFile.c	B- 178
215	ResourceVerification	EventManager.c	B- 48
216	RIDdb_init	init_rti.c	B- 220
217	rtmgr_clear	rti_manager.c	B- 300
218	rtmgr_clear_reduction_network_info	rti_manager.c	B- 273
219	rtmgr_compute_elapsed_time_statistic	rti_manager.c	B- 281
220	rtmgr_criteria_compare	rti_manager.c	B- 294
221	rtmgr_criteria_create	rti_manager.c	B- 294
222	rtmgr_federate_is_parent	rti_manager.c	B- 278
223	rtmgr_federate_processed_initial_counts	rti_manager.c	B- 277
224	rtmgr_FedExdb_init	rti_manager.c	B- 274
225	rtmgr_final_cleanup	rti_manager.c	B- 301
226	rtmgr_FOMdb_init	rti_manager.c	B- 271
227	rtmgr_get_Fed_ambsvc_time	rti_manager.c	B- 299
228	rtmgr_get_RTI_ambsvc_time	rti_manager.c	B- 295
229	rtmgr_init	rti_manager.c	B- 276
230	rtmgr_is_fedamb_svc	rti_manager.c	B- 290
231	rtmgr_printsvc_stat	rti_manager.c	B- 280
232	rtmgr_process_fedamb_svc	rti_manager.c	B- 287
233	rtmgr_process_rtiamb_svc	rti_manager.c	B- 282

#	Method name	Filename	Appendix & pg #
234	rtimgr_retrieve_svctblinfo	rti_manager.c	B- 293
235	rtimgr_RTlevent	rti_manager.c	B- 291
236	rtimgr_RTleventTEST	EventManager.c	B- 49
237	rtimgr_update_fedrtn_state_status	rti_manager.c	B- 279
238	SetBaseResourceTime	EventManager.c	B- 41
239	SetMsgDest	ReadOrdr.c	B- 105
240	SetMsgOrig	ReadOrdr.c	B- 105
241	SetRtiColors	EventManager.c	B- 10
242	SimModel	SimModel.c	B- 159
243	SimModelTEST	EventManager.c	B- 50
244	StartLBTSCalculation	EventManager.c	B- 9
245	statsmgr_accum_totals	stats_manager.c	B- 317
246	statsmgr_cleanup	stats_manager.c	B- 322
247	statsmgr_clear_accum_totals	stats_manager.c	B- 321
248	statsmgr_collect_statistic	stats_manager.c	B- 319
249	statsmgr_compute_nbr_nodes	stats_manager.c	B- 313
250	statsmgr_forward_to_sim_model	stats_manager.c	B- 314
251	statsmgr_get_statsarray_index	stats_manager.c	B- 316
252	statsmgr_init_statruns_file	stats_manager.c	B- 304
253	statsmgr_lookup_in_fedexdb	stats_manager.c	B- 308
254	statsmgr_lookup_in_fomdb	stats_manager.c	B- 307
255	statsmgr_norm_distrib	stats_manager.c	B- 315
256	statsmgr_print_accum_totals	stats_manager.c	B- 320
257	statsmgr_setup_class_in_fedexdb	stats_manager.c	B- 311
258	statsmgr_setup_fed_in_fedexdb	stats_manager.c	B- 309
259	statsmgr_setup_instance_in_fedexdb	stats_manager.c	B- 312
260	statsmgr_setup_region_in_fedexdb	stats_manager.c	B- 310
261	statsmgr_setup_stats_tables	stats_manager.c	B- 306
262	struct	FedNodes.c	B- 67
263	SubscribeByFederate	FedNodes.c	B- 70
264	TallyClearEch	UnitCharFile.c	B- 179
265	TallyEcheleon	UnitCharFile.c	B- 178
266	TallyPrintEch	UnitCharFile.c	B- 179
267	TestForDiffColor	EventManager.c	B- 37
268	tm_clear_LBTS_info	time_mgmt_svc.c	B- 337

#	Method name	Filename	Appendix & pg #
269	tm_controller_LBTS_compute	time_mgmt_svc.c	B- 328
270	tm_forward_LBTS_info	time_mgmt_svc.c	B- 332
271	tm_LBTS_requests_setup	time_mgmt_svc.c	B- 324
272	tm_query_fed_LBTS	time_mgmt_svc.c	B- 334
273	tm_time_adv_grant	time_mgmt_svc.c	B- 336
274	tm_time_adv_grant_setup	time_mgmt_svc.c	B- 333
275	tm_time_adv_request	time_mgmt_svc.c	B- 325
276	TooSmall	mood.c	B- 233
277	triangle	GrowArmy.c	B- 97
278	TSOBoundMessage	EventManager.c	B- 12
279	UnitCharacter	UnitCharFile.c	B- 173
280	UpdateEntity	SimModel.c	B- 164
281	ViewEcheleonLeft	UnitCharFile.c	B- 185
282	ViewEcheleonRight	UnitCharFile.c	B- 185
283	ViewNew	UnitCharFile.c	B- 185
284	ViewNext	UnitCharFile.c	B- 185
285	ViewRefresh	UnitCharFile.c	B- 186
286	XQueuesPrint	EventManager.c	B- 41

APPENDIX B - CPM CODE

/* File: EventManager.c */	7
extern double CurrentPhysicalTime() {	9
extern double CurrentFederateTime(int Fed) {	9
extern unsigned int StartLBTSCalculation()	9
extern double GetLBTsfromFederate(int FedIdPlus) {	9
extern unsigned int CurrentFederationColor()	9
extern void ChangeFederateColorTag(int Federate, int *Sent, int *Received)	9
extern int CountRtiColors(int Fed, int ColorTag)	10
extern int SetRtiColors(int Fed, int ColorTag)	10
extern int AnyPotentialMessagesToReceiveWithOldColorTag()	12
extern void ColorTag(int Federate, struct Event_Message *Add)	12
extern unsigned int GetColorTag(int Federate)	12
extern int TSOBoundMessage(struct Event_Message *Msg)	13
extern void AddPriorityEvent(FILE *out, char *Type,	14
extern void AddEvent(FILE *out, char *Type,	17
extern void AddEventToQueue(FILE *out, struct Event_Message **Top,	19
extern double EventManager(FILE *out, FILE *LgFile, int *AreQueuesEmpty,	21
extern int ColorRTIService(struct Event_Message *MsgPtr, int *ColorSel, int *LineOffset)	25
extern int ColorSIMinRTI(struct Event_Message *MsgPtr, int *ColorSel, int *LineOffset)	26
extern double NextEndTime(int *Federate, int *Que)	27
extern struct Event_Message *GetLowestTimeMessage(int Fed, int Que)	31
extern int CntMsg(struct Event_Message *Top)	32
extern struct Event_Message *SelectMsg(struct Event_Message **Top) /* */	32
extern void QueuesInitialize()	33
extern void PrintQueueHistory(FILE *out)	34
extern double QueueEnd(struct Event_Message *Pptr)	35
extern int QueueCompare(struct Event_Message *Testptr, struct Event_Message *Pptr)	36
extern int TestForDuplicate(struct Event_Message *Testptr)	36
extern int QueueColorCompare(unsigned int ColorTag, struct Event_Message *Pptr)	36
extern int TestForDiffColor(unsigned int ColorTag)	37
extern void PrintQueue(FILE *out, struct Event_Message *Pptr, char *Emark)	38
extern void QueuesTest()	39
extern void PrintEventsInSystem(FILE *out)	39
extern void PrintEventsProcessed(FILE *out)	40
extern void XQueuesPrint(double PTime, int Fed, int Queue, int Tag, int Offset)	41
extern void SetBaseResourceTime()	41
extern void PrintUtilizationResourceTime(FILE *out, double interval, int Replicate)	42
extern void QueuesPrint(FILE *out, int Replicate)	43
extern void PrintFedTime(FILE *out, int Fed, struct Event_Message *ptr)	45
extern struct Event_Message *SetEventMessage(int Action, int Federate, /*EventManager.c */	46
extern struct Event_Message *SetExtendEventMessage(	47
extern void ResourceVerification(FILE *out, int Fed, int Que)	48
extern double rtingr_RTIEventTEST(struct Event_Message *ptr)	49
extern double SimModelTEST(struct Event_Message *Sptr)	50
extern void CreateRTIreport(char *Which, int ColorTag, int NumberOfDestinations,	51
/* file: FedNodes.c */	53
extern void GetUnitsByCoTypeForRegion(FILE *out, int Category, /* FedNodes.c */	54
extern void CreateNodes(FILE *out,	55
extern void AddRatioUnitsToNode(FILE *out,	58
extern void AddSupportUnitsToNode(FILE *out,	60
extern int AddToNode(FILE *out,	62
extern void RemoveNodeLinksToRegionsByUnit(FILE *out, /* FedNodes.c */	63
extern void AddNodeLinksToRegionsByUnit(FILE *out,	64
extern void AddRegionToNode(struct Nodes_of_Fed_List *FedNode, /* FedNodes.c */	66
extern struct Nodes_of_Fed_List *FindNode(int Id, /* FedNodes.c */	67
extern void PrintNodesOffFed(FILE *out, /* FedNodes.c */	68
extern void PrintUnitsOffFed(FILE *out, /* FedNodes.c */	69
extern int SubcribeByFederate(FILE *out, /* FedNodes.c */	70

30 June 1999

extern int PublishByFederate(FILE *out, ...	72
/* file: FilterUnits.c */	74
extern int AddToFilterSubordinates(FILE *out, ...	75
extern int EquipInList(struct Filter_Unit_List *List)	76
extern struct Filter_Unit_List *FilterByEcheleon(FILE *out, ...	77
extern struct Filter_Unit_List *FilterNotAssignedToFed(FILE *out, ...	79
extern struct Filter_Unit_List *FilterNotInRegion(FILE *out, ...	80
extern void MergeFilterList(struct Filter_Unit_List *List1, ...	81
extern void PrintFilterList(FILE *out, ...	82
/* file: GrowArmy.c */	83
extern int imin(int A, int B) /* GrowArmy */	84
extern int imax(int A, int B)	84
extern int PickSome(int LowBound, int UpBound)	84
extern struct Unit_Characteristics *Grow_Unit_Characteristics(.....	85
extern struct Unit_List *Grow_Unit_List(char *sptr)	87
extern void Grow_Echeleon(FILE *out, ...	88
extern void GrowInitArmy(int GreenBattalions, int OtherBattalions, ...	90
extern double triangle(double c)	97
/* file: InterestGroups.c */	98
extern void GrowInterestGroup(struct Unit_Characteristics *UnCrA, ...	99
extern void Print_InterestGroup(FILE *out, struct Unit_Characteristics *UnCrA)	100
extern void Draw_InterestGroup(struct Unit_Characteristics *UnCrA)	100
/* file: ReadOrdr.c */	101
extern int ReadOrder(FILE *fptr, struct Order_Packet *pkt)	102
extern struct Order_Packet *MakeOrder(int aPcktType, ...	103
extern struct Order_Packet *DuplicateOrder(struct Order_Packet *pkt)	104
extern int SetMsgOrig(struct Order_Packet *pkt, ...	105
extern int SetMsgDest(struct Order_Packet *pkt, ...	105
extern int MsgIdTag(struct Order_Packet *pkt)	105
extern void Print_LotI(FILE *out, ...	105
extern void Print_Route(FILE *out, ...	106
extern void Print_Order(FILE *out, ...	107
extern int Print_OrderQue(FILE *out, ...	108
extern int Free_OrderQue(FILE *out, ...	109
extern void Print_NewOrder(FILE *out, ...	110
/* file: Regions.c */	111
All functions for Regions	112
extern void CreateRegions(FILE *out, ...	114
extern void AddCommandRegions(FILE *out, ...	117
extern void AddNewRegion(FILE *out, ...	119
extern void AddRegionReference(struct Unit_Region_List **RegOfUnit, ...	121
extern void AddToRegion(FILE *out, ...	122
extern int AddToRegionElements(FILE *out, ...	123
extern struct Filter_Unit_List *CmdUnitNotInRegion(.....	124
extern struct Unit_Characteristics *FillRegion(FILE *out, ...	125
extern struct Region_List *FindRegion(int RegId, ...	127
extern struct Filter_Unit_List *PutCoInRegionInFilterList(/* Regions.c */	128
extern struct Filter_Unit_List *PutAllInRegBySideInFilterList(int Side, ...	129
extern struct Filter_Unit_List *PutNumInRegBySideInFilterList(int Number, ...	130
extern struct Filter_Unit_List *PutRegionInFilterList(.....	131
extern int RemoveRegionReference(.....	132
extern void PrintRegionElements(FILE *out, ...	134
extern void PrintRegionsNodes(FILE *out, ...	135
extern void PrintRegions(FILE *out, /* Regions.c */	136
extern void PrintOneRegion(FILE *out, ...	137
extern int RegisterRegions(FILE *out, ...	138
/* file: SOAcreat.c */	140
extern struct Comm_Net_Association *c_Comm_Net_Association(char *sptr)	141
extern struct Comm_Net_List *c_Comm_Net_List(char *sptr)	141

extern struct Truth_Group_List *c_Truth_Group_List(char *sptr)	141
extern struct Node_Table_Def *c_Node_Table_Def(char *sptr)	142
extern struct Node_List *c_Node_List(char *sptr)	142
extern struct Unit_List *c_Unit_List(char *sptr)	143
extern struct InterestList *c_InterestList(char *sptr)	143
extern struct Order_Packet *c_Order_Packet(char *sptr)	143
extern struct Task_List *c_Task_List(char *sptr)	144
extern struct Serv_Characteristics *c_Serv_Characteristics(char *sptr)	144
extern struct Serv_List *c_Serv_List(char *sptr)	144
extern struct Serv_Stack *c_Serv_Stack(char *sptr)	145
extern struct Region_Definition *c_Region_Definition(char *sptr)	145
extern struct Region_Node_Handle *c_Region_Node_Handle(char *sptr)	145
extern struct Units_on_Node_List *c_Units_on_Node_List(char *sptr)	145
extern struct Nodes_wrt_Region_List *c_Nodes_wrt_Region_List(char *sptr)	146
extern struct Nodes_of_Fed_List *c_Nodes_of_Fed_List(char *sptr)	146
extern struct Region_List *c_Region_List(char *sptr)	147
extern struct Region_Element_List *c_Region_Element_List(char *sptr)	147
extern struct Unit_Region_List *c_Unit_Region_List(char *sptr)	148
extern struct Filter_Unit_List *c_Filter_Unit_List(char *sptr)	148
extern struct Federate_Destination *c_Federate_Destination(char *sptr)	148
extern struct Event_Message *c_Event_Message(char *sptr)	148
extern struct Event_Message *c_Duplicate_Event_Message(struct Event_Message *A)	149
/* file: SOAdestr.c */	151
extern void d_Comm_Net_Association(struct Comm_Net_Association *dptr)	151
extern void d_Comm_Net_List(struct Comm_Net_List *sptr)	151
extern void d_Truth_Group_List(struct Truth_Group_List *sptr)	151
extern void d_Node_Table_Def(struct Node_Table_Def *sptr)	151
extern void d_Node_List(struct Node_List *sptr)	151
extern void d_Unit_Characteristics(struct Unit_Characteristics *sptr)	151
extern void d_Unit_List(struct Unit_List *sptr)	151
extern void d_Event_Message(struct Event_Message *sptr)	151
extern void d_Order_Packet(struct Order_Packet *sptr)	151
extern void d_Filter_Unit_List(struct Filter_Unit_List *sptr)	151
extern void d_Unit_Region_List(struct Unit_Region_List *sptr)	152
extern void d_Region_Element_List(struct Region_Element_List *sptr)	152
extern void d_Task_List(struct Task_List *sptr)	152
extern void d_Serv_Characteristics(struct Serv_Characteristics *sptr)	152
extern void d_Serv_List(struct Serv_List *sptr)	152
extern void d_Serv_Stack(struct Serv_Stack *sptr)	152
extern void d_Federate_Destination(struct Federate_Destination *sptr)	152
/* file: Sim.c */	153
main(int argc, char *argv[]) /* int GreenBattalions, int OtherBattalions */	154
/* file: SimModel.c */	158
extern double SimModel(struct Event_Message *ptr, double PhysicalTime, ...	159
extern void UpdateEntity(struct Event_Message *ptr, double PhysicalTime, ...	164
extern void CreateInteraction(struct Event_Message *ptr, double PhysicalTime, ...	165
extern void InitSimModel(struct Region_Node_Handle *RNH)	166
extern void InitSimModelTest(struct Region_Node_Handle *RNH)	170
/* file: UnitCharFile.c */	172
extern int MaxEcheleon(FILE *out, struct Unit_Characteristics *UnCrA, char *str)	173
extern void Initialize_Friends()	173
extern void Initialize_Others()	173
extern int UnitCharacter(struct Unit_Characteristics *uptr, ...	173
extern void Print_UnitC_File(char filename[], ...	175
extern void Print_UnitC(FILE *out, ...	175
extern void Print_UnitC_comma(FILE *out, ...	176
/* file: UnitEcheleon.c */	177
extern int CountSubrEquip(FILE *out, ...	178
extern void ResetEcheleon(int Iset)	178

30 June 1999

extern void TallyEcheleon(struct Unit_List *ULp) /* UnitEcheleon.c */	178
extern void TallyClearEch()	179
extern void TallyPrintEch(FILE *out, char *label)	179
extern int GetTotalEquipByLevel(int i)	179
extern unsigned int GetTotalEquip()	179
extern int GetTotalPersonByLevel(int i)	180
extern void Print_EchSummary(FILE *out)	180
extern void Print_Echeleon(FILE *out, /* UnitEcheleon.c */	181
extern void PrintRTIEchelon(FILE *out,	182
extern void PrintRTIInstanceEchelon(FILE *out,	183
extern void ViewNext()	185
extern void ViewNew() /* UnitEcheleon.c */	185
extern void ViewEcheleonLeft(struct Unit_List *ULp)	185
extern void ViewEcheleonRight(struct Unit_List *ULp)	185
extern void ViewRefresh(struct Unit_List *ULp)	186
/* file: data_distrib_mgmt_svc.c */	188
extern double ddm_create_update_region(int federate_nbr, int region_nbr)	189
extern double ddm_associate_update_region(int federate_nbr)	190
/* file: declar_mgmt_svc.c */	191
extern double dm_publish_objclass(int federate_nbr, int class_nbr)	192
extern double dm_publish_interact_class(int federate_nbr,	193
static SUBSCRIBED_INFO_TYPE *create_subscribed_node(int class_nbr,	194
void add_federate_to_regions_table(int class_nbr,	195
extern double dm_subscribe_objclass(int obj_class,	196
extern double dm_subscribe_interact_class(int federate_name,	197
/* file: event_manager.c */	198
extern void eventmgr_process_event(EVENT_MESSAGE_TYPE *event_msg)	199
extern void eventmgr_change_processing_mode(EVENT_MESSAGE_TYPE *event_msg,	200
static double eventmgr_determine_events_LBTS(int federate_nbr)	201
extern void eventmgr_retrieve_LBTS_info(int federate_nbr,	202
extern int eventmgr_get_parent_name(int federate_nbr)	203
extern int eventmgr_get_destination(int federate_nbr,	204
/* file: fedrtn_mgmt_svc.c	205
extern double fm_create_fedrtn_execution(int federate_nbr,	206
static void fm_initialize_federate(FEDERATE_INFO_TYPE *federate,	207
extern double fm_join_fedrtn_execution(int federate_name,	208
extern double fm_request_fedrtn_save(int federate_nbr,	209
extern double fm_initiate_federate_save()	210
static double fm_setup_initiate_federate_save_events(int active_federates,	211
static void add_federate_to_feds_saving_list(int federate_nbr)	212
extern double fm_federate_save_begun(int federate_nbr,	213
extern double fm_federate_save_achieved(int federate_nbr)	214
static int is_federate_in_fed_saved_list(int federate_nbr)	215
extern int fm_is_fedrtn_saved()	216
extern double fm_fedrtn_save_achieved()	217
extern double fm_setup_fedrtn_complete_events(int federate_nbr,	218
/* file: init_rti.c */	219
void RIDdb_init()	220
extern void Initialize_RTI()	221
/* file: io_manager.c */	222
static int iomgr_determine_iochannel()	223
extern void iomgr_send_ioevent(EVENT_MESSAGE_TYPE *event_msg_info_ptr,	224
/* file: mood.c Xwindow utilities */	226
extern void get_GC(Window win, GC *gc, XFontStruct *font_info)	228
void load_font(XFontStruct **font_info)	229
extern void draw_text(	230
extern void draw_graphics(	232
extern void TooSmall(	233
extern void LalaInit(int TotNodes, int TotObjects)	234

30 June 1999

extern void LalaClear() {	236
extern void LalaUpdate(int Node, int ObjId, int State)	237
extern void LalaColor(int State)	237
extern void LalaPlace(int State, int X, int Y)	237
extern void LalaDraw(int Node, int ObjId, int State, ...	238
extern void LalaTimeQueue(int Node, int State, ...	239
extern void LalaDrawLink(int State, ...	239
extern void LalaFinished()	240
/* file: object_mgmt_svc.c */	241
static REGIONS_LIST_TYPE *create_regions_node(int region_nbr)	242
static OBJECT_INSTANCE_TYPE *create_obj_instance(REGIONS_LIST_TYPE *regions_ptr, ...	243
static void add_obj_to_federates_table(OBJECT_INSTANCE_TYPE *node, ...	244
extern int om_is_class_published(int class_nbr, ...	245
extern int om_instance_exists(int instance_nbr)	246
extern double om_register_instance(int class_nbr, ...	247
extern int om_is_object_registered(int instance_nbr)	249
extern double om_update_attrb_values(int federate_nbr, ...	250
extern double om_send_interaction(int federate_nbr, ...	251
extern double om_request_attrb_value_update()	252
extern FEDERATE_DESTINS_TYPE *om_create_destinations_element(int federate_nbr)	253
extern double om_discover_object(int federate_nbr, ...	254
extern double om_setup_discover_events(int federate_nbr, ...	255
extern double om_reflect_attrb_values(int instance_nbr, ...	257
extern double om_setup_reflect_events(int federate_nbr, ...	258
extern double om_receive_interaction(int class_nbr, ...	260
extern double om_setup_receive_interaction_events(int federate_nbr, ...	261
extern double om_provide_attrb_value_update()	263
/* file: rti_utils.c */	264
extern int get_string_peices(char *lstr, char *pieces[], char *delimiter)	265
extern int change_strgeices_to_onestrng(int num_peices, ...	266
/* file: rti_manager.c */	267
static int rtimgr_criteria_create(RTI_SERVICE_TBL_ENTRY_TYPE *rtisvc_tbl_ptr)	268
static void rtimgr_FOMdb_init()	271
extern void rtimgr_clear_reduction_network_info()	273
static void rtimgr_FedExdb_init()	274
extern void rtimgr_init(RTI_SERVICE_TBL_ENTRY_TYPE *rtisvc_tbl_ptr)	276
extern int rtimgr_federate_processed_initial_counts(int federate_nbr)	278
extern int rtimgr_federate_is_parent(int federate_nbr)	279
static void rtimgr_update_fedrtn_state_status(FEDEX_STATE_INFO fedex_state)	280
static void rtimgr_printsvc_stat(int federate_nbr, ...	281
static double rtimgr_compute_elapsed_time_statistic(int federate_nbr, ...	282
static double rtimgr_process_rtiamb_svc(int rtiamb_action, ...	283
extern double rtimgr_process_fedamb_svc(int fedamb_reaction, ...	288
static int rtimgr_is_fedamb_svc(int fedamb_svcnbr, ...	291
extern double rtimgr_RTlevent(EVENT_MESSAGE_TYPE *event_msg_info_ptr)	292
static void rtimgr_retrieve_svctblinfo(RTI_EVENT_MSG_TYPE *svc_msg_info, ...	294
static int rtimgr_criteria_compare(FEDEX_STATE_INFO fedex_state_status, ...	295
extern double rtimgr_get_RTI_ambsvc_time(int federate_nbr, ...	296
extern double rtimgr_get_Fed_ambsvc_time(int fed_ambsvc, ...	300
extern void rtimgr_clear()	301
extern void rtimgr_final_cleanup()	302
extern void statsmgr_init_statruns_file()	305
extern int initialize_statistic(STATISTIC_CPM_TYPE *stat_entry, ...	306
extern void statsmgr_setup_stats_tables()	307
extern double statsmgr_lookup_in_fomdb(int nbr_federates)	308
extern double statsmgr_lookup_in_fedexdb(int nbr_federates)	309
extern double statsmgr_setup_fed_in_fedexdb(int nbr_federates)	310
extern double statsmgr_setup_region_in_fedexdb(int nbr_federates)	311
extern double statsmgr_setup_class_in_fedexdb(int nbr_federates)	312

30 June 1999

extern double statsmgr_setup_instance_in_fedexdb(int nbr_federates).....	313
extern double statsmgr_compute_nbr_nodes(int nbr_federates).....	314
extern double statsmgr_forward_to_sim_model(int nbr_federates).....	315
extern double statsmgr_norm_distrib(int nbr_federates)	316
extern int statsmgr_get_statsarray_index(int action,.....	317
extern void statsmgr_accum_totals(double replicate_time, ...	318
extern void statsmgr_collect_statistic(int stat_entry_index, ...	320
extern void statsmgr_print_accum_totals().....	321
extern void statsmgr_clear_accum_totals().....	322
extern void statsmgr_cleanup().....	323
/* file: time_mgmt_svc.c */.....	324
extern double tm_LBTS_requests_setup(int federate_nbr,.....	325
extern double tm_time_adv_request(int federate_nbr,.....	326
static int all_my_subfederates_reported(int federate_nbr).....	328
extern double tm_controller_LBTS_compute(int federate_nbr, ...	329
extern void tm_forward_LBTS_info(int from_federate_nbr,.....	333
extern double tm_time_adv_grant_setup(int federate_nbr,.....	334
extern double tm_query_fed_LBTS(int federate_nbr, ...	335
extern double tm_time_adv_grant(int federate_nbr,.....	337
extern void tm_clear_LBTS_info().....	338
/* file: event.h */ /* network types */.....	339
/* eventmgr.h */	340
/* io_mgr.h */.....	342
/* eventmgr.h */	342
/* proto.h */.....	347
/* regions.h */.....	351
/* rti.h */.....	352
/* rti_services.h */.....	355
/* rtimgr.h */	355
/* serv_crit.h */	358
/* soaGcnst.h */.....	359
/* soa_cnst.h */.....	359
/* soa_defs.h */	360
typedef struct Region_Definition { /* regions for node distribution */.....	362
typedef struct Node_Definition { /* define nodes of simulation */.....	362
typedef struct Region_Node_Handle {	362
typedef struct Nodes_of_Fed_List {	362
typedef struct Units_on_Node_List {	363
typedef struct Nodes_wrt_Region_List {	363
typedef struct Region_List {	363
typedef struct Region_Element_List {	363
typedef struct Unit_Region_List {	363
typedef struct Filter_Unit_List {	363
typedef struct Unit_Characteristics {	363
typedef struct Unit_List {	364
typedef struct InterestList {	364
/* statsmgr.h */.....	367

```

/* File: EventManager.c */
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <math.h>
// #include "event.h"
#include "soa_defs.h"
#include "soa_cnst.h"
#include "rti_services.h"
// #include "serv_crit.h"
#include "proto.h"
#define DEMOONLY 0
// MaxFederates counts from 1,2,...n
// so the C or C++ index limit is n+1
// #define Largest_Number 0.1E+15
#define Largest_Number 9300.0

#define EXPAND 0.0
#define LATENCY 0.001

#define OutQ 0
#define RtiQ 1
#define TsoQ 2
#define SimQ 2

#define Low 0
#define High 1

#define MaxFederates SCENARIOLimitsOnFederates /* tmp 5 , from 10 */
#define MaxNetXfers 1
#define MaxResources (MaxFederates + MaxNetXfers)
#define Ether (MaxFederates + MaxNetXfers-1)
#define MaxQueues 3
#define LowToHigh 2
#define Statistics 5
#define MaxHistory 100
#define StateCol 10
#define EvenOrOdd 2

static double XBaseTime = 0.0, XIntervalTime = 0.10 ;
static int RTI_TIME_ADV_RQST_IdOffset = 2 ;
static int RTI_TIME_ADV_GRANT_IdOffset = 2 ;
static int RTI_RPTNG_FED_LBTS_IdOffset = 2 ;
static int RTI_RPTNG_RCV_LBTS_IdOffset = 2 ;
static int RTI_RPTNG_SND_LBTS_IdOffset = 2 ;
static int RTI_QUERY_FED_LBTS_IdOffset = 2 ;
static int WiggleColor_IdOffset = 4 ;

static int QueEmpty = 0 ;
static int InitSimOnce = 1 ;
static int StepThroughThisBeast = 0 ;
static struct Event_Message *Qp[MaxResources][MaxQueues];
/* NOTE: QueuesInitialize() will initialize the Federation Color values */
/* ColorMode: start with odd and increment thereafter */
/* use the LSB (Least Significant Bit) to indicate even or odd */
static unsigned int FederationColorTag = 1 ; /* start at 1 */
static unsigned int FederateCurrentColor[MaxFederates+1][EvenOrOdd];
static unsigned int FederateColorReceived[MaxFederates+1][EvenOrOdd];
static unsigned int FederateColorSent[MaxFederates+1][EvenOrOdd];
static int FedColorIdx[MaxFederates+1]; /* This will indicate even or odd*/

static int LastFed=0, LastQ=0 ;
static double LastTime = 0.0 ;
static double RTIServiceBase = 0.005 ;

```

30 June 1999

```

static double          SIMServiceBase = 0.01 ;
static double          EtherService = 0.001;
static double  LastEtherService = 0.0;
static double  VirtualTime  = 0.0; /* GVT */
static double  PhysicalTime = 0.0; /* System Current Time */
static double  PreviousPhysicalTime = 0.0 ;

//static double HighResource ;
static double MinNextEvent [MaxResources] ;
static int    FederateQueue [MaxResources] ;
static int    FederateCongested [MaxResources] ;

static double  StateTime [MaxResources] [StateCol];
static double  StateTpre [MaxResources] [StateCol];
static double  StateQend [MaxResources] [StateCol];

static double  Interrupts [MaxResources] ;
//static double LogicalTime [MaxResources];
static double  FederateTime [MaxResources]; /* EndTime */
static double  ResourceTime [MaxResources];
static double  BaseResourceTime [MaxResources];
static double  FederateDelay [MaxResources];
static double  LastResourceTime [MaxResources];
static double  FederateLBTSLimit [MaxResources];
//static double FederateMinEvent [MaxResources];
static double  LastService [MaxResources];
//static double PreviousValues [MaxResources+1] = { 0.0 } ;

static int      Ql [MaxResources] [MaxQueues]; /* length of queue */
static int      Qn [MaxResources] [MaxQueues]; /* number passed though */
static double   Qs [MaxResources] [MaxQueues] [Statistics];
static struct History  HistOfQue [MaxHistory];
int  Hidx = 0;
char str[12];
/* prototypes */ /* This can be used by the rtimgr_RTieventTEST & SimModelTEST*/
static unsigned short int EMX[3] ;
extern double erand48( unsigned short int X[3]);

extern double rtimgr_RTievent( struct Event_Message *ptr) ;

extern double rtimgr_RTieventTEST( struct Event_Message *ptr) ;
extern double SimModelTEST( struct Event_Message *ptr) ;

extern int QueueCompare( struct Event_Message *Testptr, struct Event_Message *Pptr );
extern int TestForDuplicate( struct Event_Message *Testptr);
extern int QueueColorCompare( unsigned int ColorTag, struct Event_Message *Pptr );
/*----- EventManager.c ----- f*/

/*----- EventManager.c ----- f*/
/*----- DocHeading */

```

30 June 1999

```

extern double CurrentPhysicalTime() {
return (PhysicalTime);
}

/*----- EventManager.c ----- f*/

/*----- DocMethod --- */
extern double CurrentFederateTime(int Fed) {
if ( Fed > 0 && Fed <= MaxFederates ) {
return( FederateDelay[Fed-1] );
}
else {
return(0);
}
}
/* replace StartLBTS Calculation with eventmgr_retrieve_LBTS_info above or not*/
/*----- EventManager.c ----- f*/
/*----- DocMethod --- */
extern unsigned int StartLBTS Calculation()
/*-----EndDocHead---*/
{
FederationColorTag += 1 ; /* goes to the next color */
return( FederationColorTag );
}

/*----- EventManager.c ----- f*/
/*----- DocMethod --- */
extern double GetLBTSfromFederate( int FedIdPlus ) {

struct Event_Message *Next ;
double dtemp ;

Next = Qp[FedIdPlus-1][2] ; /* lowest time on queue */
if ( Next != NULL ) {
dtemp = Next->Time.PhysicalTime ;
}
else {
dtemp = FederateDelay[FedIdPlus-1] ; /* know time for this federate*/
}
return( dtemp );
}
/*----- DocMethod --- */
extern unsigned int CurrentFederationColor()
/*-----EndDocHead---*/
{
return( FederationColorTag ); /* the color */
}

/*----- EventManager.c ----- f*/
/* ChangeFederateColorTag:
* Return a federate's msg counts and toggle its color mode and LBTSstate
*/

/*----- DocMethod --- */
extern void ChangeFederateColorTag( int Federate, int *Sent, int *Received )
/*-----EndDocHead---*/
{
extern int SetRtiColors( int Fed, int ColorTag );
extern int CountRtiColors( int Fed, int ColorTag );
int color, i;
*Sent = FederateColorSent[Federate-1][FedColorIndx[Federate-1]] ;
*Received = FederateColorReceived[Federate-1][FedColorIndx[Federate-1]] ;

FedColorIndx[Federate-1] += 1 ;
if ( FedColorIndx[Federate-1] >= 2 ) {
FedColorIndx[Federate-1] = 0 ; /* just toggle between 0 and 1 */
}
}

```

```

}
color = FederationColorTag;
    FederateColorSent[Federate-1][FedColorIndx[Federate-1]] = 0 ;
    FederateColorReceived[Federate-1][FedColorIndx[Federate-1]] = 0 ;
if ( FederateCurrentColor[Federate-1][FedColorIndx[Federate-1]] < FederationColorTag) {
    FederateCurrentColor[Federate-1][FedColorIndx[Federate-1]] = FederationColorTag ;
}
else if (FederateCurrentColor[Federate-1][FedColorIndx[Federate-1]] == FederationColorTag) {
    fprintf(stdout,"ColorTag1:Already done \n");
}
else {
    fprintf(stdout,"ColorTag2:Don't know whats going on? \n");
}
/* count the number in the queues for this federate */
i = SetRtiColors( Federate, color );
// i = CountRtiColors( Federate, color );

}
/*----- EventManager.c ----- f*/
/*----- DocMethod --- */
extern int CountRtiColors( int Fed, int ColorTag )
/*-----EndDocHead---*/
{
    int i,j , t;
    int Que, Depth;
    char str[12];
    struct Event_Message *ptr ;
    i = 0;

    ptr = Qp[Fed-1][OutQ];
    if ( ptr != NULL) {
        // cycle thru link list for this queue , and count up events
        // of this color and reset
        while ( ptr != NULL ) {
            if ( ptr->Rti.rti_svc_nbr == RTI_REFLECT_ATTRIB ||
                ptr->Rti.rti_svc_nbr == RTI_RECEIVE_INT ) {
                if (ColorTag != ptr->Color.ColorTag ) {

                    // fprintf( stdout,
                    // "COLOR%02d..%2d %s fed.At %9.4f ColorTag %3d Differ From %3d sched for %9.4f Bdry
                    // %2d Fed %2d OrFed %2d Uniq %4d srv %4d\n",
                    // Fed, i, ".....",PhysicalTime,
                    // ColorTag, ptr->Color.ColorTag, ptr->Time.PhysicalTime,
                    // ptr->Color.Boundary, ptr->Rti.federate_name,
                    // ptr->Rti.origin_fed_name, ptr->Time.UniqueMsgId,
                    // ptr->Rti.rti_svc_nbr );
                    // don't change for these RTI_services ptr->Color.ColorTag = ColorTag ;
                    i += 1;
                }
            }
            ptr = ptr->nqep ;
        }
        // QueuesPrint(stdout, -7);
        // fprintf(stdout,"QUERYREFLECTPress Enter \n");
        // gets(str);
    }
    return(i);
}

/*----- EventManager.c ----- f*/
/*----- DocMethod --- */
extern int SetRtiColors( int Fed, int ColorTag )
/*-----EndDocHead---*/
{
    int i,j , t;
    int Que, Depth;

```



```

char str[12];
struct Event_Message *ptr ;
i = 0;

ptr = Qp[Fed-1][RtiQ];
if ( ptr != NULL) {
    // cycle thru link list for this queue , and count up events
    // of this color and reset
    while ( ptr != NULL ) {
        if ( ptr->Rti.rti_svc_nbr == RTI_UPDATE_ATTRIB ||
            ptr->Rti.rti_svc_nbr == RTI_SEND_INT ||
            ptr->Rti.rti_svc_nbr == RTI_REFLECT_ATTRIB ||
            ptr->Rti.rti_svc_nbr == RTI_RECEIVE_INT ) {

            if (ColorTag != ptr->Color.ColorTag ) {

                // fprintf( stdout,
                // "COLOR%02d..%2d %s set.At %9.4f ColorTag %3d Differ From %3d sched for %9.4f Bdry
                %2d Fed %2d OrFed %2d Uniq %4d srv %4d\n",
                // Fed, i, ".....",PhysicalTime,
                // ColorTag, ptr->Color.ColorTag, ptr->Time.PhysicalTime,
                // ptr->Color.Boundary, ptr->Rti.federate_name,
                // ptr->Rti.origin_fed_name, ptr->Time.UniqueMsgId,
                // ptr->Rti.rti_svc_nbr );
                ptr->Color.ColorTag = ColorTag ;
                i += 1;
            }
            ptr = ptr->nqep ;
        }
    }
    ptr = Qp[Fed-1][OutQ];
    if ( ptr != NULL) {
        // cycle thru link list for this queue , and count up events
        // of this color and reset
        while ( ptr != NULL ) {
            if ( ptr->Rti.rti_svc_nbr == RTI_UPDATE_ATTRIB ||
                ptr->Rti.rti_svc_nbr == RTI_SEND_INT ||
                ptr->Rti.rti_svc_nbr == RTI_REFLECT_ATTRIB ||
                ptr->Rti.rti_svc_nbr == RTI_RECEIVE_INT ) {

                if (ColorTag != ptr->Color.ColorTag ) {

                    // fprintf( stdout,
                    // "COLOR%02d..%2d %s set.At %9.4f ColorTag %3d Differ From %3d sched for %9.4f Bdry
                    %2d Fed %2d OrFed %2d Uniq %4d srv %4d\n",
                    // Fed, i, ".....",PhysicalTime,
                    // ColorTag, ptr->Color.ColorTag, ptr->Time.PhysicalTime,
                    // ptr->Color.Boundary, ptr->Rti.federate_name,
                    // ptr->Rti.origin_fed_name, ptr->Time.UniqueMsgId,
                    // ptr->Rti.rti_svc_nbr );
                    ptr->Color.ColorTag = ColorTag ;
                    i += 1;
                }
                ptr = ptr->nqep ;
            }
        }
        // QueuesPrint(stdout,-7);
        // fprintf(stdout,"QUERYPress Enter \n");
        //gets(str);
    }
    return(i);
}

/*----- EventManager.c ----- f*/
/*----- DocMethod --- */

```

```

extern int AnyPotentialMessagesToReceiveWithOldColorTag()
/*-----EndDocHead---*/
{
    int i,j , t, count;
    int Que, Depth;
    char str[12];
    struct Event_Message *ptr ;
    i = count = 0;
    for(i=0; i< MaxFederates; i++){
        ptr = Qp[i][OutQ];
        if ( ptr != NULL) {
            // cycle thru link list for this queue , and count up events
            // of this color and reset
            while ( ptr != NULL ) {
                if ( ptr->Rti.rti_svc_nbr == RTI_REFLECT_ATTRIB ||
                    ptr->Rti.rti_svc_nbr == RTI_RPTNG_FED_LBTS ||
                    ptr->Rti.rti_svc_nbr == RTI_RPTNG_RCV_LBTS ||
                    ptr->Rti.rti_svc_nbr == RTI_RPTNG_SND_LBTS ||
                    ptr->Rti.rti_svc_nbr == RTI_RECEIVE_INT ) {

                    if ( FederationColorTag != ptr->Color.ColorTag ||
                        ptr->Rti.rti_svc_nbr == RTI_RPTNG_FED_LBTS ||
                        ptr->Rti.rti_svc_nbr == RTI_RPTNG_RCV_LBTS ||
                        ptr->Rti.rti_svc_nbr == RTI_RPTNG_SND_LBTS ) {

                        //      fprintf( stdout,
                        //      "COLOR%02d..%2d %s set.At %9.4f ColorTag %3d Differ From %3d sched for %9.4f Bdry
                        //      %2d Fed %2d OrFed %2d Uniq %4d srv %4d\n",
                        //      i+1, i, "...ANY...",PhysicalTime,
                        //      ColorTag, ptr->Color.ColorTag, ptr->Time.PhysicalTime,
                        //      ptr->Color.Boundary, ptr->Rti.federate_name,
                        //      ptr->Rti.origin_fed_name, ptr->Time.UniqueMsgId,
                        //      ptr->Rti.rti_svc_nbr );
                        count += 1;
                    }
                }
                ptr = ptr->nqep ;
            }

            fprintf(stdout,"QUERYPress Enter \n");
            //gets(str);
        }
    }
    QueuesPrint(stdout,-7);
    return(count);
}
/*----- EventManager.c ----- f*/

/*----- DocMethod */
extern void ColorTag(int Federate, struct Event_Message *Add)
/*-----EndDocHead---*/
{
    Add->Color.ColorTag = FederateCurrentColor[Federate-1][FedColorIndx[Federate-1]] ;
}

/*----- EventManager.c ----- f*/

/*----- DocMethod */
extern unsigned int GetColorTag( int Federate )
{
    return( FederateCurrentColor[Federate-1][FedColorIndx[Federate-1]] ) ;
}

/*----- EventManager.c ----- f*/

```

30 June 1999

```
/*----- DocMethod */
extern int TSOBoundMessage( struct Event_Message *Msg)
{
  int i;
  i = 0 ;
  if ( Msg->Rti.rti_svc_nbr == RTI_REFLECT_ATTRIB ||
      Msg->Rti.rti_svc_nbr == RTI_RECEIVE_INT ) {
    i = 1 ;
  }
  return( i );
}

/*----- EventManager.c ----- f*/

/*----- DocHeading */
```

30 June 1999

```

extern void AddPriorityEvent( FILE *out, char *Type,
                           struct Event_Message *Add )
/*-----EndDocHead-----*/
{
    int    FedNodeId, IdQ , i, TSOMessage;
    int    NumberOfDestinations ;
    struct Federate_Destination *Dest ;
    char   str[12];
    struct Event_Message      *Next, *ptr ;

    /* Have to determine the appropriate queue */

    FedNodeId = Add->Rti.federate_name -1 ; /* FederateOffset */
    if ( Add->Rti.federate_name > SCENARIOILimitsOnFederates || Add == NULL ||
        Add->Rti.federate_name < 1 ) {
        fprintf(out, " This node is too large|small %3d %8.3f \n", Add->Rti.federate_name,
            PhysicalTime );
        // gets(str);
    }

    Add->nqep = NULL ; /* just to be sure */

    //i = TestForDuplicate( Add );
    //if ( i > 0 ) {
    //    printf(" bad very bad need to set breakpoint %3d \n", i);
    //}

    //if ( Add->Time.PhysicalTime < FederateDelay[FedNodeId] ) {
    //    Add->Time.PhysicalTime = FederateDelay[FedNodeId] ;
    //}
    ptr = Add;
    //if 0
    //fprintf( stdout,
    //    "COLOR%02d: %8s.... pry.At %9.4f ColorTag %3d NoDiff From %3d sched for %9.4f Bdry %2d Fed
    //    %2d OrFed %2d Uniq %4d srv %4d\n",
    //    Add->Rti.federate_name, Type, PhysicalTime,
    //    ptr->Color.ColorTag, ptr->Color.ColorTag, ptr->Time.PhysicalTime,
    //    ptr->Color.Boundary, ptr->Rti.federate_name,
    //    ptr->Rti.origin_fed_name, ptr->Time.UniqueMsgId,
    //    ptr->Rti.rti_svc_nbr );
    //QueuesPrint(stdout,-7);
    //endif /* Select the queue from the contents of the message to Add */

    if ( Add != NULL ) {

        if ( Add->nqep != NULL ) { Add->nqep = NULL; }

        if ( strcmp( Type, "Out", 3 ) == 0 ||
            strcmp( Type, "OUT", 3 ) == 0 ) {

            IdQ = OutQ ;
            Add->Time.OutEnter = PhysicalTime ;
            /* WILL - when an event is a TSO event for the outq: Update/Send Interact
             * we need to affect the send msg count for this federate
             * 1) need to get the nbr_dests by either passing in the count or
             * counting up the nbr of federates on the dests list rcvd
             * 2) add stmts here, something like what follows, to increment sends for colormode
             * e.g.,if (FederationColorTag == even)
             *     FederateEvenModeSent[federate_nbr] += nbr_dests;
             *     else
             *     FederateOddModeSent[federate_nbr] += nbr_dests;
             */
            Dest = Add->destinations_list;
            if ( Dest != NULL ) { /* should not be in the out que without destination */
                TSOMessage = TSOBoundMessage(Add) ;
            }
        }
    }
}

```

30 June 1999

```

if ( TSOMessage == 1 ) {
    NumberOfDestinations = 0 ;
    do {
        NumberOfDestinations += 1 ;
        Dest = Dest->next;
    } while ( Dest != NULL );

    if (Add->Color.ColorTag ==
        FederateCurrentColor[FedNodeId][FedColorIndx[FedNodeId] ] ) {
        /* this fed sending */
        FederateColorSent[FedNodeId][ FedColorIndx[FedNodeId] ] +=
        NumberOfDestinations ;
    }
    else { /* not this color can be bad or previous color */
        if ( Add->Color.ColorTag < /*
removed = */
            FederateCurrentColor[FedNodeId][FedColorIndx[FedNodeId] ] ) {
            /* need to call function that handles straggler message */
            /* and send a message with a count of one for this destination */
            CreateRTIreport("Send", Add->Color.ColorTag,NumberOfDestinations,
                PhysicalTime, (FedNodeId+1) );
            printf("CreateReport Federate %2d (+1) FederateColor not known %3d for
service PRIev %4d \n", FedNodeId,
                Add->Color.ColorTag, Add->Rti.rti_svc_nbr);
        }
        else {
            printf("For Federate %2d (+1) FederateColor not known %3d \n", FedNodeId,
                Add->Color.ColorTag );
        }
    }
}
Next = Qp[FedNodeId][IdQ] ;
Add->nqep = Next ;
Qp[FedNodeId][IdQ] = Add ;
Ql[FedNodeId][IdQ] += 1;
Qn[FedNodeId][IdQ] += 1;
}
else if ( strcmp( Type, "InToRTI", 7) == 0 ||
    strcmp( Type, "RTI", 3) == 0 ) {

    IdQ = RtiQ ;
    Add->Time.RTIEnter = PhysicalTime ;

    if ( Add->Time.TsoRtService <= 0.0 ) { Add->Time.TsoRtService = PhysicalTime; }

    Next = Qp[FedNodeId][IdQ] ;
    Add->nqep = Next ;
    Qp[FedNodeId][IdQ] = Add ;
    Ql[FedNodeId][IdQ] += 1;
    Qn[FedNodeId][IdQ] += 1;
}
else if ( strcmp( Type, "TSO", 3) == 0 ||
    strcmp( Type, "Tso", 3) == 0 ) {

    IdQ = TsoQ ;

    Add->Time.TsoEnter = PhysicalTime ;

    Next = Qp[FedNodeId][IdQ] ;
    Add->nqep = Next ;
    Qp[FedNodeId][IdQ] = Add ;
    Ql[FedNodeId][IdQ] += 1;
    Qn[FedNodeId][IdQ] += 1;
}
}

```

```
else {
    IdQ = SimQ ;                                /* This should not happen */

    fprintf(out,
        "AddPriorityEvent neither Out, InToRTIR, or TSO type event  Fed %3d, Phys T %12.6f\n",
        FedNodeId, PhysicalTime );
}
// QueuesPrint(stdout,-7);
//     fprintf(stdout,"PRIORITY QUERY Press Enter \n");
//     gets(str);
//
//
/*----- EventManager.c ----- f*/
/*-----                               DocHeading */
```

```
extern void AddEvent( FILE *out, char *Type,
                    struct Event_Message *Add )
/*-----EndDocHead-----*/
{
    int    FedNodeId, IdQ , i, TSOMessage;
    int    NumberOfDestinations ;
    struct Federate_Destination *Dest ;
    char   str[12];

    /* Have to determine the appropriate queue */

    FedNodeId = Add->Rti.federate_name -1 ; /* FederateOffset */
    if ( Add->Rti.federate_name > SCENARIOILimitsOnFederates || Add == NULL ||
        Add->Rti.federate_name < 1 ) {
        fprintf(out, " This node is too large|small %3d %8.3f \n", Add->Rti.federate_name,
            PhysicalTime );
        // gets(str);
    }
    Add->nqep = NULL ; /* just to be sure */

    //i = TestForDuplicate( Add );
    //if ( i > 0 ) {
    //    printf(" bad very bad need to set breakpoint %3d \n", i);
    //}
    if ( Add->Time.PhysicalTime == 0.0 ) {
        Add->Time.PhysicalTime = PhysicalTime ;
    }
    /* Select the queue from the contents of the message to Add */
    if ( Add != NULL ) {
        if ( Add->nqep != NULL ) { Add->nqep = NULL; }

        if ( strcmp( Type, "Out", 3 ) == 0 ||
            strcmp( Type, "OUT", 3 ) == 0 ) {

            IdQ = OutQ ;
            Add->Time.OutEnter = PhysicalTime ;
            /* WILL - when an event is a TSO event for the outq: Update/Send Interact
             * we need to affect the send msg count for this federate
             * 1) need to get the nbr_dests by either passing in the count or
             * counting up the nbr of federates on the dests list rcvd
             * 2) add stmts here, something like what follows, to increment sends for colormode
             * e.g.,if (FederationColorTag == even)
             *     FederateEvenModeSent[federate_nbr] += nbr_dests;
             * else
             *     FederateOddModeSent[federate_nbr] += nbr_dests;
             */
            Dest = Add->destinations_list;
            if ( Dest != NULL ) { /* should not be in the out que without destination */
                TSOMessage = TSOBoundMessage(Add) ;
                if ( TSOMessage == 1 ) {
                    NumberOfDestinations = 0 ;
                    do {
                        NumberOfDestinations += 1 ;
                        Dest = Dest->next;
                    } while ( Dest != NULL );

                    if (Add->Color.ColorTag ==
                        FederateCurrentColor[FedNodeId][FedColorIndx[FedNodeId] ] ) {
                        /* this fed sending */
                        FederateColorSent[FedNodeId][ FedColorIndx[FedNodeId] ] +=
                            NumberOfDestinations ;
                    }
                    else { /* not this color can be bad or previous color */

```

30 June 1999

/*

```

removed = */
        if ( Add->Color.ColorTag <
                FederateCurrentColor[FedNodeId][FedColorIndx[FedNodeId] ] ) {
            /* need to call function that handles straggler message */
            /* and send a message with a count of one for this destination */

            CreateRTIreport("Send", Add->Color.ColorTag,NumberOfDestinations,
                PhysicalTime, (FedNodeId+1) );
            printf("CreateReport Federate %2d (+1) FederateColor not known %3d for
service ADDev %4d \n", FedNodeId,
                Add->Color.ColorTag, Add->Rti.rti_svc_nbr);
        }
        else {
            printf("For Federate %2d (+1) FederateColor not known %3d for service
%4d \n", FedNodeId,
                Add->Color.ColorTag, Add->Rti.rti_svc_nbr);
        }
    }
}
AddEventToQueue(out, &Qp[FedNodeId][OutQ], FedNodeId, IdQ, Add);
}
else if ( strncmp( Type, "InToRTI", 7) == 0 ||
        strncmp( Type, "RTI", 3) == 0 ) {

    IdQ = RtiQ ;
    Add->Time.RTIEnter = PhysicalTime ;

    if ( Add->Time.TsoRtService <= 0.0 ) { Add->Time.TsoRtService = PhysicalTime; }

    AddEventToQueue(out, &Qp[FedNodeId][RtiQ], FedNodeId, IdQ, Add);
}
else if ( strncmp( Type, "TSO", 3) == 0 ||
        strncmp( Type, "Tso", 3) == 0 ) {

    IdQ = TsoQ ;

    Add->Time.TsoEnter = PhysicalTime ;

    AddEventToQueue(out, &Qp[FedNodeId][TsoQ], FedNodeId, IdQ, Add);
}
else {
    IdQ = SimQ ;
    /* This should not happen */

    fprintf(out,
        "AddEvent neither Out, InToRTIR, or TSO type event Fed %3d, Phys T %12.6f\n",
        FedNodeId, PhysicalTime );
}
}
}

/*----- EventManager.c ----- f*/
/*----- DocHeading */

```


30 June 1999

```

extern void AddEventToQueue( FILE *out, struct Event_Message **Top,
                           int Fed, int Qid,
                           struct Event_Message *Add )
/*-----EndDocHead-----*/
{
/* if this is called for IO it is out put queue entry */
int    FedNodeId, IdQ, NotInserted ;
struct Event_Message    *cPkt, *nPkt, *pPkt;
char  str[12];

cPkt = *Top ;

// if(cPkt !=NULL)fprintf(out," Add Ptime:%8.3f Top Ptime:%8.3f ", Add->Time.PhysicalTime, cPkt->Time.PhysicalTime );

NotInserted = 1 ;
FedNodeId = Fed ;
IdQ = Qid ;
Add->nqep = NULL ; /* just to be sure */
cPkt = *Top;
if ( QueEmpty > 0 ) {
    fprintf(out,"QueEmpty %3d Possibility Add_forFed %3d %8.3f Fed %2d IdQ %2d \n",
        QueEmpty, Add->Rti.federate_name, PhysicalTime, FedNodeId, IdQ );
    // gets(str);
}

if ( *Top == NULL || Add->Time.PhysicalTime < cPkt->Time.PhysicalTime ) {
    if ( *Top == NULL ) {
        *Top = Add ;
        //fprintf(out," Null add to %2d que %2d %8.8x %8.8x, %8.3f\n",
        //    FedNodeId, IdQ, Add, *Top, Add->Time.PhysicalTime );
    }
    else {
        Add->nqep = cPkt;
        *Top = Add ;
        // fprintf(out," Null add to %2d que %2d %8.8x %8.8x, %8.3f < %8.3f\n",
        //    FedNodeId, IdQ, Add, *Top, Add->Time.PhysicalTime, cPkt->Time.PhysicalTime );
    }
    Ql[FedNodeId ][IdQ ] += 1;
    Qn[FedNodeId ][IdQ ] += 1;
}
else {
    cPkt = *Top;
    do {
        nPkt = cPkt->nqep;
        //fprintf(out,"Enter s %8.3f %3d, %8.3f ",
        //    cPkt->Time.PhysicalTime,cPkt->Time.Label, Add->Time.PhysicalTime);
        // if ( nPkt != NULL ) { fprintf(out," %8.3f ", nPkt->Time.PhysicalTime ); }
        // fprintf(out,"\n");

        if ( nPkt != NULL &&
            Add->Time.PhysicalTime <= nPkt->Time.PhysicalTime &&
            cPkt->Time.PhysicalTime <= Add->Time.PhysicalTime ) {

            if ( cPkt->Time.PhysicalTime == Add->Time.PhysicalTime ) {
                if ( cPkt->Time.Label > 0 ) { /* do while == & insert */
                    do {
                        pPkt = cPkt ;
                        cPkt = cPkt->nqep ;
                        //if(cPkt!=NULL){
                        //    fprintf(out," Tim = Tim to %2d que %2d %8.3f\n", FedNodeId,
                        //        IdQ, cPkt->Time.Label, Add->Time.PhysicalTime);}
                    } while ( cPkt != NULL && /* cPkt != cPkt->nqep && */
                        cPkt->Time.PhysicalTime == Add->Time.PhysicalTime );

                    if ( cPkt == NULL ) { /* insert at end */

```

```

        Add->Time.Label = pPkt->Time.Label + 1 ;
        pPkt->nqep = Add ;
        NotInserted = 0 ;
        cPkt = Add ;
        //fprintf(out," cPkt == NULL to %2d que %2d \n", FedNodeId, IdQ);
    }
    else {
        Add->Time.Label = pPkt->Time.Label + 1 ;
        Add->nqep = pPkt->nqep ;
        pPkt->nqep = Add ;
        NotInserted = 0 ;
        //    fprintf(out," cPkt != NULL to %2d que %2d Label %1d %1d\n",
        //        FedNodeId, IdQ, pPkt->Time.Label, Add->Time.Label);
    }
}
else {
    cPkt->Time.Label = 1 ;
    Add->Time.Label = 2 ;
    Add->nqep = cPkt->nqep ;
    cPkt->nqep = Add ;
    NotInserted = 0 ;
    //fprintf(out," Label == 0    to %2d que %2d \n", FedNodeId, IdQ);
}
} /* end of c = add */
else if ( Add->Time.PhysicalTime ==  nPkt->Time.PhysicalTime ) {
    // fprintf(out," NextTime==    to %2d que %2d \n", FedNodeId, IdQ);
} /* end of stuff if equal */
else {
    Add->nqep = cPkt->nqep ;
    cPkt->nqep = Add ;
    NotInserted = 0 ;
    // fprintf(out," Time != Time to %2d que %2d \n", FedNodeId, IdQ);
}
}
else if ( nPkt == NULL) {
    if ( cPkt->Time.PhysicalTime ==  Add->Time.PhysicalTime ) {
        cPkt->Time.Label = 1 ;
        Add->Time.Label = 2 ;
        cPkt->nqep = Add ;
        NotInserted = 0 ;
        //fprintf(out," nPkt==NULL T to %2d que %2d \n", FedNodeId, IdQ);
    }
    else {
        cPkt->nqep = Add ;
        NotInserted = 0 ;
        //fprintf(out," nPkt == NULL to %2d que %2d \n", FedNodeId, IdQ);
    }
}
}

pPkt = cPkt ;
cPkt = cPkt->nqep ;
} while ( cPkt != NULL && NotInserted) ;

Qn[FedNodeId][IdQ ] += 1;
Ql[FedNodeId ][IdQ ] += 1;
// fprintf(stdout, "QueE at %8.3f Fed %2d  Que %2d      num %5d   Length %5d \n",
//    PhysicalTime, FedNodeId , IdQ, Qn[FedNodeId][IdQ ], Ql[FedNodeId ][IdQ ] );
/* fprintf( stderr, "Addmsg current %8.8x  picked %8.8x for %2d \n", cPkt, pPkt, i );*/
} /* else stuff on queue */

}
/*----- EVENT MANAGER ----- */
/*----- DocHeading */

```

30 June 1999

```

extern double EventManager(FILE *out, FILE *LgFile, int *AreQueuesEmpty,
                           struct Region_Node_Handle *RNH )

/*-----EndDocHead-----*/
{
    int    Fed, IdQ, FedNodeId, ColorId, IdOffset ;
    int    TSOMessage ;
    struct Federate_Destination *Dest ;
    struct Event_Message *Next, *aNewM ;
    // double dtemp;
    char str[24];

    ColorId = IdOffset = 0 ;
    PhysicalTime = NextEndTime( &Fed, &IdQ );

    *AreQueuesEmpty = 1 ;

    if(Fed < 0 || IdQ < 0) {
        fprintf(out, " Did not pick a next time => all service must != 0   Press Enter  \n");
        QueuesPrint(out, -7) ;
        // gets(str);
        *AreQueuesEmpty = 0 ;
    }
    Next = GetLowestTimeMessage( Fed, IdQ);

    if ( Next != NULL ) {
        // if ( FederateDelay[Ether] < PhysicalTime ) { /* Ether cannot be used earlier than now */
        //     FederateDelay[Ether] = PhysicalTime ;
        // }

        //printf("EM                      EMFed %2d IdQ %2d Sim %1d Id %2d Time %8.3f\n",
        // Fed, IdQ, Next->WhoGetsIt.SIM, Next->Time.UniqueMsgId, Next->Time.PhysicalTime );

        if ( IdQ == OutQ ) { /* need to distribute */
            Dest = Next->destinations_list;
            if ( Dest != NULL ) { /* should not be in the out que without destination */

                ColorRTIService( Next, &ColorId, &IdOffset ) ;
                if ( FederateDelay[Ether] < FederateDelay[Fed] ) {
                    FederateDelay[Ether] = FederateDelay[Fed] ;
                }
                EtherService = LATENCY ;
                do {

                    FedNodeId = Dest->federate ; /* these start at 1 one */
                    aNewM = c_Duplicate_Event_Message( Next ) ; /* time of Xfer */

                    //aNewM->Time.PhysicalTime = FederateDelay[Ether] + EtherService ;
                    if ( aNewM->Rti.rti_svc_nbr == RTI_TIME_ADV_RQST ) {
                        aNewM->Time.PhysicalTime = FederateDelay[Ether] + EtherService ;
                    }
                    else {
                        aNewM->Time.PhysicalTime = FederateDelay[Ether] + EtherService ;
                    }
                    aNewM->Time.OutService = aNewM->Time.PhysicalTime - EtherService ;
                    aNewM->Rti.federate_name = FedNodeId; /* FederateOffset */
                    aNewM->Time.OutComplete = aNewM->Time.PhysicalTime ;

                    /* Is this a TSO message ? all .SIM == 1 messages are */
                    /* even though the delivery is to the "InToRTI" Queue the .SIM must be set */
                    /* rememeber that in this file federates start 0 (zero) relative */
                    TSOMessage = TSOBoundMessage(Next) ;
                    if ( TSOMessage == 1 ){
                        if ( aNewM->Color.ColorTag ==
                            FederateCurrentColor[FedNodeId-1][FedColorIndx[FedNodeId-1]] ) {
                            /* the fed receiving for destination*/

```

30 June 1999

```

        FederateColorReceived[FedNodeId-1][FedColorIndx[FedNodeId-1] ] += 1 ;
        /* this fed sending */
    }
    else { /* not this color can be bad or previous color */
        if ( aNewM->Color.ColorTag <=
            FederateCurrentColor[FedNodeId-1][FedColorIndx[FedNodeId-1] ] ) {
            /* need to call function that handles straggler message */
            /* and send a message with a count of one for this destination */
            CreateRTIreport("Receive", aNewM->Color.ColorTag, 1,
                PhysicalTime +0.0004, FedNodeId );
            printf("CreateReport Federate %2d (+1) FederateColor not known %3d for
EVMgr service %4d \n", FedNodeId,
                aNewM->Color.ColorTag, aNewM->Rti.rti_svc_nbr);
        }
        else {
            printf("For Federate %2d (+1) FederateColor not known %3d \n", Fed,
                aNewM->Color.ColorTag );
        }
    }
}

LastEtherService = EtherService ;
/* time mgmt admin services get priority on the queues */
//fprintf(stdout,"%sAdd to OUT Queue RtiService number %5d \n",
// ".....", aNewM->Rti.rti_svc_nbr );
if ( aNewM->Rti.rti_svc_nbr == RTI_TIME_ADV_RQST ||
    aNewM->Rti.rti_svc_nbr == RTI_QUERY_FED_LBTS ||
    aNewM->Rti.rti_svc_nbr == RTI_RPTNG_FED_LBTS ||
    aNewM->Rti.rti_svc_nbr == RTI_RPTNG_RCV_LBTS ||
    aNewM->Rti.rti_svc_nbr == RTI_RPTNG_SND_LBTS ||
    aNewM->Rti.rti_svc_nbr == RTI_TIME_ADV_GRANT ) {
    fprintf(stdout,"%sAdd to Queue %2d RtiService number %5d for %8.5f \n",
        ".....",
        aNewM->Rti.federate_name, aNewM->Rti.rti_svc_nbr, aNewM->Time.PhysicalTime );
    AddPriorityEvent(out, "InToRTI", aNewM );
}
else {
    AddEvent(out, "InToRTI", aNewM ); /* always here first 11/24/98 agreed */
}

    Dest = Dest->next;
} while ( Dest != NULL );
EtherService = LATENCY ;
LastService[Ether] = EtherService ;
ResourceTime[Ether] += LastService[Ether] ;
FederateDelay[Ether] += LastService[Ether] ;

d_Federate_Destination(Next->destinations_list); /*destroy whole list of destinations */

Next->destinations_list = NULL;
d_Event_Message( Next);

}
else {
    fprintf( stdout,
        "DeleteMessageFromIOQueue-NoDestination, %d Rti %1d SIM %1d\n",
        Next->Rti.rti_svc_nbr, Next->WhoGetsIt.RTI,Next->WhoGetsIt.SIM);
    d_Event_Message( Next);
    Next = NULL ;
}
} /* if IO output */

/* Here Needs to be a piece of code that counts the send and received for the
for current color of the LBTS interval
*/

```

```

else if ( IdQ == RtiQ ) { /* This is just a move from the OutQ queue to the RTI Que */
    /* with the statistic of an io interrupt */
    Interrupts[Fed] += 1 ;
    //if ( FederateDelay[Fed] < FederateTime[Fed] ) {
    //    FederateDelay[Fed] = FederateTime[Fed];
    //}
    if ( Next->Rti.rti_svc_nbr == RTI_DISCVR_SETUP ) {
        // fprintf(stdout, "%sCall rtimgr_RTIEvent RtiService    %5d \n",
        //    ".....", Next->Rti.rti_svc_nbr );
    }
    //if ( Next->Rti.rti_svc_nbr == RTI_TIME_ADV_RQST ) {
    //    ColorRTIService( Next, &ColorId, &IdOffset );
    //}
#if ( DEMOONLY )
    LastService[Fed] = rtimgr_RTIEventTEST( Next ); /* Test driver of RTI Model */
#else
    LastService[Fed] = rtimgr_RTIEvent( Next ); /* RTI Model */
#endif
    ResourceTime[Fed] += LastService[Fed] ;

    if ( ColorId == 0 ) {
        ColorRTIService( Next, &ColorId, &IdOffset );
    }

    FederateDelay[Fed] += LastService[Fed] ;
    if ( Next->WhoGetsIt.RTI == 1 ) {
        ; /* do nothing */
    }
    else if ( Next->WhoGetsIt.SIM == 1 ) {
        //fprintf(stdout, "%sAdd to Tso Queue RtiService number %5d \n",
        //    ".....", Next->Rti.rti_svc_nbr );
        Next->Time.TsoRtComplete = FederateDelay[Fed] + LastService[Fed];
        Next->Time.PhysicalTime = Next->Time.VirtualTime ;
        Next->WhoGetsIt.SIM = 0 ;
        AddEvent( out, "TSO", Next );
    }
    else { /* delete the orig event */
        d_Event_Message( Next );
    }
}

/* if RTI MODEL */

else if ( IdQ == TsoQ ) { /* then this is the smallest increment to PhysicalTime */
    /* across all of the queues */

    if ( FederateDelay[Fed] < FederateTime[Fed] ) {
        FederateDelay[Fed] = FederateTime[Fed];
    }
    Next->Time.TsoService = FederateDelay[Fed] ;

#if ( DEMOONLY )
    LastService[Fed] = SimModelTEST( Next ); /* Test driver of Sim Unit Model */
#else
    LastService[Fed] = SimModel( Next, PhysicalTime, RNH ); /* Sim Unit Model */
#endif
/*
    if ( Next->Color.Boundary == 0 ) {
        if ( Next->Color.ColorTag & 0x00000001 ) {
            ColorId = 20 ; IdOffset = 6 ;
        }
        else {
            ColorId = 21 ; IdOffset = 8 ;
        }
    }
    else { ColorId = Next->Color.Boundary; IdOffset = 17 ;

```

```

    }
    */
    ResourceTime[Fed] += LastService[Fed] ;
    FederateDelay[Fed] += LastService[Fed] ;

    } /* if sim model */

//ResourceVerification( LgFile, Fed, IdQ );

if ( IdQ > 0 ) {
    XQueuesPrint( PhysicalTime, Fed, IdQ, ColorId, IdOffset ); /* XWINDOW QUEUE DISPLAY
    */
}
else {
    XQueuesPrint( PhysicalTime, Ether, IdQ, ColorId, IdOffset ); /* XWINDOW QUEUE DISPLAY
    */
}
LastFed = Fed;
LastQ = IdQ ;
} /* end of as long as Next != NULL */
/*
fprintf(out,"queue type phy time fed1 fed2 fed3 fed4 fed5 Enet cml1 cml2 cml3 cml4 cml5
Enet\n");
for (k=0; k<MaxQueues; k++) {
    fprintf(out,"EventQ%3d %9.6f: ", k, PhysicalTime );
    for (m=0; m<MaxResources; m++ ) { fprintf(out, "%5d", Ql[m][k]); }
    for (m=0; m<MaxResources; m++ ) { fprintf(out, "%5d", Qn[m][k]); }
    fprintf(out,"\n");
}
*/

return( PhysicalTime );
}
/*----- end of EVENT MANAGER ----- */
/*----- DocHeading ----- */

```

30 June 1999

extern int ColorRTIService(struct Event_Message *MsgPtr, int *ColorSel, int *LineOffset)

```

{
    int ColorId, IdOffset;
    ColorId = IdOffset = 0;

    if ( MsgPtr->Rti.rti_svc_nbr == RTI_TIME_ADV_RQST ) {
        ColorId = 1 ;
        IdOffset = 20 ;
        RTI_TIME_ADV_RQST_IdOffset = 20 ;
    }
    else if ( MsgPtr->Rti.rti_svc_nbr == RTI_TIME_ADV_GRANT ) {
        ColorId = 4; /* magenta */
        IdOffset = 20 + RTI_TIME_ADV_GRANT_IdOffset ;
        if ( RTI_TIME_ADV_GRANT_IdOffset == 2 ) { RTI_TIME_ADV_GRANT_IdOffset = 0 ; }
        else { RTI_TIME_ADV_GRANT_IdOffset = 2 ; }
    }
    else if ( MsgPtr->Rti.rti_svc_nbr == RTI_RPTNG_FED_LBTS ) {
        ColorId = 6; /* 5 gold, 11 brown */
        IdOffset = 10 + RTI_RPTNG_FED_LBTS_IdOffset ;
        if ( RTI_RPTNG_FED_LBTS_IdOffset == 2 ) { RTI_RPTNG_FED_LBTS_IdOffset = 0 ; }
        else { RTI_RPTNG_FED_LBTS_IdOffset = 2 ; }
    }
    else if ( MsgPtr->Rti.rti_svc_nbr == RTI_RPTNG_RCV_LBTS ) {
        ColorId = 6; /* 5 gold, 11 brown */
        IdOffset = 20 + RTI_RPTNG_RCV_LBTS_IdOffset ;
        if ( RTI_RPTNG_RCV_LBTS_IdOffset == 2 ) { RTI_RPTNG_RCV_LBTS_IdOffset = 0 ; }
        else { RTI_RPTNG_RCV_LBTS_IdOffset = 2 ; }
    }
    else if ( MsgPtr->Rti.rti_svc_nbr == RTI_RPTNG_SND_LBTS ) {
        ColorId = 1 ; /* 5 gold, 11 brown */
        IdOffset = 30 + RTI_RPTNG_SND_LBTS_IdOffset ;
        if ( RTI_RPTNG_SND_LBTS_IdOffset == 2 ) { RTI_RPTNG_SND_LBTS_IdOffset = 0 ; }
        else { RTI_RPTNG_SND_LBTS_IdOffset = 2 ; }
    }
    else if ( MsgPtr->Rti.rti_svc_nbr == RTI_QUERY_FED_LBTS ) {
        ColorId = 5 ; /* 6 peru, 2 green */
        IdOffset = 10 + RTI_QUERY_FED_LBTS_IdOffset ;
        if ( RTI_QUERY_FED_LBTS_IdOffset == 2 ) { RTI_QUERY_FED_LBTS_IdOffset = 0 ; }
        else { RTI_QUERY_FED_LBTS_IdOffset = 2 ; }
    }

    /* */
    else if (MsgPtr->Rti.rti_svc_nbr == RTI_UPDATE_ATTRIB ) {
        ColorSIMinRTI( MsgPtr, &ColorId, &IdOffset );
        IdOffset += WiggleColor_IdOffset ;
        if ( WiggleColor_IdOffset == 4 ) { WiggleColor_IdOffset = 0 ; }
        else { WiggleColor_IdOffset = 4 ; }
    }
    else if (MsgPtr->Rti.rti_svc_nbr == RTI_REFLECT_ATTRIB ) {
        ColorSIMinRTI( MsgPtr, &ColorId, &IdOffset );
        IdOffset += WiggleColor_IdOffset ;
        if ( WiggleColor_IdOffset == 4 ) { WiggleColor_IdOffset = 0 ; }
        else { WiggleColor_IdOffset = 4 ; }
    }
    else if (MsgPtr->Rti.rti_svc_nbr == RTI_SEND_INT ) {
        ColorSIMinRTI( MsgPtr, &ColorId, &IdOffset );
        IdOffset += WiggleColor_IdOffset ;
        if ( WiggleColor_IdOffset == 4 ) { WiggleColor_IdOffset = 0 ; }
        else { WiggleColor_IdOffset = 4 ; }
    }
    else if (MsgPtr->Rti.rti_svc_nbr == RTI_RECEIVE_INT ) {
        ColorSIMinRTI( MsgPtr, &ColorId, &IdOffset );
        IdOffset += WiggleColor_IdOffset ;
        if ( WiggleColor_IdOffset == 4 ) { WiggleColor_IdOffset = 0 ; }
        else { WiggleColor_IdOffset = 4 ; }
    }
}

```

30 June 1999

```

/*  */
    //else if ( Next->Rti.rti_svc_nbr == RTI_DISCVR_SETUP ) {
        // fprintf(stdout,"%sCall rtimgr_RTIEvent RtiService    %5d \n",
        //          ".....", Next->Rti.rti_svc_nbr );
    //}
    *ColorSel    = ColorId;
    *LineOffset = IdOffset ;
    return( ColorId );
}

/*----- end of  EVENT MANAGER ----- */
/*----- DocMethod */
extern int ColorSMinRTI( struct Event_Message *MsgPtr, int *ColorSel, int *LineOffset )
{
    int ColorId, IdOffset ;
    ColorId = IdOffset = 0 ;

    if ( MsgPtr->Color.Boundary == 0 && MsgPtr->WhoGetsIt.SIM == 1 ) {
        if ( MsgPtr->Color.ColorTag & 0x00000001 ) {
            ColorId = 20 ; IdOffset = 14 ;
        }
        else {
            ColorId = 21 ; IdOffset = 14 ;
        }
    }
    *ColorSel    = ColorId;
    *LineOffset = IdOffset ;
    return( ColorId );
}
/*----- NextEndTime NextEndTime NextEndTime f*/
/*----- DocHeading */

```



```

extern double NextEndTime(int *Federate, int *Que)
/*-----EndDocHead-----*/
{
    int i,j ;
    int      MinIOFed ;
    struct Event_Message *Next ;
    double   MinFedTime, MinEvent ;
    double   NextPhysicalTime, MinFedt ;
    char     str[12];
    int MinFedIdx, Congestion, IOCongestion ;

    MinFedTime = Largest_Number;
    MinFedt = NextPhysicalTime = Largest_Number;

    *Federate = -1;
    *Que      = -1;

    i = j = 0 ;

    /* congested => federate's queue is backed up with items
       that are older than the current phys time at that federate.
    */

    IOCongestion = Congestion = 0 ;

    /* loop thru Federates,
       - save off the queue nbr of the minimum phys time event found for all
         of each federate's queues
       - keep track of the federate with the minimum time event overall
    */
    MinFedIdx = -1 ;
    MinFedTime = Largest_Number;
    for (i=0; i< MaxFederates; i++ ) {
        FederateCongested[i] = -1 ;
        FederateQueue[i]     = -1 ;          /* no events for resource */

        if (FederateDelay[i] < MinFedTime) {
            MinFedTime = FederateDelay[i] ;
            MinFedIdx  = i ;
        }
        for ( j = 1; j < MaxQueues; j++ ) { /* Is a federate Q Congested ?*/
            MinNextEvent[i] = FederateDelay[i]; /* assume congested */
            Next = Qp[i][j] ;
            if ( Next != NULL ) {
                if ( MinNextEvent[i] >= Next->Time.PhysicalTime ) {
                    FederateCongested[i] = j ; /* remember Q */
                    Congestion = Congestion | ( 0x1 << i );
                    MinNextEvent[i] = Next->Time.PhysicalTime ; /* FederateDelay[i] */
                    FederateQueue[i] = j ; /* Add any priority selection here! */
                }
            }
        }
    } /* for queue */

    if ( FederateCongested[i] < 0 ) { /* not congested */
        MinNextEvent[i] = Largest_Number ;
        for ( j = 1; j < MaxQueues; j++ ) { /* Which Queue is lowest ?*/
            Next = Qp[i][j] ;
            if ( Next != NULL ) {
                if ( MinNextEvent[i] >= Next->Time.PhysicalTime ) {
                    FederateQueue[i] = j ;
                    MinNextEvent[i] = Next->Time.PhysicalTime ;
                }
            }
        }
    } /* for queue */
} /* Federate not congested */

```

30 June 1999

```

} /* end of for ALL MaxFederates; */

//MinNextEvent[i] - is either FederateDelay(congested)
//                  or      p->Time.PhysicalTime(not congested)

//,MinFedIdx,MinFedTime,FederateCongested[i],FederateQueue[i],MinNextEvent[i],Congestion

/* find the min phys time event for the ethernet (out) queues of each federate */

FederateCongested[Ether] = -1 ;
MinIOFed                = -1 ;
MinNextEvent[Ether]     = Largest_Number ;
if ( Congestion == 0 ) {
//printf(" NoCongestion for RTI & TSO queues \n");
}
if ( FederateDelay[Ether] <= MinFedTime ) { /* || Congestion == 0 ) {  Ether may go next */

    for (i=0; i< MaxFederates; i++ ) {
        Next = Qp[i][0] ; /* Only for the IO Cards in the LAN */
        if ( Next != NULL ) {
            if ( MinNextEvent[Ether] >= Next->Time.PhysicalTime || /* IO Card */
                Next->Time.PhysicalTime <= MinNextEvent[i] ) { /* Fed */

                //  MinNextEvent[Ether] = Next->Time.PhysicalTime;

                if ( FederateCongested[i] < 0 ) { /* Federate not congested */
                    /* But the IO card may be */
                    IOCongestion = IOCongestion | ( 0x1 << Ether );
                    if ( Next->Time.PhysicalTime <= FederateDelay[i] || /* IOCongestion*/
                        Next->Time.PhysicalTime <= MinNextEvent[i] ) {
                        if ( MinIOFed > -1 ) {
                            if ( MinNextEvent[Ether] > Next->Time.PhysicalTime ) {
                                MinNextEvent[Ether] = Next->Time.PhysicalTime; /* selected lowest IO
Card */
                                FederateCongested[i] = 0 ; /* remember Q */
                                MinIOFed = i ;
                            }
                        }
                        else { /* first Pick */
                            FederateCongested[i] = 0 ; /* remember Q since IO card is congested */
                            MinIOFed = i ;
                            MinNextEvent[Ether] = Next->Time.PhysicalTime; /* selected lowest IO
Card */
                        }
                    }
                }
            }
        }
        else { /* Fed is congested MinNextEvent[i] should == FederateDelay[i]*/
            if ( Next->Time.PhysicalTime <= FederateDelay[i] ||
                Next->Time.PhysicalTime <= MinNextEvent[i] ) {
                if ( MinIOFed > -1 ) {
                    if ( MinNextEvent[Ether] > Next->Time.PhysicalTime ) {
                        MinNextEvent[Ether] = Next->Time.PhysicalTime; /* selected lowest IO
Card */
                        MinIOFed = i ;
                    }
                }
                else { /* first Pick */
                    MinNextEvent[Ether] = Next->Time.PhysicalTime; /* selected lowest IO
Card */
                    MinIOFed = i ;
                }
            }
        }
    }
} /* Queue time is less */
} /* end of for MaxFederates; */
}

```

```

if ( MinIOFed >= 0 ) { /* then ether net is next */
    *Federate = MinIOFed ;
    *Que      = 0 ;
    if ( IOCongestion > 0 ) {
        NextPhysicalTime = FederateDelay[Ether]; /* FederateDelay[MinIOFed] ; */
        MinFedt          = FederateDelay[Ether]; /* FederateDelay[MinIOFed] ; */
    }
    else { /* not congested */
        Next = Qp[MinIOFed][0] ;
        NextPhysicalTime = Next->Time.PhysicalTime ;
        MinFedt          = Next->Time.PhysicalTime ;
    }
    for ( i=0; i< MaxFederates; i++ ) {
        if ( MinFedt > FederateDelay[i] && FederateCongested[i] < 0 ) {
            FederateDelay[i] = MinFedt ; /* advance system time */
        }
    }
    if ( FederateDelay[Ether] < NextPhysicalTime ) {
        FederateDelay[Ether] = NextPhysicalTime;
    }
}
else if ( Congestion == 0 ) { /* Minimum Federate Event */
    // printf("                                not Congested \n");
    MinFedt = Largest_Number;
    for ( i=0; i< MaxFederates; i++ ) {
        if ( MinNextEvent[i] <= MinFedt ) {
            MinFedt = MinNextEvent[i] ; /* whichever federate in less */
            MinFedIdx = i ;
        }
    }
    if ( FederateDelay[Ether] > MinFedt ) { MinFedt = FederateDelay[Ether] ; }
    *Federate = MinFedIdx ;
    *Que      = FederateQueue[MinFedIdx] ;
    for ( i=0; i< MaxFederates ; i++ ) {
        FederateDelay[i] = MinFedt ; /* advance system time */
    }
    NextPhysicalTime = MinFedt ;
}
else if ( Congestion > 0 ) { /* the minimum congested federate */
    MinFedt = Largest_Number;
    for ( i=0; i< MaxFederates; i++ ) {
        if ( FederateCongested[i] > 0 && FederateDelay[i] < MinFedt ) {
            MinFedt = FederateDelay[i] ; /* whichever federate in less */
            MinFedIdx = i ;
        }
        else { /* not congested */
            if ( MinNextEvent[i] <= MinFedt ) {
                MinFedt = MinNextEvent[i] ; /* Next event is less */
                MinFedIdx = i ;
            }
        }
    }
}
*Federate = MinFedIdx ;
*Que      = FederateQueue[MinFedIdx] ;
for ( i=0; i< MaxFederates; i++ ) {
    if ( MinFedt > FederateDelay[i] ) { /* && FederateCongested[i] < 0 ) { */
        FederateDelay[i] = MinFedt ; /* advance system time */
    }
}
NextPhysicalTime = MinFedt ;
}

if ( NextPhysicalTime < 0.000001 ) {
    printf("Physicaltime small\n");
}

```

```

for ( i=0; i<MaxResources; i++) { FederateTime[i] = FederateDelay[i] ; }
/* */
//fprintf(stdout,
//"\nQUEUESTATES   %8.3f      %8.3f      %8.3f MinIOFed %2d  MinFedIdx, %1d MinFedTime, %8.3f
Congest IO %2.2x fed %2.2x\n",
//  NextPhysicalTime, PhysicalTime,  MinFedt, MinIOFed,
//                      MinFedIdx,  MinFedTime,  IOCongestion,  Congestion ) ;
//for ( i=0; i<MaxResources; i++) {
// fprintf(stdout,"QUEUESTATE %2d: t %8.3f Avail %8.3f  Min %8.3f  Dif %8.3f Que   Fq %2d Cg %2d
*F %2d *Q %2d\n", i,
//  FederateTime[i], FederateDelay[i], MinNextEvent[i], (MinNextEvent[i]-FederateDelay[i]),
//  FederateQueue[i],
//  FederateCongested[i], *Federate, *Que );
//}
//QueuesPrint(stdout,-7);

if ( NextPhysicalTime == Largest_Number) {
    NextPhysicalTime =  PhysicalTime ;
}
if ( StepThroughThisBeast > 0 ) {
    printf(" Stepping through  press Enter \n" );
    gets(str);
}
// if ( PhysicalTime > 0.4 ) { StepThroughThisBeast = 1 ; }
/* */

return( NextPhysicalTime );
}

/*-----  -----  GetLowestMessage  f*/
/*-----  DocHeading  */

```

30 June 1999

```

extern struct Event_Message *GetLowestTimeMessage( int Fed, int Que )
/*-----EndDocHead-----*/
{
    struct Event_Message *fPtr;
    char str[12];
    int i,j;

    fPtr = Qp[Fed][Que] ;
    /* if null then all queues are empty! */ /* Add to History Queue */

    if ( fPtr != NULL ) {

        Qp[Fed][Que] = fPtr->nqep ;
        Ql[Fed][Que] -= 1 ;
        fPtr->nqep = NULL ;
        for ( i=0; i< MaxFederates; i++) {
            for ( j=0; j< MaxQueues; j++) {
                if ( Ql[i][j] > 0 && Qp[i][j] == NULL ) {
                    QueEmpty += 1;
                    fprintf(stdout,
                        "QueEmpty but not a ZERO count on QueLength G Fed %2d Q %2d at %8.3f   ???\n",
                            j, j, PhysicalTime );
                    // if ( QueEmpty < 2 ) { gets(str); }

                }
            }
        }

        // if ( Ql[Fed][Que] < 0 || ( Ql[Fed][Que] == 0 && Qp[Fed][Que] != NULL)) {
        //     printf(" Very Bad \n");
        //     QueuesPrint(stdout,-7);
        //     gets(str);
        // }
        if ( Qp[Fed][Que] == NULL && Ql[Fed][Que] > 0 ) {
            fprintf(stdout,
                "QueEmpty but not a ZERO count on QueLength Fed %2d Q %2d at %8.3f   ???\n",
                    Fed, Que, PhysicalTime );
            // gets(str);
        }
    }
    else { /* the selected queue was null */
        fprintf(stdout,
            "QueEmpty but was selected for the next event Fed %2d Q %2d at %8.3f   ???\n",
                Fed, Que, PhysicalTime );
        // gets(str);
    }

    /* -----
    fPtr->Marked_Delete = 1 ;

    HistOfQue[Hidx].Fed = Fed ;
    HistOfQue[Hidx].Que = Que ;
    HistOfQue[Hidx].EM = fPtr ;
    Hidx = (Hidx +1) % MaxHistory ;
    Lptr = HistOfQue[Hidx].EM ;
    HistOfQue[Hidx].EM = NULL;

    if ( Lptr != NULL ) {
        if ( Lptr->Marked_Delete > 0 && Lptr->Marked_Delete < 3) {
            printf(
                "\nDELETE EVENT MESSAGE HISTORY           %3d      Fed %2d      Q %2d      %8.3f ##
%4d\n",
                Hidx, HistOfQue[Hidx].Fed, HistOfQue[Hidx].Que, Lptr->Time.PhysicalTime,number_deleted
            );
            number_deleted +=1 ;
            d_Event_Message( Lptr );
        }
    }
}

```

```

        printf("Delete from History  Press Enter\n");
        // gets(str);
    }
    else { Lptr->Marked_Delete = -1 ;
    }
}
}
-----
*/

return( fPtr );
} /* Event_Message *GetLowestTimeMessage( int Fed, int Que ) */

/*----- CntMsg f*/
/*----- DocMethod */
extern int CntMsg( struct Event_Message *Top )
{
    struct Event_Message *cPkt ;
    int i ;

    i = 0 ;
    cPkt = Top ;
    if ( Top == NULL ) { i = 0; }
    else {
        cPkt = Top;
        while ( cPkt != NULL ) {
            i += 1;
            cPkt = cPkt->nqep ;
            /* fprintf( stderr, "Cntmsg current %8.8x  for %2d \n", cPkt, i );*/
        }
    }
    return(i); /* returns the length of the queue */
}

/*----- SelectMsg f*/
/*----- DocMethod */
extern struct Event_Message *SelectMsg( struct Event_Message **Top) /* */
{
    return( *Top );
}

/*----- EventManager.c ----- f*/
/*----- DocHeading */

```

30 June 1999

```

extern void QueuesInitialize()
/*-----EndDocHead---*/
{
    int i,j,k;

    PhysicalTime = 0.0;    /* System Current Time */
    PreviousPhysicalTime = 0.0 ;

    for(i=0; i<MaxResources; i++) {
        for (k=0; k<2; k++) {
            FederationColorTag = 1;
            FedColorIndx[i] = 1 ;
            if ( k == FedColorIndx[i] ) {
                FederateCurrentColor[i][k] = FederationColorTag ; }
            else { FederateCurrentColor[i][k] = 0 ; }
            FederateColorReceived[i][k] = 0 ;
            FederateColorSent[i][k] = 0 ;
        }
        for(j=0; j<MaxQueues; j++) {
            Qp[i][j] = NULL ;
            Ql[i][j] = 0 ;
            Qn[i][j] = 0 ;
        }
        for(j=0; j<StateCol ; j++) {
            StateTime[i][j]= 0.0 ;
            StateTpre[i][j]= 0.0 ;
            StateQend[i][j]= 0.0 ;
        }
        // LogicalTime[i] = 0.0;
        Interrupts[i] = 0.0;
        FederateTime[i] = 0.0;
        ResourceTime[i] = 0.0;
        FederateDelay[i] = 0.0;
        // FederateMinEvent[i] = 0.0;
        LastService[i] = 0.0;
    }
    PhysicalTime = 0.0;
    VirtualTime = 0.0;
    Hidx = 0 ;
    for( i = 0; i<MaxHistory; i++) {
        HistOfQue[i].Fed = -1 ;
        HistOfQue[i].Que = -1 ;
        HistOfQue[i].EM = NULL ;
    }
}/* end of QueuesInitialize() */
/*----- EventManager.c ----- f*/
/*----- DocHeading */

```

30 June 1999

extern void PrintQueueHistory(FILE *out)

```

/*-----EndDocHead---*/
{
    int i,j;
    struct Event_Message *Eptr ;
    i= ((Hidx+2) % MaxHistory);
    j=2;
    fprintf(out,"idx:H Fed, Q, Fed, Ll, PT, VT, dif(V-P), OrgId \n");
    do {
        Eptr = HistOfQue[i].EM ;
        if (Eptr!= NULL) {
            fprintf(out,"%3d:H %3d, %1d, ", i, HistOfQue[i].Fed, HistOfQue[i].Que);
            fprintf(out,"%3d, %1d, %8.3f, %8.3f, %8.3f, %5d,\n", Eptr->Rti.federate_name,
                Eptr->Time.Label, Eptr->Time.PhysicalTime,
                Eptr->Time.VirtualTime,
                (Eptr->Time.VirtualTime - Eptr->Time.PhysicalTime),
                Eptr->Time.UniqueMsgId) ;
        }
        j +=1 ;
        i = ((Hidx+j) % MaxHistory);
    } while ( i != ((Hidx+1) % MaxHistory) );
}

```

```

/*----- EventManager.c ----- f*/
/*----- DocHeading */

```



```
extern double QueueEnd( struct Event_Message *Pptr )
/*-----EndDocHead---*/
{
    int i;
    struct Event_Message *Eptr;
    double dtemp;
    Eptr = Pptr ;
    i=0;
    dtemp = 0.0;

    while ( Eptr != NULL ) { /*  && Eptr != Eptr->nqep ) { */
        dtemp = Eptr->Time.PhysicalTime ;
        //      if ( i > 10 ) {
        //          printf(" in loop \n");
        //          gets(str);
        //      }
        i +=1;
        Eptr = Eptr->nqep ;
    }
    return( dtemp );
}

/*-----DocHeading */
```

30 June 1999

```

extern int QueueCompare( struct Event_Message *Testptr, struct Event_Message *Pptr )
/*-----EndDocHead---*/
{
    int i, Match ;
    struct Event_Message *Eptr;
    //double dtemp;
    Eptr = Pptr ;
    i = Match = 0;
    //dtemp = 0.0;

    while ( Eptr != NULL && Match == 0 ) { /* && Eptr != Eptr->nqep ) { */
        //dtemp = Eptr->Time.PhysicalTime ;
        if (Testptr == Eptr ) {
            Match = 1 ;
        }
        i += 1;
        Eptr = Eptr->nqep ;
    }
    if ( Match > 0 ) { return( i ); }
    else { return( 0 ); }
}
/*----- EventManager.c ----- f*/
/*----- DocMethod */

extern int TestForDuplicate( struct Event_Message *Testptr )
{
    int i,j , t;
    int Fed, Que, Depth;
    char str[12];
    struct Event_Message *ptr ;
    Fed = Que = Depth = -1 ;
    for(i=0; i<MaxResources; i++) {
        for(j=0; j<MaxQueues; j++) {
            ptr = Qp[i][j];
            if ( ptr != NULL ) {
                t = QueueCompare( Testptr, ptr);
                if ( t > 0 ) {
                    Fed = i ; Que = j ; Depth = t ;
                    printf(" duplicate pointer found Resource(FED) %2d Que %2d depth %3d %s\n",
                        Fed, Que, Depth, "Press Enter To Continue" );
                    gets(str);
                }
            }
        }
    }
    return( Depth );
}
/*----- DocMethod */

extern int QueueColorCompare( unsigned int ColorTag, struct Event_Message *Pptr )
/*-----EndDocHead---*/
{
    /* Compare for not equal */
    int i, Match ;
    struct Event_Message *Eptr;
    //double dtemp;
    Eptr = Pptr ;
    i = Match = 0;
    //dtemp = 0.0;

    while ( Eptr != NULL ) { /* && Eptr != Eptr->nqep ) { */
        // dtemp = Eptr->Time.PhysicalTime ;
        if (ColorTag != Eptr->Color.ColorTag ) {
            Match = 1 ;
            fprintf( stdout,
                ".....At %9.4f %sColorTagCurrent %3d Differ From %3d sched for %9.4f Bdry %2d Fed
                %2d OrFed %2d Uniq %4d srv %4d\n",
                PhysicalTime, "...",

```

30 June 1999

```

//      ColorTag, Eptr->Color.ColorTag, Eptr->Time.PhysicalTime,
//      Eptr->Color.Boundary, Eptr->Rti.federate_name,
//      Eptr->Rti.origin_fed_name, Eptr->Time.UniqueMsgId,
//      Eptr->Rti.rti_svc_nbr );
//      //gets(str);
//      }
//      i += 1;
//      Eptr = Eptr->nqep ;
//      }
//      if ( Match > 0 ) { return( i ); }
//      else { return( 0 ); }
//      }
/*----- EventManager.c ----- f*/
/*----- DocMethod */

extern int TestForDiffColor( unsigned int ColorTag)
{
int i,j , t;
int Fed, Que, Depth;
char str[12];
struct Event_Message *ptr ;

Fed = Que = Depth = -1 ;
for(i=0; i<MaxResources; i++) {
for(j=0; j<MaxQueues; j++) {
ptr = Qp[i][j];
if ( ptr != NULL) {
t = QueueColorCompare( ColorTag, ptr);
if ( t > 0 && Que == 0 ) {
Fed = i ; Que = j ; Depth = t ;
fprintf(stdout,
"Fd%2d Q%2d At %9.4f %sColorTagCurrent %3d MismatchedColor found Resource(FED) %2d
Que %2d depth %3d %s\n",
Fed, Que, PhysicalTime, ".....", ColorTag,
Fed, Que, Depth, "Press Enter " );
// gets(str);
}
}
}
}

return( Depth );
}
/*----- EventManager.c ----- f*/
/*----- DocHeading */

```

30 June 1999

extern void PrintQueue(FILE *out, struct Event_Message *Pptr, char *Emark)

/*-----EndDocHead---*/

```

{
    int i;
    struct Event_Message *Eptr;
    Eptr = Pptr ;
    fprintf(out,"idx:Q Fed, Ll, PT, VT, dif(V-P), OrgId %s\n",Emark);
    i=0;
    while ( Eptr != NULL && i < 16 ) {
        fprintf(out,"%3d, %3d, %1d, %8.3f, %8.3f, %8.3f, %5d, %s \n", i, Eptr->Rti.federate_name,
            Eptr->Time.Label, Eptr->Time.PhysicalTime,
            Eptr->Time.VirtualTime,
            (Eptr->Time.VirtualTime - Eptr->Time.PhysicalTime),
            Eptr->Time.UniqueMsgId, Emark) ;
        i +=1;
        Eptr = Eptr->nqep ;
    }
}

```

```

/*----- EventManager.c ----- f*/
/*----- DocHeading */

```

```
extern void QueuesTest()
```

```
/*-----EndDocHead-----*/
```

```
{
    int i,j;
    double dtemp, kdbl ;
    struct Event_Message *ptr ;
    kdbl = 0.0;
    for(i=0; i<MaxFederates; i++) {
        for(j=0; j<MaxQueues; j++) {
            Qn[i][j] = 0 ;
            Ql[i][j] = 0 ;
            dtemp = kdbl ;
            kdbl += .1 ;
            /* if ( j == 0 ) {
                ptr = SetEventMessage(1, i, 1, 1, dtemp, dtemp, "QT");
                ptr->destinations_list = c_Federate_Destination("QT") ;
                ptr->destinations_list->next = c_Federate_Destination("QT") ;
                ptr->destinations_list->next->next = c_Federate_Destination("QT") ;
                ptr->destinations_list->next->next->next = c_Federate_Destination("QT") ;
                ptr->destinations_list->federate = (i+1) % MaxFederates ;
                ptr->destinations_list->next->federate = (i+1) % MaxFederates ;
                ptr->destinations_list->next->next->federate = (i+1) % MaxFederates ;
                ptr->destinations_list->next->next->next->federate = (i+1) % MaxFederates ;
                AddEvent( stdout, "Out" , ptr );
            }
        }
    }
    /* ptr = SetEventMessage(1, 0, 1,0, 1.0, 2.0, "QT"); AddEvent( stdout, "InToRTI" , ptr );
    ptr = SetEventMessage(1, 1, 1,0, 1.0, 2.0, "QT"); AddEvent( stdout, "InToRTI" , ptr );
    ptr = SetEventMessage(1, 2, 1,0, 1.0, 2.0, "QT"); AddEvent( stdout, "InToRTI" , ptr );
    ptr = SetEventMessage(1, 3, 1,0, 1.0, 2.0, "QT"); AddEvent( stdout, "InToRTI" , ptr );
    ptr = SetEventMessage(1, 0, 1,0, 1.0, 2.0, "QT"); AddEvent( stdout, "InToRTI" , ptr );
    ptr = SetEventMessage(1, 1, 1,0, 2.0, 4.0, "QT"); AddEvent( stdout, "InToRTI" , ptr );
    ptr = SetEventMessage(1, 1, 1,0, 1.5, 3.0, "QT"); AddEvent( stdout, "InToRTI" , ptr );
    ptr = SetEventMessage(1, 0, 1,0, 0.7, 1.1, "QT"); AddEvent( stdout, "InToRTI" , ptr );
    ptr = SetEventMessage(1, 0, 1,0, 1.5, 2.5, "QT"); AddEvent( stdout, "InToRTI" , ptr );
    ptr = SetEventMessage(1, 1, 1,1, 2.0, 4.0, "QT"); AddEvent( stdout, "InToRTI" , ptr );
    ptr = SetEventMessage(1, 1, 1,1, 1.5, 3.0, "QT"); AddEvent( stdout, "InToRTI" , ptr );
    ptr = SetEventMessage(1, 1, 1,1, 1.4, 2.5, "QT"); AddEvent( stdout, "InToRTI" , ptr );
    */
    for( j=0; j<MaxFederates; j++) {
        ptr = SetEventMessage(1, j, 0,1, 0.05, 0.1, "QT"); AddEvent( stdout, "TSO" , ptr );
        ptr = SetEventMessage(1, j, 0,1, 0.051, 0.1, "QT"); AddEvent( stdout, "TSO" , ptr );
        ptr = SetEventMessage(1, j, 0,1, 0.052, 0.1, "QT"); AddEvent( stdout, "TSO" , ptr );
        ptr = SetEventMessage(1, j, 0,1, 0.053, 0.1, "QT"); AddEvent( stdout, "TSO" , ptr );
        ptr = SetEventMessage(1, j, 0,1, 0.054, 0.1, "QT"); AddEvent( stdout, "TSO" , ptr );
        ptr = SetEventMessage(1, j, 0,1, 0.055, 0.1, "QT"); AddEvent( stdout, "TSO" , ptr );
        ptr = SetEventMessage(1, j, 0,1, 0.056, 0.1, "QT"); AddEvent( stdout, "TSO" , ptr );
        ptr = SetEventMessage(1, j, 0,1, 0.057, 0.1, "QT"); AddEvent( stdout, "TSO" , ptr );
        ptr = SetEventMessage(1, j, 0,1, 0.058, 0.1, "QT"); AddEvent( stdout, "TSO" , ptr );
        ptr = SetEventMessage(1, j, 0,1, 0.059, 0.1, "QT"); AddEvent( stdout, "TSO" , ptr );
    }
}
```

```
PrintQueue(stdout, Qp[0][TsoQ], "begin" );
```

```
} /* end of QueuesInitialize() () () () () () */
```

```
/*----- EventManager.c ----- f*/
```

```
/*----- DocMethod */
```

```
extern void PrintEventsInSystem( FILE *out)
```

```
{
    int k,m ;
    for (k=0; k<MaxQueues; k++) {
        if ( k == 0 ) fprintf(out, "OutQ %9.6f:", PhysicalTime );
```

```

if ( k == 1 ) fprintf(out," RTIQ %9.6f:", PhysicalTime );
if ( k == 2 ) fprintf(out," TsoQ %9.6f:", PhysicalTime );
for (m=0; m<MaxResources; m++ ) { fprintf(out, "%5d", Ql[m][k]); }
}
fprintf(out,"\n");
} /* end of short PrintEventsInSystem( */

/*----- EventManager.c ----- f*/
/*----- DocMethod */
extern void PrintEventsProcessed( FILE *out)
{
int k,m ;
for (k=0; k<MaxQueues; k++) {
if ( k == 0 ) fprintf(out,"OutQ %9.6f:", PhysicalTime );
if ( k == 1 ) fprintf(out," RTIQ %9.6f:", PhysicalTime );
if ( k == 2 ) fprintf(out," TsoQ %9.6f:", PhysicalTime );
for (m=0; m<MaxResources; m++ ) { fprintf(out, "%5d", Qn[m][k]); }
}
fprintf(out,"\n");
} /* end of short PrintEventsInSystem( */

/*----- EventManager.c ----- f*/
/*----- DocHeading */

```

30 June 1999

```

extern void XQueuesPrint(double PTime, int Fed, int Queue, int Tag, int Offset )
/*-----EndDocHead-----*/
{
    int j;
    char str[12];

    if ( PhysicalTime + 0.003 > XBaseTime + XIntervalTime ) {

        fprintf(stdout,
            "+++:, %8.3f XQueuesDisplayed  RESET DISPLAY XXXXXXXXXXXXXXXXXXXX \n",
            PhysicalTime );

        //printf("press enter for next interval \n" );
        if ( PhysicalTime > STARTDISPLAYINTERVAL ) {
            QueuesPrint(stdout,-7);
            // printf("press enter for next interval \n" );
            gets(str);
        }

        LalaClear();
        XBaseTime = PhysicalTime ;      /* XIntervalTime; */
    }

    if ( Fed >= 0 ){

        j = (Queue+1);

        if (Fed == Ether) {
            j = 8 ;

            LalaTimeQueue( Fed, j,
                (FederateDelay[Fed]-LastService[Fed]),
                (FederateDelay[Fed]), XBaseTime, XIntervalTime, Tag, Offset );
            // fprintf(stdout,
            //     "XQueR %2d at  %8.3f Fed %8.3f to %8.3f service %8.3f\n",
            //     Fed, PhysicalTime, (FederateDelay[Fed]-LastService[Fed]),
            //     FederateDelay[Fed], LastService[Fed] );
        }
        else {

            LalaTimeQueue( Fed, j,
                (FederateDelay[Fed]-LastService[Fed]),
                (FederateDelay[Fed]), XBaseTime, XIntervalTime, Tag, Offset );
            // fprintf(stdout,
            //     "XQueR %2d at  %8.3f Fed %8.3f to %8.3f service %8.3f\n",
            //     Fed, PhysicalTime, (FederateDelay[Fed]-LastService[Fed]),
            //     FederateDelay[Fed], LastService[Fed] );
        }
    }

}

} /* end of XQueuesPrint() */

/*-----DocMethod */
extern void SetBaseResourceTime()
/*-----EndDocHead-----*/
{
    int i;
    for( i = 0; i<MaxResources; i++) {
        BaseResourceTime[i] = ResourceTime[i] ;
        LastResourceTime[i] = ResourceTime[i] ;
    }
}
/*-----DocMethod */

```

```
extern void PrintUtilizationResourceTime( FILE *out, double interval, int Replicate )
{
    int i;

    fprintf(out, "U,%3d,%10.5f,", Replicate, PhysicalTime );

    for( i = 0; i<MaxResources; i++) {
        fprintf(out, "%9.6g,", (ResourceTime[i]-LastResourceTime[i])/interval );
        LastResourceTime[i] = ResourceTime[i] ;
    }
    fprintf(out, "\n");
}
/*-----EndDocHead---*/
/*-----DocHeading */
```



```
extern void QueuesPrint(FILE *out, int Replicate)
```

```
/*-----EndDocHead-----*/
{
    int i,j,k;
    struct Event_Message *ptr ;
    double lowQueue, lowDiff, lowFed ;
    int    lqF, lqQ, ldF, ldQ, lF ;

    //fprintf(out, " Federate, Delay, Resource, Service \n");
    fprintf(out, "+++, Physical time: %8.3f QueuesDisplayed\n",
        PhysicalTime );
    /*fprintf(out, "---, %8.3f, %8.3f, %8.3f, %8.3f EtherDelay \n",
        EtherNetTime, EtherNetDelay, EtherNetResourceTime, LastEtherService );
    */
    fprintf(out, "Rep,Fed#, Cg, Time, Physic, ResT, Serv, OUTq, offset, items,
    RTIq, offset, items, SIMq, offset, items, Fed#,\n");
    lowFed = lowQueue = lowDiff = Largest_Number ;
    lF = lqF = lqQ = ldF = ldQ = 0;
    for(i=0; i<MaxResources; i++) {
        if ( lowFed > FederateDelay[i] ) { lowFed = FederateDelay[i]; lF = i; }
        for(j=0; j<MaxQueues; j++) {
            ptr = Qp[i][j];
            if ( ptr != NULL) {
                if ( ptr->Time.PhysicalTime < lowQueue ) {
                    lqF = i; lqQ = j ;
                    lowQueue = ptr->Time.PhysicalTime ;
                }
                if ( PhysicalTime - FederateDelay[i] < lowDiff ) {
                    ldF = i; ldQ = j ;
                    lowDiff = ptr->Time.PhysicalTime - FederateDelay[i] ;
                }
            }
        }
    }
    for(i=0; i<MaxResources; i++) {
        StateTime[i][0] = FederateTime[i] ;
        StateTime[i][1] = FederateDelay[i];
        StateTime[i][2] = ResourceTime[i] ;
        StateTime[i][3] = LastService[i] ;

        if ( i == lF ) {
            fprintf(out,"%3d,%2d:~%2d, %8.3f, %8.3f, %8.3f, %8.3f; ",Replicate, i, FederateCongested[i],
                FederateTime[i], FederateDelay[i], ResourceTime[i], LastService[i] );
        }
        else if ( i == LastFed ) {
            fprintf(out,"%3d,%2d:~%2d, %8.3f, %8.3f, %8.3f, %8.3f; ",Replicate,i, FederateCongested[i],
                FederateTime[i], FederateDelay[i], ResourceTime[i], LastService[i] );
        }
        else {
            fprintf(out,"%3d,%2d: %2d, %8.3f, %8.3f, %8.3f, %8.3f; ",Replicate,i, FederateCongested[i],
                FederateTime[i], FederateDelay[i], ResourceTime[i], LastService[i] );
        }
    }

    k = 4 ;
    for(j=0; j< MaxQueues; j++) {

        ptr = Qp[i][j];

        if ( ptr != NULL) {

            StateTime[i][k] = ptr->Time.PhysicalTime ;
            // StateQend[i][k] = QueueEnd(ptr); /* find the end of the QUEUE */
            k += 1 ;

            StateTime[i][k] = ptr->Time.PhysicalTime - FederateDelay[i] ;
            // StateQend[i][k] = StateQend[i][k-1] - ptr->Time.PhysicalTime ;
        }
    }
}
```

```

    k += 1 ;

    if ( i == lqF && j == lqQ && i == ldF && j == ldQ ) {
//      fprintf(out,"%8.3f<~%8.3f<>%4d, ", ptr->Time.PhysicalTime,
        fprintf(out,"%8.3f, %8.3f, %4d, ", ptr->Time.PhysicalTime,
            (ptr->Time.PhysicalTime - FederateDelay[i] ), Ql[i][j]);
    }
    else if ( i == lqF && j == lqQ ) {
//      fprintf(out,"%8.3f<~%8.3f, %4d, ", ptr->Time.PhysicalTime,
        fprintf(out,"%8.3f, %8.3f, %4d, ", ptr->Time.PhysicalTime,
            (ptr->Time.PhysicalTime - FederateDelay[i] ), Ql[i][j]);
    }
    else if ( i == ldF && j == ldQ ) {
//      fprintf(out,"%8.3f, %8.3f<~%4d, ", ptr->Time.PhysicalTime,
        fprintf(out,"%8.3f, %8.3f, %4d, ", ptr->Time.PhysicalTime,
            (ptr->Time.PhysicalTime - FederateDelay[i] ), Ql[i][j]);
    }
    else {
        fprintf(out,"%8.3f, %8.3f %4d, ", ptr->Time.PhysicalTime,
            (ptr->Time.PhysicalTime - FederateDelay[i] ), Ql[i][j]);
    }
}
else {

    fprintf(out,"-----, -----, %4d, ", Ql[i][j] );
    StateTime[i][k] = StateTpre[i][k] ;
    StateQend[i][k] = 0.0 ;
    k += 1;
    StateTime[i][k] = StateTpre[i][k] ;
    StateQend[i][k] = 0.0 ;
    k += 1;
}
}
if ( i == lF ) { fprintf(out,"+%2d\n",i); }
else if ( i == LastFed ) { fprintf(out,"~%2d\n",i); }
else { fprintf(out," %2d\n",i); }
}
/*
fprintf(out, "deltas of previous \n");
for(i=0; i<MaxResources; i++) {
    fprintf(out, " %2d:", i );

    for(j=0; j<StateCol ; j++) {

        fprintf(out,"%8.3f, ", StateTpre[i][j] - StateTime[i][j] );
        if ( j == 5 || j == 7 || j == 9 ) {
            fprintf(out," , ");
        }
        StateTpre[i][j] = StateTime[i][j] ;
    }
    fprintf(out,"\n");
}
fprintf(out, "length of Physical time on queues \n");
for(i=0; i<MaxResources; i++) {
    fprintf(out, " %2d:", i );
    for(j=0; j<StateCol ; j++) {
        if ( j < 4 ) { fprintf(out," , "); }
        else { fprintf(out,"%8.3f, ", StateQend[i][j] ); }
        if ( j == 5 || j == 7 || j == 9 ) {
            fprintf(out," , ");
        }
    }
    fprintf(out,"\n");
}
}
*/
} /* end of QueuesPrint() */

```

30 June 1999

```
/*----- DocMethod */
extern void PrintFedTime(FILE *out, int Fed, struct Event_Message *ptr)
/*-----EndDocHead---*/
{
    fprintf(out, "Fed%2.2d: p%8.3f, f%8.3f, d%8.3f, r%8.3f, s%8.3f, ->%8.3f ->V%8.3f \n",
        Fed, PhysicalTime, FederateTime[Fed], FederateDelay[Fed],
        ResourceTime[Fed], LastService[Fed],
        ptr->Time.PhysicalTime, ptr->Time.VirtualTime );
}

/*----- EventManager.c ----- f*/
/*----- DocHeading */
```

30 June 1999

```

extern struct Event_Message *SetEventMessage( int Action, int Federate, /*EventManager.c */
      int RTIcommand, int SIMcommand, double lPhysicalTime,
      double lVirtualTime, char *sptr )
{
  struct Event_Message *tmp_ref ;
  tmp_ref = c_Event_Message(sptr);

  tmp_ref->Rti.rti_svc_nbr      = Action ; /* action to perform */
  tmp_ref->Rti.fedrtn_exname    = 7 * 70 ; /* name is number */
  tmp_ref->Rti.federate_name    = Federate ; /* name is number of node */
  tmp_ref->WhoGetsIt.RTI        = RTIcommand;
  tmp_ref->WhoGetsIt.SIM        = SIMcommand;
  tmp_ref->Color.ColorTag       = GetColorTag( Federate );
  tmp_ref->Time.PhysicalTime     = PhysicalTime + lPhysicalTime ;
  tmp_ref->Time.VirtualTime     = PhysicalTime + lVirtualTime ;
  tmp_ref->nqep                  = NULL;
  return (tmp_ref);
}

/*----- EventManager.c ----- f*/
/*----- DocHeading */

```

30 June 1999

```

extern struct Event_Message *SetExtendEventMessage(
    int RTIcommand,
    int SIMcommand,
    int Action,                /* rti_svc_nbr */
    int fedrtn_exname,
    int Federate,              /* federate_name */
    int obj_class_nbr,
    int obj_instance_nbr,
    int interact_class_nbr,
    int interact_instance_nbr ,
    double fedrtn_time ,
    int region_nbr,
    int routing_space_nbr ,
    int nbr_rcvd_msgs ,
    int nbr_sent_msgs ,
    double LBTS_time ,
    double lPhysicalTime,
    double lVirtualTime, char *sptr )
{
    struct Event_Message *tmp_ref ;
    tmp_ref = c_Event_Message(sptr);

    tmp_ref->Rti.rti_svc_nbr      = Action ; /* action to perform */
    tmp_ref->Rti.fedrtn_exname     = fedrtn_exname ; /* 7 * 70 ; name is number */
    tmp_ref->Rti.federate_name     = Federate ; /* name is number of node */
    tmp_ref->Rti.origin_fed_name   = Federate ; /* federate originating cmd */
    tmp_ref->Rti.obj_class_nbr     = obj_class_nbr ;
    tmp_ref->Rti.obj_instance_nbr  = obj_instance_nbr ;
    tmp_ref->Rti.interact_class_nbr = interact_class_nbr ;
    tmp_ref->Rti.interact_instance_nbr = interact_instance_nbr ;
    tmp_ref->Rti.fedrtn_time       = fedrtn_time ;
    tmp_ref->Rti.region_nbr        = region_nbr ;
    tmp_ref->Rti.routing_space_nbr  = routing_space_nbr ;
    tmp_ref->Rti.nbr_rcvd_msgs     = nbr_rcvd_msgs ;
    tmp_ref->Rti.nbr_sent_msgs     = nbr_sent_msgs ;
    tmp_ref->Rti.LBTS_time         = LBTS_time ;

    tmp_ref->WhoGetsIt.RTI         = RTIcommand;
    tmp_ref->WhoGetsIt.SIM         = SIMcommand;
    tmp_ref->Color.ColorTag        = GetColorTag( Federate );
    tmp_ref->Time.PhysicalTime     = lPhysicalTime ;
    tmp_ref->Time.VirtualTime      = lVirtualTime ;
    tmp_ref->nqep                  = NULL;
    return (tmp_ref);
}
/*-----*/
/*----- DocHeading */

```

30 June 1999

```

extern void ResourceVerification(FILE *out, int Fed, int Que)
{
    int i, j ;
    struct Event_Message *ptr;

    fprintf(out, "%6.3f: ", PhysicalTime );

    for(i=0; i<MaxResources; i++) {
        if ( i == Fed ) { fprintf(out, "%6.3f; ", FederateDelay[i]); }
        else {             fprintf(out, "%6.3f; ", FederateDelay[i]); }

        for(j=0; j<MaxQueues; j++) {
            if ( i != MaxResources-1 && ( j < 1 || i < MaxResources -1 ) ) {
                ptr = Qp[i][j];
                if ( ptr != NULL && i == Fed && j==Que) {
                    fprintf(out, "%6.3f, ", -1*ptr->Time.PhysicalTime );
                }
                else if ( ptr != NULL ) {
                    fprintf(out, "%6.3f, ", ptr->Time.PhysicalTime );
                }
                else { fprintf(out, "-----, " );
                }
            }
        }
    }
    fprintf(out, "\n");
    fprintf(out, "      : ", );

    for(i=0; i<MaxResources; i++) {
        fprintf(out, "      , ");
        for(j=0; j<MaxQueues; j++) {
            if ( i != MaxResources-1 && ( j < 1 || i < MaxResources -1 ) ) {
                ptr = Qp[i][j];
                if ( ptr != NULL ) { ptr = ptr->nqep ; }
                if ( ptr != NULL && i == Fed && j==Que) {
                    fprintf(out, "%6.3f, ", -1*ptr->Time.PhysicalTime );
                }
                else if ( ptr != NULL ) {
                    fprintf(out, "%6.3f, ", ptr->Time.PhysicalTime );
                }
                else { fprintf(out, "-----, " );
                }
            }
        }
    }
    fprintf(out, "\n");
    //for(i=0; i<MaxResources; i++) {
    //    fprintf(out, "%6.3f, ", LastService[i] );
    // }
    //PreviousValues[0] = PhysicalTime ;
    //for(i=0; i<MaxResources; i++) {
    //    PreviousValues[i+1] = FederateDelay[i];
    // }
    //fprintf(out, ": %6.3f\n", PreviousValues[0] );
}

/*----- EventManager.c ----- f*/
/*----- DocHeading */

```

```
extern double rtimgr_RTleventTEST( struct Event_Message *ptr)
{
    struct Event_Message *pref ;
    int i,j,k, m;
    double Service, NextTime ;

    m = PickSome(0, (MaxFederates-1) ); /* number to receive */
    m = 1 ;
    //Service = m*RTIServiceBase * erand48( EMX) + RTIServiceBase ;
    Service = RTIServiceBase * erand48( EMX) + RTIServiceBase/2.0 ;

    if ( ptr->WhoGetsIt.SIM == 1 ) {

        m = PickSome(0, 3 ); /* number to receive */
        i = PickSome(0, (MaxFederates-1) );
        j = PickSome(0, (MaxFederates-1) );
        k = PickSome(0, (MaxFederates-1) );
        k = PickSome(0, 1 );
        // if (k) { k = PickSome(0, 1 ) ; }

        NextTime = FederateDelay[i] + Service/2.0 ;

        pref = SetEventMessage(1, i, 0, k, NextTime, NextTime, "rtimgr"); /* just data 0, 0 */
        fprintf(stdout,
            "\nAdd to Fed %2d, %2d, %2d to Out queue as    Service %8.3f Next %8.3f CurFed %8.3f, PhyTim
            %8.3f\n",
                i, j, k,
                Service, NextTime, FederateDelay[i], PhysicalTime);

        if ( m > 0 ) {
            pref->destinations_list          = c_Federate_Destination("QT") ;
            pref->destinations_list->federate = i ; }
        if ( m > 1 ) {
            pref->destinations_list->next      = c_Federate_Destination("QT") ;
            pref->destinations_list->next->federate = j ; }
        if ( m > 2 ) {
            pref->destinations_list->next->next = c_Federate_Destination("QT") ;
            pref->destinations_list->next->next->federate = k ; }

        AddEvent( stdout, "Out" , pref );
    }
    else {
        //if ( ptr->Marked_Delete < 0 ){ /* && ptr->WhoGetsIt.SIM == 0 ) { */
        printf("DELETE EVENT MESSAGE  RtiMgr\n");
        // printf("Delete from RtiMgr  Press Enter\n"); gets(str);
        // }
        //else {
        // ptr->Marked_Delete = 1 ;
        //}
    }
    //d_Event_Message( ptr ); /* delete */

    return( Service );
}

/*----- EventManager.c ----- f*/
/*----- DocHeading */
```

30 June 1999

```

extern double SimModelTEST( struct Event_Message *Sptr)
{

    struct Event_Message *pref ;
    int i,j;
    double Service, NextTime ;

    Service = SIMServiceBase * erand48( EMX) + SIMServiceBase/2 ;
    //Service = SIMServiceBase + SIMServiceBase ;

    i = Sptr->Rti.federate_name - 1 ; /* FederateOffset */

    NextTime = PhysicalTime + ((SIMServiceBase/2) * erand48( EMX)) ;

    j = PickSome(0,1);

    if ( j && Sptr->WhoGetsIt.SIM == 1) { /* 0 for sim model should just mean data */
        pref = SetEventMessage(1, i, 0, 1, NextTime, NextTime, "SimMdl");
        AddEvent( stdout, "InToRTI", pref );
    }
    Sptr->Time.PhysicalTime += 1.0 ; /* fixed time interval */

    fprintf(stdout,
"\nAdd by Sim %2d, %2d, %2d to SIM queue as    Service %8.3f Next %8.3f CurFed %8.3f, PhyTim
%8.3f Nxt %8.3f %2d\n",
        i, i, i,
        Service, NextTime, FederateDelay[i], PhysicalTime, Sptr->Time.PhysicalTime,i);

    Sptr->Marked_Delete = 4 ; /* never delete */

    if ( Sptr->WhoGetsIt.SIM == 0 ) {
        d_Event_Message( Sptr ); /* delete event message data from external Federate*/
    }
    else { /* time step events continue by having the WhoGetsIt.SIM set*/
        AddEvent( stdout, "TSO", Sptr ) ;
    }
    return( Service );
}

/*----- EventManager.c ----- f*/
/*----- DocHeading */

```



```

extern void CreateRTIreport(char *Which, int ColorTag, int NumberOfDestinations,
                           double Time, int Federate ){
    struct Event_Message *NewPtr ;
    int    dest_nbr;

    /* This 'Federate' starts counting at 1 */

    if ( strcmp( Which, "Send") == 0 ) {
        NewPtr = SetExtendEventMessage(
            1,                                /* RTIcommand, */
            0,                                /* SIMcommand, */
            RTI_RPTNG_SND_LBTS,               /* Action, */
            1,                                /* fedrtn_exname,*/
            Federate,                          /* Federate */
            3,                                /* obj_class_nbr, */
            0,                                /* obj_instance_nbr,*/
            0,                                /* interact_class_nbr, */
            0,                                /* interact_instance_nbr */
            FederateDelay[Federate-1],         /* fedrtn_time ??? */
            0,                                /* region_nbr, */
            0,                                /* routing_space_nbr */
            0,                                /* nbr_rcvd_msgs */
            NumberOfDestinations,              /* nbr_sent_msgs */
            FederateDelay[Federate-1],         /* LBTS_time */
            Time,                             /* lPhysicalTime, */
            Time,                             /* lVirtualTime,*/
            "Init" );                         /* just a note */

        /* fill NewPtr->destinations list */
        dest_nbr = eventmgr_get_destination(Federate,NewPtr);
        /* when the dest is the federate, put the event on the local rti q*/
        if (dest_nbr == Federate)
            AddPriorityEvent( stdout, "InToRTI", NewPtr );
        else /* send the report to the parent */
            AddPriorityEvent( stdout, "Out", NewPtr );
    }

    if (strcmp( Which, "Receive") == 0 ) {
        NewPtr = SetExtendEventMessage(
            1,                                /* RTIcommand, */
            0,                                /* SIMcommand, */
            RTI_RPTNG_RCV_LBTS,               /* Action, */
            1,                                /* fedrtn_exname,*/
            Federate,                          /* Federate */
            0,                                /* obj_class_nbr, */
            0,                                /* obj_instance_nbr,*/
            0,                                /* interact_class_nbr, */
            0,                                /* interact_instance_nbr */
            FederateDelay[Federate-1],         /* fedrtn_time ????? */
            0,                                /* region_nbr, */
            0,                                /* routing_space_nbr */
            NumberOfDestinations,              /* nbr_rcvd_msgs */
            0,                                /* nbr_sent_msgs */
            FederateDelay[Federate-1],         /* LBTS_time */
            Time,                             /* lPhysicalTime, */
            Time,                             /* lVirtualTime,*/
            "Init" );                         /* just a note */

        /* fill NewPtr->destinations list */
        dest_nbr = eventmgr_get_destination(Federate,NewPtr);
        /* when the destination is the federate, put the event on the local rti q*/
        if (dest_nbr == Federate)
            AddPriorityEvent( stdout, "InToRTI", NewPtr );
        else
            AddPriorityEvent( stdout, "Out", NewPtr );
    }
} /* end CreateRTIreport(); */

```

30 June 1999

/*-----
/*-----

----- */
DocHeading */

```
/* file: FedNodes.c */
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <math.h>
#include "soa_defs.h"
#include "soa_cnst.h"
#include "proto.h"
#include "rti_services.h"
```

```
double PublishStart = 0.04 ;
double SubscribeStart = 0.045 ;
```

```
/*----- FedNodes.c ----- Create Federates f*/
```

```
/*-----*/
/*----- DocHeading */
```

```
extern void GetUnitsByCoTypeForRegion(FILE *out, int Category, /* FedNodes.c */
                                     char Side,
                                     struct Region_List *Regions,
                                     struct Region_Element_List *RegElements,
                                     struct Filter_Unit_List **CatTypeList)

/*-----EndDocHead---*/
{
    struct Filter_Unit_List *FilterList;
    struct Region_List *RegList ;

    RegList = Regions;
    //PrintOneRegion(out, Regions, "BeginofByTypeRegion\n" );

    if ( RegList->Category == Category ) { /* Warfighters or support */

        if ( RegElements != NULL ) {

            FilterList = PutCoInRegionInFilterList( RegElements );

            if ( *CatTypeList == NULL ) {
                *CatTypeList = FilterList ; }

            else {
                MergeFilterList( *CatTypeList, FilterList);
            }
        }
    }
    // PrintOneRegion(out, Regions, " END ofByTypeRegion\n" );
}

/*----- CREATE FEDERATE NODES and Assign Units */
/*----- DocHeading */
```

```

extern void CreateNodes( FILE *out,
                        struct Region_Node_Handle *Region_Node_Handles )
/*-----EndDocHead-----*/
{
    // int i,j,k, GnEquipPerRegion,OtEquipPerRegion;
    struct Filter_Unit_List *GnHigh, *OtHigh;
    struct Filter_Unit_List *GnLow, *OtLow ;
    // struct Filter_Unit_List *tempF ;
    struct Node_Definition *NodDef;
    struct Region_List *RegList ;
    struct Nodes_of_Fed_List *NodeOfFed ;
    struct Nodes_of_Fed_List *NodeOfFedList ;
    int CatOneDone, CatTwoDone ;
    struct Unit_Characteristics *UnitChar ;

    /* Number of nodes for Category 1 data */
    /* Number of nodes for Category 2 data */

    CatOneDone = 0 ;
    CatTwoDone = 0 ;
    Region_Node_Handles->xtFed = NULL ; /* c_Nodes_of_Fed_List(" CreateRegions "); */

    RegList = Region_Node_Handles->xtReg ;
    NodDef = Region_Node_Handles->NodesDefined ;
    GnHigh = OtHigh = GnLow = OtLow = NULL ;

    do {
        if ( RegList != NULL ) {
            // if ( RegList->EquipGn* > NodDef-> NodeEq )

            /* the regions that tend to be a heterogeneous forces */
            do {
                GetUnitsByCoTypeForRegion(out, 1, 'G', RegList, RegList->xtGnRegEle, &GnHigh);
                GetUnitsByCoTypeForRegion(out, 1, 'O', RegList, RegList->xtOtRegEle, &OtHigh);

                //fprintf(out," Region %2d: %1d: ", RegList->Id, RegList->Category );

                if ( GnHigh == NULL && OtLow == NULL ) { /* finished with category 1 */
                    CatOneDone = 1;
                }
                else {
                    AddRatioUnitsToNode(out, Region_Node_Handles,
                                         RegList->EquipGn, RegList->EquipOt,
                                         &GnHigh, &OtHigh);
                }
            } while( GnHigh !=NULL && OtHigh != NULL && CatOneDone == 0 ) ;
            /* the regions that tend to be a homogeneous forces */
            // PrintNodesOfFed( stdout, Region_Node_Handles->xtFed, " Cat 1 Building\n");

            RegList = RegList->xtReg ;
        }
    } while( RegList != NULL );

    fprintf(stdout, "\n");
    PrintNodesOfFed( stdout, Region_Node_Handles->xtFed, " Cat 1 Building\n");

    NodeOfFed = Region_Node_Handles->xtFed; /* Thats all for the High Nodes*/

    while( NodeOfFed->xtNodeOfFed !=NULL) { NodeOfFed = NodeOfFed->xtNodeOfFed; }
    NodeOfFed->Initialized = 1 ;

    /* Do the rest to the remaining nodes starting with Green units */

    RegList = Region_Node_Handles->xtReg ;
    NodDef = Region_Node_Handles->NodesDefined ;
    GnLow = GnHigh; /* don't lose any */

```

30 June 1999

```

OtLow = OtHigh;

do {
    if ( RegList != NULL ) {

        GetUnitsByCoTypeForRegion(out, 2, 'G', RegList, RegList->xtGnRegEle, &GnLow);
        //fprintf(out, " Region %2d: %1d: ", RegList->Id, RegList->Category );

        RegList = RegList->xtReg ;
    }
} while( RegList != NULL );

/* the regions that tend to be a homogeneous forces */
do {
    AddSupportUnitsToNode(out, Region_Node_Handles, &GnLow, &OtLow);
} while( GnLow !=NULL );

// PrintNodesOfFed( stdout, Region_Node_Handles->xtFed, "Cat 2 Building\n");

/* Do the rest to the remaining nodes with Other units*/
CatTwoDone = 0;
RegList = Region_Node_Handles->xtReg ;
NodDef = Region_Node_Handles->NodesDefined ;
GnHigh = OtHigh = GnLow = OtLow = NULL ;
do {
    if ( RegList != NULL ) {
        GetUnitsByCoTypeForRegion(out, 2, 'O', RegList, RegList->xtOtRegEle, &OtLow);
        //fprintf(out, " Region %2d: %1d: ", RegList->Id, RegList->Category );
        RegList = RegList->xtReg ;
    }
} while( RegList != NULL );

/* the regions that tend to be a homogeneous forces */
do {
    AddSupportUnitsToNode(out, Region_Node_Handles, &GnLow, &OtLow);
} while( OtLow != NULL && CatTwoDone == 0 );

PrintNodesOfFed( stdout, Region_Node_Handles->xtFed, " AT EndBuilding\n");

/* Just before it is finished must establish links between Regions and nodes */

UnitChar = Region_Node_Handles->UnitGn ;
NodeOfFedList = Region_Node_Handles->xtFed ;
RegList = Region_Node_Handles->xtReg ;

while( UnitChar != NULL ){

    AddNodeLinksToRegionsByUnit( out, UnitChar,
                                RegList, NodeOfFedList);
    UnitChar = UnitChar->ngep;
}

UnitChar = Region_Node_Handles->UnitOt ;
RegList = Region_Node_Handles->xtReg ;
while( UnitChar != NULL ){

    AddNodeLinksToRegionsByUnit( out, UnitChar,
                                RegList, NodeOfFedList);
    UnitChar = UnitChar->ngep;
}

PrintNodesOfFed( stdout, Region_Node_Handles->xtFed, " AT EndBuilding\n");
PrintRegionsNodes( stdout, RegList, " byRegions\n" );
}

```

/*-----*/
/*----- DocHeading */

30 June 1999

```

extern void AddRatioUnitsToNode( FILE *out,
                                struct Region_Node_Handle *Region_Node_Handles,
                                int Gn, int Ot,
                                struct Filter_Unit_List **GnList,
                                struct Filter_Unit_List **OtList )
/*-----EndDocHead-----*/
{ /* GnList & OtList will be updated */
/* need to go through the list of companies and add #:# ratio */
/* until the node has the prescribed percent of enties */

    struct Filter_Unit_List    *GnRegUnits, *OtRegUnits, *tFillList;
    struct Nodes_of_Fed_List    *NodeOfFed ;
    struct Node_Definition      *NodeDef;

    double Ratio;
    int    CurGn, CurOt, tEquip;
    double CurRatio;
    extern void PrintFilterList( FILE *out,
                                struct Filter_Unit_List *FilteredList,
                                char *Emark );
    GnRegUnits = *GnList;
    OtRegUnits = *OtList;

    NodeDef = Region_Node_Handles->NodesDefined ;
    NodeOfFed = Region_Node_Handles->xtFed;

    // find the beginning of the list for the Federate | Nodes
    do { /* do until more needs to be added to the Unit List */
        if ( NodeOfFed == NULL ) {
            Region_Node_Handles->xtFed = c_Nodes_of_Fed_List("AddRatio NodeId+=1");
            NodeOfFed = Region_Node_Handles->xtFed ;
        }
        else {
            while( NodeOfFed->xtNodeOfFed !=NULL) { NodeOfFed = NodeOfFed->xtNodeOfFed; }
            if ( NodeOfFed->Initialized == 1 &&
                NodeOfFed->NodeId < SCENARIOHigh ) {
                NodeOfFed->xtNodeOfFed= c_Nodes_of_Fed_List("AddRatio NodeId+=1");
                NodeOfFed = NodeOfFed->xtNodeOfFed ;
            } /* Otherwise finish filling Node */
        }
    }
    // FORCE RATIO is implied by the number of G:O in a region
    if ( Ot > 0 ) {
        Ratio = (double)Gn/(double)Ot;
    }
    else {
        Ratio = 0.0;
    }

    fprintf(out,
        "Approx Node %2d High %5d Ratio %7.4f Low %5d H(%5d,%5d) L(%5d, %5d) AddRatio ",
        NodeOfFed->NodeId, NodeDef->HighEquipment, Ratio, NodeDef->LowEquipment,
        NodeDef->HighGn, NodeDef->HighOt, NodeDef->LowGn, NodeDef->LowOt );

    CurGn = NodeOfFed->NumGn ;
    CurOt = NodeOfFed->NumOt ;
    CurRatio = NodeOfFed->Ratio ;

    //PrintFilterList(stdout, GnRegUnits, "AddtoNodes1\n");
    /* Need more Gn add one company + subordinates (opt CMDR)
        { (Ratio - Ratio*0.1) || (OtRegUnits == NULL &&) */
    do {
        if ( CurRatio < 0.8 && GnRegUnits != NULL ) {
            tEquip = AddToNode( out, NodeOfFed, GnRegUnits->UChrp );
            NodeOfFed->NumGn += tEquip;
            CurGn = NodeOfFed->NumGn ;
            tFillList = GnRegUnits ;
            GnRegUnits = GnRegUnits->nxtFilteredUnitp;
        }
    }
}

```


30 June 1999

```

        tFillList->nxtFilteredUnitp = NULL ;
        d_Filter_Unit_List( tFillList );      /* deallocate memory */
    }
    else if (OtRegUnits != NULL) {      /* need more Ot */
        tEquip = AddToNode( out, NodeOfFed, OtRegUnits->UChrp );
        NodeOfFed->NumOt += tEquip;
        CurOt = NodeOfFed->NumOt ;
        tFillList = OtRegUnits ;
        OtRegUnits = OtRegUnits->nxtFilteredUnitp;
        tFillList->nxtFilteredUnitp = NULL ;
        d_Filter_Unit_List( tFillList );      /* deallocate memory */
    }
    if ( CurOt > 0 ) { CurRatio = (double)CurGn/(double)CurOt ; }
    else { CurRatio = 10.0 ; }
    NodeOfFed->Ratio = CurRatio ;

//fprintf(out,"CurGn %5d CurOt %5d %7.4f  \n",
//      NodeOfFed->NumGn, NodeOfFed->NumOt, NodeOfFed->Ratio );

} while( GnRegUnits != NULL && OtRegUnits != NULL &&      /* Not at end of either list*/
        (CurGn + CurOt) < (NodeDef->HighEquipment - NodeDef->HighEquipment * 0.2)) ; /*
and node not full */

fprintf(out,"CurGn %5d CurOt %5d %7.4f  out of the loop \n",
        NodeOfFed->NumGn,NodeOfFed->NumOt,NodeOfFed->Ratio );

if ( NodeDef->HighEquipment - NodeDef->HighEquipment * 0.2 <= (CurGn + CurOt) ) {
    NodeOfFed->Initialized = 1 ;
}
} while( GnRegUnits != NULL && OtRegUnits != NULL ) ;
*GnList = GnRegUnits;
*OtList = OtRegUnits;
}

/*-----                                DocHeading      */

```

30 June 1999

```

extern void AddSupportUnitsToNode( FILE *out,
                                struct Region_Node_Handle *Region_Node_Handles,
                                struct Filter_Unit_List **GnList,
                                struct Filter_Unit_List **OtList )
/*-----EndDocHead-----*/
{ /* GnList & OtList will be updated */
/* need to go through the list of companies and add #:# ratio */
/* until the node has the prescribed percent of enties */

    struct Filter_Unit_List *GnRegUnits, *OtRegUnits, *tFillList;
    struct Nodes_of_Fed_List *NodeOfFed ;
    struct Node_Definition *NodeDef;
    double Ratio;
    int CurGn, CurOt, tEquip;
    double CurRatio;

    GnRegUnits = *GnList;
    OtRegUnits = *OtList;
    NodeDef = Region_Node_Handles->NodesDefined ;
    NodeOfFed = Region_Node_Handles->xtFed;

    // find the beginning of the list for the Federate | Nodes
    do { /* do until more needs to be added to the Unit List */
        if ( NodeOfFed == NULL ) {
            Region_Node_Handles->xtFed = c_Nodes_of_Fed_List("AddUnits NodeId+=1");
            NodeOfFed = Region_Node_Handles->xtFed ;
        }
        else {
            while( NodeOfFed->xtNodeOfFed !=NULL) { NodeOfFed = NodeOfFed->xtNodeOfFed; }
            if ( NodeOfFed->Initialized == 1 &&
                NodeOfFed->NodeId < SCENARIOLimitsOnFederates ) {
                NodeOfFed->xtNodeOfFed= c_Nodes_of_Fed_List("AddUnits NodeId+=1");
                NodeOfFed = NodeOfFed->xtNodeOfFed ;
            } /* Otherwise finish filling Node */
        }

        fprintf(out,
            "Approx Node %2d High %5d Ratio %7.4f Low %5d H(%5d,%5d) L(%5d, %5d) AddSupport",
                NodeOfFed->NodeId, NodeDef->HighEquipment, Ratio, NodeDef->LowEquipment,
                NodeDef->HighGn, NodeDef->HighOt, NodeDef->LowGn, NodeDef->LowOt );

        CurGn = NodeOfFed->NumGn ;
        CurOt = NodeOfFed->NumOt ;
        CurRatio = NodeOfFed->Ratio ;

        // PrintFilterList(stdout, GnRegUnits, " AddtoNodes2\n");

        do { /* Need more Gn add one company + subordinates (opt CMDR)*/
            if ( (*GnList != NULL) ) {

                tEquip = AddToNode( out, NodeOfFed, GnRegUnits->UChrp );
                NodeOfFed->NumGn += tEquip;
                CurGn = NodeOfFed->NumGn ;
                tFillList = GnRegUnits ;
                GnRegUnits = GnRegUnits->nxtFilteredUnitp;
                tFillList->nxtFilteredUnitp = NULL ;
                d_Filter_Unit_List( tFillList ); /* deallocate memory */
            }
            else if ( *OtList != NULL) { /* need more Ot */

                tEquip = AddToNode( out, NodeOfFed, OtRegUnits->UChrp );
                NodeOfFed->NumOt += tEquip;
                CurOt = NodeOfFed->NumOt ;
                tFillList = OtRegUnits ;
                OtRegUnits = OtRegUnits->nxtFilteredUnitp;
                tFillList->nxtFilteredUnitp = NULL ;
            }
        }
    }
}

```

30 June 1999

```

        d_Filter_Unit_List( tFillList );      /* deallocate memory */
    }
    if ( CurOt > 0 ) { CurRatio = (double)CurGn/(double)CurOt ; }
    else { CurRatio = 10.0 ; }
    NodeOfFed->Ratio = CurRatio ;

} while( GnRegUnits != NULL && OtRegUnits != NULL && /* Not at end of either list*/
        (CurGn + CurOt) < (NodeDef->LowEquipment - NodeDef->LowEquipment * 0.2)) ; /* and
node not full */

fprintf(out, "CurGn %5d CurOt %5d %7.4f      \n",
        NodeOfFed->NumGn, NodeOfFed->NumOt, NodeOfFed->Ratio);

if ( NodeDef->LowEquipment - NodeDef->LowEquipment * 0.2 <= (CurGn + CurOt) ) {
    NodeOfFed->Initialized = 1 ;
}
} while( GnRegUnits != NULL && OtRegUnits != NULL ) ;
*GnList = GnRegUnits;
*OtList = OtRegUnits;
} /* END of AddSupportUnitsToNode */

/*----- DocHeading */

```

30 June 1999

```

extern int AddToNode( FILE *out,
                    struct Nodes_of_Fed_List *NodeOfFed,
                    struct Unit_Characteristics *UnitChar )
/*-----EndDocHead-----*/
{ /* GnList & OtList will be updated */
  /* need to go through the list of companies and add #:# ratio */
  /* until the node has the prescribed percent of enties */

  struct Units_on_Node_List *UnitOnNode ;
  struct Nodes_of_Fed_List *tNodeOfFed ;
  struct Unit_Characteristics *UnChar ;
  struct Unit_List *UnList ;
  int PiecesOfEquip;
/*
  // What to do
  // if Unit node is still == 0 then no assignment has taken place
  // add unit to node & update Unit_Characteristics
  // add subordinates to node & update Unit_Characteristics
  // add Commander to Node & update Unit_Characteristics
*/
  PiecesOfEquip = 0 ;

  if ( NodeOfFed->UnitOnNode == NULL ) { /* first unit assigned to node */
    NodeOfFed->UnitOnNode = c_Units_on_Node_List( " AddToNode");
    UnitOnNode = NodeOfFed->UnitOnNode;
  }
  else {
    UnitOnNode = NodeOfFed->UnitOnNode; /* find the last one */
    while (UnitOnNode->xtUnitOnNode != NULL) { UnitOnNode = UnitOnNode->xtUnitOnNode;}
    UnitOnNode->xtUnitOnNode = c_Units_on_Node_List( " AddToNode s");
    UnitOnNode = UnitOnNode->xtUnitOnNode;
  }
  if ( UnitChar->FedNode == 0 ) { /* assign to node */
    UnitChar->FedNode = NodeOfFed->NodeId ;
    PiecesOfEquip += UnitChar->Equipment ;
    UnitOnNode->UChrp = UnitChar ;
    // printf("Name: %s \n", UnitChar->Name );
  }
  // Now assign subordinates
  if ( UnitChar->ULstp != NULL && UnitChar->ULstp->Subrdp != NULL ) {
    UnList = UnitChar->ULstp->Subrdp ;
    UnChar = UnList->UChrp ; /* just core dump if UChrp == NULL & debug */

    do {
      UnChar = UnList->UChrp ;
      if ( UnChar->FedNode != 0 ) {
        fprintf(stderr, "Warning: AddToNode Try to overwriting Unit Node assignment %2d with
%2d\n",
              UnChar->FedNode, NodeOfFed->NodeId );
      }
      else {
        UnitOnNode->xtUnitOnNode = c_Units_on_Node_List( " AddToNode s");
        UnitOnNode = UnitOnNode->xtUnitOnNode;

        UnChar->FedNode = NodeOfFed->NodeId ;
        PiecesOfEquip += UnChar->Equipment ;
        UnitOnNode->UChrp = UnChar ;
        // printf("Name: %s %4d \n", UnChar->Name, PiecesOfEquip);
      }
      UnList = UnList->nrep ;
    } while (UnList != NULL ); /* && UnList != UnitChar->ULstp->Subrdp ) ;*/
  } /* not NULLs */

  // Now assign the commander

```

30 June 1999

```

if ( UnitChar->ULstp != NULL && UnitChar->ULstp->UCmdp != NULL ) {

    if ( UnitChar->ULstp->UCmdp->UChrp != NULL &&
        UnitChar->ULstp->UCmdp->UChrp->FedNode == 0 ){ /* not previously assigned */

        UnList = UnitChar->ULstp->UCmdp ;
        UnChar = UnList->UChrp ;           /* just core dump if UChrp == NULL & debug */

        //      printf("Name: %s %5d bn\n",  UnChar->ULstp->UCmdp->UChrp->Name,
        //              UnChar->ULstp->UCmdp->UChrp->FedNode);

        UnitOnNode->xtUnitOnNode = c_Units_on_Node_List( " AddToNode s");
        UnitOnNode = UnitOnNode->xtUnitOnNode;

        UnChar->FedNode = NodeOfFed->NodeId ;
        PiecesOfEquip    += UnChar->Equipment ;
        UnitOnNode->UChrp = UnChar ;
        //      printf("Name: %s    bn  %4d \n", UnChar->Name, PiecesOfEquip);

    }
}
return(PiecesOfEquip);
}

/*-----*/
/*----- DocMethod */
extern void RemoveNodeLinksToRegionsByUnit( FILE *out, /* FedNodes.c */
        struct Unit_Characteristics *UnitChar,
        struct Region_List          *RegList,
        struct Nodes_of_Fed_List    *FedList )
/*-----EndDocHead---*/
{
    ;
    // if any Unit on node references that region then it cannot
    // be removed.
    // so first have to filter a list off all units in that region
    // then
    // if any Unit != UnitChar is still in that region for that
    // node then the links stay that same
    //
    //
}/*-----*/
/*-----*/
/*----- DocHeading */

```

30 June 1999

```

extern void AddNodeLinksToRegionsByUnit( FILE *out,
                                         struct Unit_Characteristics *UnitChar,
                                         struct Region_List *RegList,
                                         struct Nodes_of_Fed_List *FedList )
/*-----EndDocHead---*/
{
    struct Unit_Characteristics *UnChar ;
    struct Region_List *cRegList, *uRegList ;
    struct Nodes_of_Fed_List *cFedList, *aFedNode ;
    struct Nodes_wrt_Region_List *cNodeWRTRegion, *pNodeWRTRegion ;
    struct Unit_Region_List *cRegOfUnit ;
    int RegionAlreadyReferencesNode ;

    cRegList = RegList ;
    cFedList = FedList ;
    UnChar = UnitChar ;
    RegionAlreadyReferencesNode = 0 ;

    if ( UnChar != NULL && RegList != NULL && FedList != NULL ) {

        if (UnChar->FedNode != 0 ) { /* Unit is allocated to Federate - otherwise ignore*/
            cRegOfUnit = UnChar->RegOfUnit;

            while (cRegOfUnit != NULL ) {

                uRegList = cRegOfUnit->xtReg; /* for this region add a NodeWRTFed reference*/

                if ( uRegList->NodeWRTRegion == NULL ) { /* associate 1st Node to region */

                    uRegList->NodeWRTRegion = c_Nodes_wrt_Region_List("AddSupportUnitsToNode cReg");
                    cNodeWRTRegion = uRegList->NodeWRTRegion ;

                    aFedNode = FindNode(UnChar->FedNode, FedList);

                    if ( aFedNode != NULL ) { /* for a REGION this is the first node added */
                        cNodeWRTRegion->NodeId = UnChar->FedNode ;
                        cNodeWRTRegion->NodeOfFed = aFedNode ;
                        AddRegionToNode( aFedNode, uRegList );
                    }

                }
                else { /* go through Node Federate list on Region */
                    pNodeWRTRegion = NULL ;
                    cNodeWRTRegion = uRegList->NodeWRTRegion ; /* for a Region */
                    RegionAlreadyReferencesNode = 0 ;

                    while( cNodeWRTRegion != NULL ) {
                        if ( cNodeWRTRegion->NodeId == UnChar->FedNode ) {
                            RegionAlreadyReferencesNode = 1 ;
                        }
                        pNodeWRTRegion = cNodeWRTRegion;
                        cNodeWRTRegion = cNodeWRTRegion->xtNodeWRTRegion;
                    }
                    if ( pNodeWRTRegion != NULL && RegionAlreadyReferencesNode == 0 ) {
                        /* then add to list */
                        pNodeWRTRegion->xtNodeWRTRegion = c_Nodes_wrt_Region_List("AddSupportUnitsToNode
cReg");
                        pNodeWRTRegion = pNodeWRTRegion->xtNodeWRTRegion ;

                        aFedNode = FindNode( UnChar->FedNode, FedList); /* pointer */

                        if ( aFedNode != NULL ) {
                            pNodeWRTRegion->NodeId = UnChar->FedNode ;
                            pNodeWRTRegion->NodeOfFed = aFedNode ;
                        }
                    }
                }
                // Does that node know of the region ?
            }
        }
    }
}

```

30 June 1999

```
/* for the node make sure that the REGION is referenced */

    AddRegionToNode( aFedNode, uRegList );
}
}
cRegOfUnit = cRegOfUnit->nxtRegOfUnit ;
}
} /* if FedNode */
} /* if no pointers are null */
} /* end of AddNodeLinks */

/*----- DocHeading */
```

30 June 1999

```

extern void AddRegionToNode( struct Nodes_of_Fed_List *FedNode, /* FedNodes.c */
                           struct Region_List *RegListEle )
/*-----EndDocHead---*/
{
    struct Unit_Region_List      *cRegOnNode, *pRegOnNode;
    struct Region_List           *foundRegOnNodeAlready ;

    foundRegOnNodeAlready = NULL ;

    if ( FedNode->RegOnNode == NULL ) {

        FedNode->RegOnNode = c_Unit_Region_List("AddRegionToNode") ;
        cRegOnNode = FedNode->RegOnNode;
        cRegOnNode->xtReg = RegListEle ;

    }
    else {
        cRegOnNode = FedNode->RegOnNode;
        pRegOnNode = NULL ;
        while( cRegOnNode != NULL ) {

            if ( cRegOnNode->xtReg == RegListEle ) {
                foundRegOnNodeAlready = cRegOnNode->xtReg;
                printf(" OnURL %s ", foundRegOnNodeAlready->Name );
            }

            pRegOnNode = cRegOnNode;
            cRegOnNode = cRegOnNode->nxtRegOfUnit;
        }
        if ( foundRegOnNodeAlready == NULL && pRegOnNode != NULL) { /* not already added */

            pRegOnNode->nxtRegOfUnit = c_Unit_Region_List("AddRegionToNode A");
            pRegOnNode = pRegOnNode->nxtRegOfUnit ;
            pRegOnNode->xtReg = RegListEle ;

        }
    }
}

/*----- DocHeading */

```



```
extern struct Nodes_of_Fed_List *FindNode( int Id, /* FedNodes.c */
                                         struct Nodes_of_Fed_List *FedList )
/*-----EndDocHead---*/
{
    struct Nodes_of_Fed_List    *iFedList, *foundFedNode ;

    iFedList = FedList ;

    while( iFedList != NULL ) {

        if ( iFedList->NodeId == Id ) { foundFedNode = iFedList ; }

        iFedList = iFedList->xtNodeOfFed;
    }
    return( foundFedNode );
}

/*-----*/
/*-----DocHeading-----*/
```

30 June 1999

```

extern void PrintNodesOfFed( FILE *out, /* FedNodes.c */
    struct Nodes_of_Fed_List *NodeOfFedList ,
    char *Emark )
/*-----EndDocHead---*/
{
    struct Nodes_of_Fed_List *NodeOfFed;
    struct Unit_Region_List *RegOnNode;
    int Gn, Ot ;
    double Rat ;
    NodeOfFed = NodeOfFedList;
    if ( NodeOfFed != NULL ) {
        Gn = Ot = 0 ;
        do {
            fprintf(out, "Node %2d Init %1d Gn %5d Ot %5d Ratio %7.4f  ",
                NodeOfFed->NodeId, NodeOfFed->Initialized,
                NodeOfFed->NumGn, NodeOfFed->NumOt, NodeOfFed->Ratio );

            RegOnNode = NodeOfFed->RegOnNode;
            while(RegOnNode != NULL) {
                fprintf(out, " %s", RegOnNode->xtReg->Name );
                RegOnNode = RegOnNode->nxtRegOfUnit ;
            }
            fprintf(out, ":%s", Emark);

            Gn += NodeOfFed->NumGn;
            Ot += NodeOfFed->NumOt;

            NodeOfFed = NodeOfFed->xtNodeOfFed ;
        } while( NodeOfFed != NULL );

        if ( Ot != 0 ) { Rat = (double)Gn/(double)Ot ; }
        else { Rat = 0.0 ; }

        fprintf(out, "Node ALL total Gn %5d Ot %5d Ratio %7.4f  :%s\n",
            Gn, Ot, Rat, Emark);
    }
}
/*----- DocHeading */

```

30 June 1999

```

extern void PrintUnitsOfFed( FILE *out, /* FedNodes.c */
    struct Nodes_of_Fed_List *NodeOfFedList ,
    char *Emark )
/*-----EndDocHead-----*/
{
    struct Nodes_of_Fed_List *NodeOfFed;
    struct Units_on_Node_List *UnitOnNode;

    int Gn, Ot ;
    double Rat ;

    NodeOfFed = NodeOfFedList;

    if ( NodeOfFed != NULL ) {
        Gn = Ot = 0 ;
        do {
            fprintf(out, "Node %2d Init %1d Gn %5d Ot %5d Ratio %7.4f  \n",
                NodeOfFed->NodeId, NodeOfFed->Initialized,
                NodeOfFed->NumGn, NodeOfFed->NumOt, NodeOfFed->Ratio );
            Gn = 0 ;
            UnitOnNode = NodeOfFed->UnitOnNode;
            while( UnitOnNode != NULL ) {
                fprintf(out, "      %s      %4d\n", UnitOnNode->UChrp->Name,
                    UnitOnNode->UChrp->Equipment );
                Gn += UnitOnNode->UChrp->Equipment ;
                UnitOnNode = UnitOnNode->xtUnitOnNode ;
            }

            fprintf(out, "Node %2d Total Equipment %5d :%s\n",
                NodeOfFed->NodeId, Gn, Emark);

            NodeOfFed = NodeOfFed->xtNodeOfFed ;
        } while( NodeOfFed != NULL );
    }
}
/*-----*/
/*----- DocHeading -----*/

```

30 June 1999

```

extern int SubscribeByFederate( FILE *out, /* FedNodes.c */
                               struct Nodes_of_Fed_List *NodeOfFedList )
/*-----EndDocHead-----*/
{
    struct Nodes_of_Fed_List *NodeOfFed;
    struct Unit_Region_List *RegOnNode;
    int Gn, Ot, i, j, SubscinVENTORY, InActInventory, RegInventory, grand;
    struct Event_Message *NewMsg;
    int obj_class_nbr, interact_class_nbr;
    double Rat;
    double SubTime, IncrTime;

    SubTime = SubscribeStart;

    // Subscribe for federates with region
    // Region 1 Route, Health Order, report, Fire, Sense, Supply
    // Region 2,3 Route, Health Fire, Supply
    // Region 4,5 Route, Health Fire, Supply

    // Publish for federates with region
    // Region 1 Route, Health Order, report, Fire, Sense
    // Region 2,3 Route, Health Supply
    // Region 4,5 Route, Health Supply

    NodeOfFed = NodeOfFedList;
    if ( NodeOfFed != NULL ) {
        Gn = Ot = 0;
        grand = 0;
        do {
            SubscinVENTORY = 0;
            InActInventory = 0;
            RegInventory = 0;
            fprintf(out, "SubscribeNode %2d ", NodeOfFed->NodeId);
            IncrTime += 0.0001;

            RegOnNode = NodeOfFed->RegOnNode;
            fprintf(out, "ForRegionSubscribe: ");
            while (RegOnNode != NULL) {
                fprintf(out, " %s", RegOnNode->xtReg->Name);
                for (j=0; j<ObjectsInSOM; j++) {
                    if ( RegOnNode->xtReg->SubObjects[j] > 0 &&
                        NodeOfFed->Objects[j] > 0 ) {
                        obj_class_nbr = j + OffsetObject;
                        NewMsg = SetExtendEventMessage(
                            1, /* RTIcommand, */
                            0, /* SIMcommand, */
                            RTI_SUBSCRIBE_OBJCLSS, /* Action, */
                            1, /* fedrtn_exname, */
                            NodeOfFed->NodeId, /* Federate */
                            obj_class_nbr, /* obj_class_nbr, */
                            0, /* obj_instance_nbr, */
                            0, /* interact_class_nbr, */
                            0, /* interact_instance_nbr */
                            0.0, /* fedrtn_time */
                            RegOnNode->xtReg->Id, /* region_nbr, */
                            0, /* routing_space_nbr */
                            0, /* nbr_rcvd_msgs */
                            0, /* nbr_sent_msgs */
                            0.0, /* LBTS_time */
                            (SubTime+IncrTime), /* lPhysicalTime, */
                            (SubTime+IncrTime), /* lVirtualTime, */
                            "Init" ); /* just a note */
                        AddEvent( stdout, "InToRTI", NewMsg );
                        SubscinVENTORY += 1;
                    }
                }
                RegOnNode = RegOnNode->next;
            }
        } while (NodeOfFed->next != NULL);
    }
}

```

30 June 1999

```

for(j=0; j<InteractionsInSOM; j++) {
    if ( RegOnNode->xtReg->SubInteract[j] > 0 &&
        NodeOfFed->Interact[j] > 0 ) {
        interact_class_nbr = j + OffsetInteraction ;
        NewMsg = SetExtendEventMessage(
            1,                                /* RTIcommand, */
            0,                                /* SIMcommand, */
            RTI_SUBSCRIBE_INTCLSS,            /* Action, */
            1,                                /* fedrtn_exname,*/
            NodeOfFed->NodeId,                 /* Federate */
            0,                                /* obj_class_nbr, */
            0,                                /* obj_instance_nbr,*/
            interact_class_nbr,               /* interact_class_nbr, */
            0,                                /* interact_instance_nbr */
            0.0,                              /* fedrtn_time */
            RegOnNode->xtReg->Id,              /* region_nbr, */
            0,                                /* routing_space_nbr */
            0,                                /* nbr_rcvd_msgs */
            0,                                /* nbr_sent_msgs */
            0.0,                              /* LBTS_time */
            (SubTime+IncrTime),               /* lPhysicalTime, */
            (SubTime+IncrTime),               /* lVirtualTime, */
            "Init" );                         /* just a note */
        AddEvent( stdout, "InToRTI", NewMsg );
        InActinventory += 1 ;
    }
}
RegInventory +=1 ;
RegOnNode = RegOnNode->nxtRegOfUnit ;
}
fprintf( out, "\n" );

Gn += NodeOfFed->NumGn;
Ot += NodeOfFed->NumOt;

fprintf(out,
    "SubscribeNode %2d to %2d Regions Subscribes %4d Interaction %4d by %8.3f\n",
        NodeOfFed->NodeId, RegInventory, Subscinventory, InActinventory,
        (SubTime+IncrTime) );
grand += Subscinventory + InActinventory;
NodeOfFed = NodeOfFed->xtNodeOfFed ;

} while( NodeOfFed != NULL );

if ( Ot != 0 ) { Rat = (double)Gn/(double)Ot ; }
else { Rat = 0.0 ; }

fprintf(out,"Node ALL total Gn %5d Ot %5d Ratio %7.4f    \n",
    Gn, Ot, Rat );
}
return(grand);
}

/*-----*/
/*----- DocHeading */

```

30 June 1999

```

extern int PublishByFederate( FILE *out,
                             struct Nodes_of_Fed_List *NodeOfFedList )
/*-----EndDocHead-----*/
{
    struct Nodes_of_Fed_List *NodeOfFed;
    struct Unit_Region_List *RegOnNode;
    int Gn, Ot, i, j, SubscinVENTORY, InActInventory, RegInventory, grand;
    struct Event_Message *NewMsg;
    int obj_class_nbr, interact_class_nbr, Publish;
    double Rat;
    double PubTime, IncrTime;

    PubTime = PublishStart;

    NodeOfFed = NodeOfFedList;
    if ( NodeOfFed != NULL ) {
        Gn = Ot = 0;
        grand = 0;
        IncrTime = 0.0;
        do {
            SubscinVENTORY = 0;
            InActInventory = 0;
            RegInventory = 0;
            fprintf(out, "PublishNode %2d ", NodeOfFed->NodeId);
            IncrTime += 0.001;
            for(j=0; j<5; j++) {
                Publish = 0;
                RegOnNode = NodeOfFed->RegOnNode;
                while(RegOnNode != NULL) {
                    printf("%2d Region %2d Objects %1d %s \n",
                           j, RegOnNode->xtReg->Id,
                           RegOnNode->xtReg->PubObjects[j],
                           RegOnNode->xtReg->Name);
                    if ( RegOnNode->xtReg->PubObjects[j] > 0 ) {
                        Publish = 1;
                    }
                    RegOnNode = RegOnNode->nxtRegOfUnit;
                }

                if ( Publish && NodeOfFed->Objects[j] > 0 ) {
                    obj_class_nbr = j + OffsetObject;
                    NewMsg = SetExtendEventMessage(
                        1, /* RTIcommand, */
                        0, /* SIMcommand, */
                        RTI_PUBLISH_OBJCLSS, /* Action, */
                        1, /* fedrtn_exname, */
                        NodeOfFed->NodeId, /* Federate */
                        obj_class_nbr, /* obj_class_nbr, */
                        0, /* obj_instance_nbr, */
                        0, /* interact_class_nbr, */
                        0, /* interact_instance_nbr */
                        0.0, /* fedrtn_time */
                        0, /* region_nbr, */
                        0, /* routing_space_nbr */
                        0, /* nbr_rcvd_msgs */
                        0, /* nbr_sent_msgs */
                        0.0, /* LBTS_time */
                        (PubTime+IncrTime), /* lPhysicalTime, */
                        (PubTime+IncrTime), /* lVirtualTime, */
                        "Init" ); /* just a note */
                    AddEvent( stdout, "InToRTI", NewMsg );
                    SubscinVENTORY += 1;
                }
            }
        }
    }
}

```

30 June 1999

```

IncrTime += 0.001;
for(j=0; j<5; j++) {

    Publish = 0 ;
    RegOnNode = NodeOfFed->RegOnNode;
    while(RegOnNode != NULL) {
        printf("%2d Region %2d Interact %1d          %s \n",
            j, RegOnNode->xtReg->Id ,
            RegOnNode->xtReg->PubInteract[j],
            RegOnNode->xtReg->Name ) ;
        if ( RegOnNode->xtReg->PubInteract[j] > 0 ) {
            Publish = 1 ;
        }
        RegOnNode = RegOnNode->nxtRegOfUnit ;
    }
    if ( Publish && NodeOfFed->Interact[j] > 0) {
        interact_class_nbr = j + OffsetInteraction ;
        NewMsg = SetExtendEventMessage(
            1,                                /* RTIcommand, */
            0,                                /* SIMcommand, */
            RTI_PUBLISH_INTCLSS,              /* Action, */
            1,                                /* fedrtn_exname,*/
            NodeOfFed->NodeId,                 /* Federate */
            0,                                /* obj_class_nbr, */
            0,                                /* obj_instance_nbr,*/
            interact_class_nbr,               /* interact_class_nbr, */
            0,                                /* interact_instance_nbr */
            0.0,                              /* fedrtn_time */
            0,                                /* region_nbr, */
            0,                                /* routing_space_nbr */
            0,                                /* nbr_rcvd_msgs */
            0,                                /* nbr_sent_msgs */
            0.0,                              /* LBTS_time */
            (PubTime+IncrTime),               /* lPhysicalTime, */
            (PubTime+IncrTime),               /* lVirtualTime,*/
            "Init" ) ;                       /* just a note */
        AddEvent( stdout, "InToRTI", NewMsg );
        InActinventory += 1 ;
    }
}

fprintf( out, "\n" );

Gn += NodeOfFed->NumGn;
Ot += NodeOfFed->NumOt;

fprintf(out, "PublishNode %2d PublishObjCls %4d PublishIntCls %4d by %8.3f\n",
    NodeOfFed->NodeId, Subscinventory, InActinventory,
    (PubTime+IncrTime) );
grand += Subscinventory + InActinventory;

NodeOfFed = NodeOfFed->xtNodeOfFed ;

} while( NodeOfFed != NULL );

}
return(grand);
}
/* ----- end PublishByFederate */
/*----- DocHeading */

```

30 June 1999

```

/* file: FilterUnits.c */
/*----- */

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <math.h>
#include "soa_defs.h"
#include "soa_cnst.h"
#include "proto.h"
/*-----
All functions in FilterUnits

extern int  AddToFilterSubordinates( FILE  *out,
                                   struct Unit_List  *ULp,
                                   struct Filter_Unit_List  *Top )

extern int EquipInList( struct Filter_Unit_List  *List )

extern struct Filter_Unit_List  *FilterByEcheleon( FILE  *out,
            struct Unit_Characteristics  *UnCharOrigin,
            int  Echeleon)

extern struct Filter_Unit_List  *FilterByEchEquip( FILE  *out,
            struct Unit_Characteristics  *UnCharOrigin,
            int  Echeleon,
            int  TotalEquip )

extern struct Filter_Unit_List  *FilterNotAssignedToFed( FILE  *out,
            struct Unit_Characteristics  *UnCharOrigin )

extern struct Filter_Unit_List  *FilterNotInRegion( FILE  *out,
            struct Unit_Characteristics  *UnCharOrigin )

extern void MergeFilterList( struct Filter_Unit_List  *List1,
                             struct Filter_Unit_List  *List2 )

extern void PrintFilterList( FILE  *out,
            struct Filter_Unit_List  *FilteredList,
            char  *Emark )
*/

/* ----- UnitHier.c -----      View_Refresh f*/
/*-----                          DocHeading */

```


30 June 1999

```

extern int AddToFilterSubordinates( FILE *out,
                                   struct Unit_List *ULp,
                                   struct Filter_Unit_List *Top )

/*-----EndDocHead---*/
{
extern struct Filter_Unit_List *c_Filter_Unit_List(char *s);
struct Unit_List *cULp;
struct Filter_Unit_List *top ;

int Total, KeepTotal;
Total = 0; KeepTotal = 0 ;
cULp = ULp ;
top = Top;

if ( ULp != NULL && top != NULL ) {

do {

        top->nxtFilteredUnitp = c_Filter_Unit_List("FilterByEcheleon" );
top      = top->nxtFilteredUnitp;
top->UChrp = cULp->UChrp ;

if ( cULp->Subrdp != NULL ) {

        KeepTotal = Total ;
        Total = AddToFilterSubordinates( out, cULp->Subrdp, top );

        while( top->nxtFilteredUnitp != NULL) { top = top->nxtFilteredUnitp ;} /* go to end */

        Total += KeepTotal ;

}
else {

        Total += cULp->UChrp->Equipment ;

}

printf("AddToFilterSubord  %5d, %5d, %17s %1d\n", Total,
        cULp->UChrp->Equipment, cULp->UChrp->Name, cULp->UChrp->Echeleon );
//printf("AddToFilterSubord  %5d, %17s %1d\n", Total, cULp->UChrp->Name, cULp->UChrp-
>Echeleon );

        cULp = cULp->nrep ;

} while (cULp != NULL ) ;

}

printf("AddToFilterSubord  %5d,\n", Total);
return(Total);
}

/*-----DocHeading */

```

30 June 1999

```
extern int EquipnList( struct Filter_Unit_List *List )
```

```
/*-----EndDocHead---*/  
{
```

```
int Equipment ;  
Equipment = 0 ;
```

```
if ( List != NULL ) {
```

```
do {
```

```
    Equipment += List->UChrp->Equipment ;
```

```
    List = List->nxtFilteredUnitp ;
```

```
    } while ( List->nxtFilteredUnitp != NULL ) ;
```

```
}  
return(Equipment);  
}
```

```
/*----- GrowHier.c ----- FilterByEcheleon f*/  
/*----- DocHeading */
```

30 June 1999

```

extern struct Filter_Unit_List *FilterByEcheleon( FILE *out,
struct Unit_Characteristics
int
*UnCharOrigin,
Echeleon)

/*-----EndDocHead---*/
{
    struct Filter_Unit_List *top, *cur ;
    extern struct Filter_Unit_List *c_Filter_Unit_List(char *s);
    struct Unit_Characteristics *lUnChr;

    top = cur = NULL ;
    lUnChr = UnCharOrigin ;

do {

    if ( lUnChr->Echeleon == Echeleon ) {

        if ( top == NULL ) { /* initialize */

            top = c_Filter_Unit_List("FilterByEcheleon Start" );
            cur = top ;
        }
        else {

            cur->nxtFilteredUnitp = c_Filter_Unit_List("FilterByEcheleon" );
            cur = cur->nxtFilteredUnitp ;
        }

        cur->UChrp = lUnChr ;
    }

    lUnChr = lUnChr->ngep ;

} while ( lUnChr != NULL );

return( top );
}

/*----- GrowHier.c ----- FilterByEchEquip f*/
extern struct Filter_Unit_List *FilterByEchEquip( FILE *out,
struct Unit_Characteristics
int
int
*UnCharOrigin,
Echeleon,
TotalEquip )

/*-----EndDocHead---*/
{
    struct Filter_Unit_List *top, *cur ;
    extern struct Filter_Unit_List *c_Filter_Unit_List(char *s);
    extern int AddToFilterSubordinates( FILE *out,
struct Unit_List *ULp,
struct Filter_Unit_List *top );
    struct Unit_Characteristics *lUnChr;
    int SelectedEquip, AddedEquip;
    SelectedEquip = 0;
    top = cur = NULL ;
    lUnChr = UnCharOrigin ;

do {

    if ( lUnChr->Echeleon >= Echeleon && TotalEquip > SelectedEquip ) {

        if ( top == NULL ) { /* initialize */
            top = c_Filter_Unit_List("FilterByEcheleon Start" );
            cur = top ;
        }

        else {

```

30 June 1999

```

    cur->nxtFilteredUnitp = c_Filter_Unit_List("FilterByEcheleon" );
    cur = cur->nxtFilteredUnitp ;
}

cur->UChrp = lUnChr ;
SelectedEquip += lUnChr->Equipment ;

/* include all subordinates */

AddedEquip = AddToFilterSubordinates( out, lUnChr->ULstp->Subrdp, cur );

while( cur->nxtFilteredUnitp != NULL) { cur = cur->nxtFilteredUnitp ; }

if (      out != NULL ) {
    fprintf(out, "AddToFilterSubord %5d, %17s \n", AddedEquip,
        lUnChr->ULstp->Subrdp->UChrp->Name );
}
SelectedEquip += AddedEquip ;
}

lUnChr = lUnChr->ngep ;

} while ( lUnChr != NULL && TotalEquip > SelectedEquip );

return( top );
}

```

```

/*----- GrowHier.c ----- FilterNotInRegion f*/
/*----- DocHeading */

```

30 June 1999

```

extern struct Filter_Unit_List *FilterNotAssignedToFed( FILE *out,
    struct Unit_Characteristics *UnCharOrigin )
/*-----EndDocHead---*/
{
    struct Filter_Unit_List      *top, *cur ;
    struct Unit_Characteristics  *lUnChr;

    top = cur = NULL ;
    lUnChr = UnCharOrigin ;

    do {

        if ( lUnChr->FedNode == 0 ) {          /* not assigned to a Federate */

            if ( top == NULL ) { /* initialize */

                top = c_Filter_Unit_List("FilterByEcheleon Start" );
                cur = top ;
            }

            else {

                cur->nxtFilteredUnitp = c_Filter_Unit_List("FilterByEcheleon" );
                cur = cur->nxtFilteredUnitp ;
            }

            cur->UChrp = lUnChr ;
        }

        lUnChr = lUnChr->ngep ;
    } while ( lUnChr != NULL );

    return( top );
}

/*----- GrowHier.c ----- FilterNotInRegion f*/
/*----- DocHeading */

```

30 June 1999

```

extern struct Filter_Unit_List *FilterNotInRegion( FILE *out,
    struct Unit_Characteristics
    *UnCharOrigin )
/*-----EndDocHead---*/
{
    struct Filter_Unit_List      *top, *cur ;

    struct Unit_Characteristics  *lUnChr;

    top = cur = NULL ;
    lUnChr = UnCharOrigin ;
    do {

        if ( lUnChr->RegOfUnit == NULL ) {

            if ( top == NULL ) { /* initialize */

                top = c_Filter_Unit_List("FilterByEcheleon Start" );
                cur = top ;
            }
            else {

                cur->nxtFilteredUnitp = c_Filter_Unit_List("FilterByEcheleon" );
                cur = cur->nxtFilteredUnitp ;
            }
            cur->UChrp = lUnChr ;
        }
        lUnChr = lUnChr->ngep ;
    } while ( lUnChr != NULL );

    return( top );
}

/*----- DocHeading */

```

30 June 1999

```
extern void MergeFilterList( struct Filter_Unit_List *List1,
                             struct Filter_Unit_List *List2 )

/*-----EndDocHead---*/
{

if ( List1 != NULL ) {

    while ( List1->nxtFilteredUnitp != NULL ) {

        List1 = List1->nxtFilteredUnitp ;

    }

    if ( List2 !=NULL ) {

        List1->nxtFilteredUnitp = List2 ;

    }
}
else { List1 = List2 ; }
}

/*-----DocHeading */
```

30 June 1999

```

extern void PrintFilterList( FILE *out,
    struct Filter_Unit_List *FilteredList,
    char *Emark )
/*-----EndDocHead---*/
{
    struct Unit_Characteristics *UnCharX;

    int i ;

    if ( FilteredList != NULL ) {
        do {
            UnCharX = FilteredList->UChrp ;

            i = CountSubrEquip(stdout, UnCharX->ULstp->Subrdp ) ;

            i += UnCharX->Equipment;

            fprintf(out,"fl %5d %-21s Node %2d X %4d -Y %4d Ech %3d equip %4d Sub %4d  %s",
                UnCharX->Id, UnCharX->Name, UnCharX->FedNode,
                UnCharX->ViewHoriz, UnCharX->ViewVert, UnCharX->Echeleon, UnCharX->Equipment, i,
                Emark);

            FilteredList = FilteredList->nxtFilteredUnitp ;

        } while ( FilteredList != NULL ) ;
    }
/*-----DocHeading */

```



```

/* file: GrowArmy.c */
/* Grow Main Definitions
-----EndDocHead-----*/

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <math.h>
#include <unistd.h>
#include "soa_defs.h"
#include "soa_cnst.h"
#include "proto.h"

unsigned short int TriangleRandU[3];
static struct Unit_Characteristics *DirectToUnit[LimitOnUnits] ;
typedef struct Unit_Echeleon {
int    CanDo ;
int    TotMax ;
int    Cnt ;
int    Ent ;
int    Sub ;
int    MinSub;
int    MaxSub;
char   EchName[24];
} Unit_Echelon_Type ;

int Levellimit = 6 ; /* This is where to select the level of hierarchies */
int gLevel = 0 ;
int oLevel = 0 ;
int counter = 0 ;
int Gid = 0 ;
unsigned short int GrowXsubi[3];

static struct Unit_List          *UnLstG ;
static struct Unit_List          *UnLstO ;

int                               Total_Force_Battalions;

struct Region_Definition         Regions;
struct Node_Definition           Nodes;

/* Beginning of function prototypes */

struct    Unit_Echeleon  Guide[9] ; /* 0 is Army, 7 is Squad */

/*----- GrowArmy.c ----- f*/
/*----- DocHeading */

```

```

extern int imin( int A, int B ) /* GrowArmy */
{
    if ( A <= B ) { return(A) ; }
    else         { return(B) ; }
}

/*----- GrowArmy.c ----- f*/

/*----- DocMethod */
extern int imax( int A, int B )
{
    if ( A <= B ) { return(B) ; }
    else         { return(A) ; }
}

/*-----InitStat.c ----- PickInt f*/
/*----- DocMethod */
extern int PickSome( int LowBound, int UpBound )
{
    int k ;
    double x, rX ;
    // double erand48();

    /* fprintf(stderr,"%d\n", UpBound ); fprintf( stderr, "%12.6f\n", rX ); */
    rX = erand48(GrowXsubi) ;
    x = (((double) UpBound) - ((double) LowBound)) * rX + 0.51 ; /* adjust away from 0.0 */
    x = x + ((double) LowBound) ;

    /* fprintf(stderr,"%10.6f\n", x );*/
    k = (int)x;
    if (k > UpBound) k = UpBound ;
    return( k );
}

extern struct Unit_Characteristics *FindUnit( int Id)
{
    struct Unit_Characteristics *UnCharLocated ;

    UnCharLocated = NULL ;

    if ( DirectToUnit[Id] != NULL ) {
        UnCharLocated = DirectToUnit[Id] ;
    }

    return( UnCharLocated ) ;
}
/* func_Name *c_Unit_Characteristics f*/
/*----- DocHeading */

```

30 June 1999

```

extern struct Unit_Characteristics *Grow_Unit_Characteristics(
    int *Level, int Side, char *sptr )
/*-----EndDocHead-----*/{
    struct Unit_Characteristics *tmp_ref;
    char tmpstr[128];
    int i;
    int PickInt();

    tmp_ref = (struct Unit_Characteristics *)malloc( sizeof( struct Unit_Characteristics ) );
    if ( errno == EINVAL ) {
        fprintf( stderr, " %s: (EINVAL) malloc has requested 0 bytes. \n", sptr );
    }
    if ( errno == ENOMEM ) {
        fprintf( stderr, " %s: (ENOMEM) not enough storage space was available. \n", sptr );
        exit(-1);
    }
    // strcpy(tmp_ref->Name, "\0" );
    tmp_ref->LastTime      = 0.0 ;
    tmp_ref->ReportRate    = 1.4425*60.0 * triangle( 0.92 ) ;
    tmp_ref->OrderRate     = 1.4425*60.0 * triangle( 0.92 ) ;
    tmp_ref->FireRate      = 1.802*60.0 * triangle( 0.92 ) ;
    tmp_ref->SenseRate     = 0.636*60.0 * triangle( 0.92 ) ;
    tmp_ref->MoveRate      = 0.778*60 * triangle( 0.92 ) ;
    tmp_ref->Environment   = 1.0 ;
    tmp_ref->Velocity      = 7.0 ;
    tmp_ref->Discovered    = 0 ; /* counters */
    tmp_ref->Updated       = 0 ; /* counters */
    tmp_ref->Interaction   = 0 ; /* counters */
    tmp_ref->Id            = Gid ;
                        Gid += 1 ;

    DirectToUnit[tmp_ref->Id] = tmp_ref ; /* quick reference to unit by Id */

    tmp_ref->Activity      = 0 ;
    tmp_ref->Force         = Side ;
    tmp_ref->Designation   = 0 ;
    tmp_ref->Subordinates  = 0 ;
    if ( *Level >= 4 ) {
        tmp_ref->Equipment = 4 ;
    }
    else {
        tmp_ref->Equipment = PickSome( 5, 5 ) * ( 8 - (*Level) );
    }
    tmp_ref->Personnel     = tmp_ref->Equipment * PickSome(3, 10) ;

    for(i=0; i< ObjectsInSOM ; i++) { tmp_ref->ObjectInstance[i] = 0 ; }
    for(i=0; i< ObjectsInSOM ; i++) { tmp_ref->Objects[i] = 0 ; }

    if ( *Level == 6 ) { tmp_ref->Objects[RouteSOM ] = 1 ; }
    if ( *Level < 6 ) { tmp_ref->Objects[MissionSOM ] = 1 ; }
    if ( *Level <= 3 ) { tmp_ref->Objects[UnitTypeSOM] = 1 ; } /* never updates*/
    if ( *Level < 6 ) { tmp_ref->Objects[PlanSOM ] = 1 ; }
    if ( *Level <= 6 ) { tmp_ref->Objects[HealthSOM ] = 1 ; }

    for(i=0; i< InteractionsInSOM ; i++) { tmp_ref->Interact[i] = 0 ; }

    if ( *Level < 6 ) { tmp_ref->Interact[OrderSOM ] = ORDER ; }
    if ( *Level <= 6 ) { tmp_ref->Interact[ReportSOM ] = REPORT ; }
    if ( *Level == 6 ) { tmp_ref->Interact[FireSOM ] = FIRE ; }
    if ( *Level == 6 ) { tmp_ref->Interact[SenseSOM ] = SENSE ; }
    if ( *Level == 6 ) { tmp_ref->Interact[SupplySOM ] = SUPPLY ; }

    tmp_ref->Rolled_Equipment = 1 ;
    tmp_ref->Rolled_Personnel = 1 ;
    tmp_ref->DataSize        = 0 ;
    tmp_ref->Echeleon        = *Level ; /* 0-Army,1-Corp,2-Div,3-Bri,4-Bn,5-Co,6-Plt */

```

30 June 1999

```

tmp_ref->CpuNode      = 0 ;
tmp_ref->Controller    = 0 ;
tmp_ref->GlobalId     = 0 ;
tmp_ref->Federate      = 0 ;
tmp_ref->FedNode       = 0 ;
tmp_ref->ViewHoriz     = 0 ;
tmp_ref->ViewVert      = 0 ;
tmp_ref->ViewColor     = *Level + 10 * (Side-1) + 50 ;
tmp_ref->ViewState     = *Level + 10 * (Side-1) + 50 ;
tmp_ref->InLstp        = NULL ;
tmp_ref->CmNetp        = NULL ;
tmp_ref->Truthp        = NULL ;
tmp_ref->SvStkp        = NULL ;
tmp_ref->AltSvLp       = NULL ;
tmp_ref->ServLp        = NULL ;
tmp_ref->ULstp         = NULL ;
tmp_ref->RegOfUnit     = NULL ;
tmp_ref->ngep          = NULL ; /* tmp_ref ; */
Guide[*Level].Ent += tmp_ref->Equipment ;
Guide[*Level].Cnt += 1 ;
sprintf(tmpstr,"%s%04d-%02d-%04d", Guide[*Level].EchName, Guide[*Level].Cnt, *Level, tmp_ref-
>Id );
strcpy( tmp_ref->Name, tmpstr );

// printf("Q"); for(i=0; i<=*Level; i++ ) { printf("."); } printf("%s\n", tmp_ref->Name );

return (tmp_ref);
}

/*----- GrowArmy.c ----- Grow_Unit List f*/
/*----- DocHeading */

```

30 June 1999

```

extern struct Unit_List *Grow_Unit_List( char *sptr )
/*-----EndDocHead---*/
{
    struct Unit_List *tmp_ref;
    tmp_ref = (struct Unit_List *) malloc( sizeof( struct Unit_List ) ) ;
    if ( errno == EINVAL ) {
        fprintf( stderr, " %s: (EINVAL) malloc has requested 0 bytes. \n", sptr );    }
    if ( errno == ENOMEM ) {
        fprintf( stderr, " %s: (ENOMEM) not enough storage space was available. \n", sptr);
        exit(-1);    }
    tmp_ref->UChrp    = NULL ;    /* Unit_Characteristics type */
    tmp_ref->nrep      = NULL ;
    tmp_ref->UCmdp     = NULL ;
    tmp_ref->Subrdp    = NULL ;

    return (tmp_ref);
}

/*----- GrowArmy.c ----- Grow_Echeleon f*/
/*----- GrowArmy.c ----- Grow_Echeleon f*/
/*----- GrowArmy.c ----- Grow_Echeleon f*/
/*----- DocHeading */

```

30 June 1999

```

extern void Grow_Echeleon( FILE      *out,
                          int        NumberOfPeers,
                          int        Side,
                          struct Unit_List *Cmdr,
                          struct Unit_List **Ulp,
                          struct Unit_Characteristics **UnCharG,
                          int        *Level)
/*-----EndDocHead-----*/
{
    struct Unit_Characteristics *UnChp, *Top, *pTop;
    struct Unit_List            *UnLtp;
    int NumOfSubordinate;
    int i ;

    i = 0 ;
    UnLtp = NULL ;

    do {
        i += 1 ;
        //printf("Q i %2d      Peers %3d  %s Tot %5d ", i, NumberOfPeers,
        //      Guide[*Level].EchName, Guide[*Level].TotMax );
        if ( *Ulp == NULL ) {
            *Ulp = Grow_Unit_List(" now ");
            UnLtp = *Ulp ;
        }
        else {
            UnLtp->nrep = Grow_Unit_List(" now ");
            UnLtp = UnLtp->nrep ;
        }
        UnLtp->UCmdp = Cmdr ; /* set the reference to the commander of the unit */
        if ( *UnCharG == NULL ) {
            *UnCharG = Grow_Unit_Characteristics( Level, Side, " now ");
            UnChp = *UnCharG ;
        }
        else {
            UnChp->ngep = Grow_Unit_Characteristics( Level, Side, " now ");
            UnChp = UnChp->ngep ;
        }
        UnChp->ULtp = UnLtp;
        UnLtp->UChrp = UnChp;

        // printf("peers Top %8.8x %21s \n", UnChp, UnChp->Name );

        if ( *Level <  LevelLimit ) { /* make some subordinates */
            *Level += 1;
            if ( Guide[*Level].TotMax <= 5 ) {
                if ( FIXCOMMANDERS > 0 ) { NumOfSubordinate = FIXCOMMANDERS ; }
                else {
                    NumOfSubordinate = Guide[*Level].TotMax ;
                }
            }
            else { /* NumOfSubordinate = 5; */
                NumOfSubordinate = PickSome( Guide[*Level].MinSub, Guide[*Level].MaxSub );
                if ( *Level == 6 ) {
                    if ( FIXSUBORDINATES > 0 ) { NumOfSubordinate = FIXSUBORDINATES; }
                }
                if ( NumOfSubordinate + Guide[*Level].Cnt > Guide[*Level].TotMax ) {
                    NumOfSubordinate = Guide[*Level].TotMax - Guide[*Level].Cnt ;
                }
            }
            Guide[*Level-1].Sub += NumOfSubordinate;

            Grow_Echeleon( out, NumOfSubordinate, Side, UnLtp, &UnLtp->Subrdp, &UnChp->ngep, Level
        );
        Top = UnChp->ngep;
    }
}

```

30 June 1999

```

        // printf("      Top %8.8x %21s ", Top, Top->Name );
while ( Top != NULL ) { pTop = Top; Top = Top->ngep; } /* have to advance to end */
UnChp = pTop ;
        // printf("pTop %8.8x %21s \n", pTop, pTop->Name );
//      UnLtp->Subrdp->UCmdp = UnLtp;          /* point to commander */
*Level -= 1 ;          // printf("|%s|",UnLtp->Subrdp->UCmdp->UChrp->Name );
    }

// printf("i:%2d, Lvl %2d, UnLtp, %8.8x, *Ulp, %8.8x, UnCharG %8.8x * %8.8x\n",
//      i, *Level,UnLtp, *Ulp, UnCharG, *UnCharG );

} while ( i < NumberOfPeers || (*Level == 4 && Guide[4].TotMax >= Guide[4].Cnt &&
        Guide[3].TotMax <= Guide[3].Cnt ) ) ;

/* printf("Q i %2d      .Peers %3d  %s Tot %5d  Cnt %5d \n",i, NumberOfPeers,
        Guide[*Level].EchName,Guide[*Level].TotMax,
        Guide[*Level].Cnt);

*/

} /* end Grow_Echeleon */

/*----- GrowArmy.c ----- Grow_Echeleon f*/
/*----- GrowArmy.c ----- Grow_Echeleon f*/
/*----- GrowArmy.c ----- Grow_Echeleon f*/

/*----- DocHeading      Grow */

```

30 June 1999

```

extern void GrowInitArmy( int GreenBattalions, int OtherBattalions,
                        struct Region_Node_Handle *RegionNodeHandles)
{
/*-----EndDocHead---*/

extern void PrintRegions( FILE *out, /* GrowArmy */
                        struct Region_List *RegList,
                        char *Emark );
extern void PrintFilterList( FILE *out,
                        struct Filter_Unit_List *FilteredList,
                        char *Emark );

struct Filter_Unit_List *FilteredUnits, *FilteredList, *FilteredNum ;

struct Unit_Characteristics *UnCharG, *UnCharO;
struct Unit_Characteristics *LastGnUnitAssigned, *LastOtUnitAssigned ;
struct Unit_Characteristics *UnChar, *UnCharX;
FILE *out;
//FILE *in_orders;
//struct Region_Node_Handle *RegionNodeHandles;
struct Region_List *RegList ;
struct Region_Element_List *ElementOfRegion, *pRegEle ;

int EquipOtatLevel, EquipGnatLevel;
int HighGn, HighOt, LowGn, LowOt;
int i,j,k,l,m,n,o,p,q,r,s,t,u,v ;

//FILE *in_orders;

//double dtemp, c_time ;
//char justtext[128];
char str[128] ;

void ResetEcheleon();

i=j=k=l=m=n=o=p=q=r=s=t=u=v=1;
i=j+k+l+m+n+o+p+q+r+s+t+u+v;

/* LalaInit(1,1); only when single testing */

Total_Force_Battalions = GreenBattalions ;

for( i=0; i< LimitOnUnits; i++) { DirectToUnit[i] = NULL ; }

/* Army->Corps->Division->Brigade->Battalion->Company->Platoon->Squad */
for (i=0; i<8; i++) {
    Guide[i].Cnt = 0;
    Guide[i].TotMax = 0;
    Guide[i].CanDo = 1;
    Guide[i].Ent = 0;
    Guide[i].Sub = 0 ;
    Guide[i].MinSub = 2;
    Guide[i].MaxSub = 5;
    if ( i > 3 ) { Guide[i].MinSub = 4; Guide[i].MaxSub = 7; }; printf(" Guide \n");
    if (i == 0) { strcpy( Guide[i].EchName, "ArmB" ); }
    if (i == 1) { strcpy( Guide[i].EchName, "CrpB" ); }
    if (i == 2) { strcpy( Guide[i].EchName, "DivB" ); }
    if (i == 3) { strcpy( Guide[i].EchName, "BriB" ); }
    if (i == 4) { strcpy( Guide[i].EchName, "BatB" ); }
    if (i == 5) { strcpy( Guide[i].EchName, "ComB" ); }
    if (i == 6) { strcpy( Guide[i].EchName, "PltB" ); }
    if (i == 7) { strcpy( Guide[i].EchName, "SqdB" ); }
}
Guide[7].TotMax = 99999 ;
Guide[6].TotMax = 99999 ;

```


30 June 1999

```

Guide[5].TotMax = 99999 ;
Guide[4].TotMax = Total_Force_Battalions ;
Guide[3].TotMax = Total_Force_Battalions / 4; /* (PickSome(4, 4)) ;   totals to create */
Guide[2].TotMax = imax( 1, ( Guide[3].TotMax / 2 ) ) ;
Guide[1].TotMax = imax( 1, ( Guide[2].TotMax / 2 ) ) ;
Guide[0].TotMax = imax( 1, ( Guide[1].TotMax / 2 ) ) ;

printf("Q A %3d, Cp %3d, Dv %3d, Bg %3d, Bn %3d \n",Guide[0].TotMax, Guide[1].TotMax,
      Guide[2].TotMax,Guide[3].TotMax, Guide[4].TotMax);

//}
/*   for # of Army           */
/*   for # of Corps          */
/*   for # of Divisions      */
/*   for # of Brigades       */
/*   for # of Battalions     */
/*   for # of Companies       */
/*   for # of Platoons       */
/*   for # of Squads         */
printf("Grow_Unit_list\n");
UnCharG = NULL ;
UnLstG = NULL ;
gLevel = 0;
printf("      %8.8x \n", UnLstG );
/* the commander is NULL for top echeleon; Truly God is in control*/
/* FILE, NumOfPeers, Side, Commander(UnitList), UnitList, UnitChar, Echeleon
*/
Grow_Echeleon( stderr, Guide[0].TotMax, 1, NULL, &UnLstG, &UnCharG, &gLevel); /* grow an Army
*/
printf("press enter AFTER GROW ECHELEON \n");
gets(str);

for(i=0; i<8; i++) {
  printf("Gfor   Guide  %2d Cnt %4d Sub %4d Max %4d Ent %6d \n",
    i, Guide[i].Cnt, Guide[i].Sub, Guide[i].TotMax, Guide[i].Ent );
}
Total_Force_Battalions = OtherBattalions ;

oLevel = 0;
for (i=0; i<8; i++) {
  Guide[i].Cnt = 0;
  Guide[i].TotMax = 0;
  Guide[i].CanDo = 1;
  Guide[i].Ent = 0;
  Guide[i].Sub = 0 ;
  Guide[i].MinSub = 2;
  Guide[i].MaxSub = 5;
  Guide[i].EchName[3] = 'Z' ;
  if ( i > 3 ) { Guide[i].MinSub = 4; Guide[i].MaxSub = 7; }; printf(" Guide \n");
}
Guide[7].TotMax = 99999 ;
Guide[6].TotMax = 99999 ;
Guide[5].TotMax = 99999 ;
Guide[4].TotMax = Total_Force_Battalions ;
Guide[3].TotMax = Total_Force_Battalions / 4; /* ( PickSome(4, 4) ;   totals to create */
Guide[2].TotMax = imax( 1, ( Guide[3].TotMax / 2 ) ) ;
Guide[1].TotMax = imax( 1, ( Guide[2].TotMax / 2 ) ) ;
Guide[0].TotMax = imax( 1, ( Guide[1].TotMax / 2 ) ) ;

UnCharO = NULL ;
UnLstO = NULL ;
oLevel = 0;
/* the commander is NULL for top echeleon; Truly God is in control*/
Grow_Echeleon( stderr, Guide[0].TotMax, 2, NULL, &UnLstO, &UnCharO, &oLevel); /* grow an Army
*/

printf("\nFirst| %8.8x %8.8x \n", UnLstG, UnLstG->nrep );

```

```

for(i=0; i<8; i++) {
    printf("Ofor    Guide  %2d Cnt %4d  Sub %4d  Max %4d Ent %6d \n", i, Guide[i].Cnt,
    Guide[i].Sub,
        Guide[i].TotMax,  Guide[i].Ent );
}
printf("Q A %3d, Cp %3d, Dv %3d, Bg %3d, Bn %3d \n",Guide[0].TotMax, Guide[1].TotMax,
        Guide[2].TotMax, Guide[3].TotMax, Guide[4].TotMax );

    Print_Echeleon( stdout, UnLstG );
gets(str);
    Print_Echeleon( stdout, UnLstO );
gets(str);

//out = fopen("GrownUnit.dat", "w");
    printf("      %8.8x ", UnLstG );
    printf("      %s \n", UnLstG->UChrp->Name );

//printf("press enter AFTER GROW ECHELEON \n");
//gets(str);

TallyClearEch();
TallyEcheleon( UnLstG );
TallyPrintEch( stdout, " Green  ");

printf(" Total Equipment:  %7d \n", (GetTotalEquip()) );
EquipGnatLevel = GetTotalEquipByLevel(  MaxEcheleon( NULL, UnCharG, "Green Tally for
EquipatLevel" ) );
EquipGnatLevel += GetTotalEquipByLevel( (MaxEcheleon( NULL, UnCharG,"Green Tally") -1) );
printf(" Green Equip %5d \n", EquipGnatLevel );

TallyClearEch();
TallyEcheleon( UnLstO );
TallyPrintEch( stdout, " Other  ");
printf(" Total Equipment:  %7d \n", (GetTotalEquip()) );

EquipOtatLevel = GetTotalEquipByLevel(  MaxEcheleon( NULL, UnCharO, "Other Tally for
EquipatLevel" ) );
EquipOtatLevel += GetTotalEquipByLevel( (MaxEcheleon( NULL, UnCharO,"Other Tally") -1) );
printf(" Other Equip %5d \n", EquipOtatLevel );
gets(str);

Regions.high = 1;
Regions.Commit = .25 ;
Regions.Ratio = 1.0 ;
Regions.other = 1 ;    /* for each force */

TallyClearEch();
TallyEcheleon( UnLstG );
Nodes.TotalGnEquip =  GetTotalEquipByLevel(4) +
        GetTotalEquipByLevel(5) + GetTotalEquipByLevel(6) ;

TallyClearEch();
TallyEcheleon( UnLstO );
Nodes.TotalOtEquip =  GetTotalEquipByLevel(4) +
        GetTotalEquipByLevel(5) + GetTotalEquipByLevel(6) ;

printf("Start intended setup Green High %5d Low %5d, TotalEquip %5d, At level %5d\n",
        Nodes.HighGn,  Nodes.LowGn, Nodes.TotalGnEquip, EquipGnatLevel );
printf("Start intended setup OTHER High %5d Low %5d, TotalEquip %5d, At level %5d\n",
        Nodes.HighOt, Nodes.LowOt, Nodes.TotalOtEquip, EquipOtatLevel );
gets(str);
RegionNodeHandles->RegionsDefined = &Regions ;
RegionNodeHandles->NodesDefined   = &Nodes ;
RegionNodeHandles->xtFed = NULL ;

```

30 June 1999

```

RegionNodeHandles->xtReg =    NULL ;
RegionNodeHandles->UnitGn =   UnCharG ;
RegionNodeHandles->UnitOt =   UnCharO ;
RegionNodeHandles->UnListGn = UnLstG ;
RegionNodeHandles->UnListOt = UnLstO ;
/*                                CREATE REGIONS */

CreateRegions(stdout, EquipGnatLevel, EquipOtatLevel, &Regions, RegionNodeHandles);

printf("returned %8.8x \n", RegionNodeHandles->xtReg );
/* for each region */

RegList = RegionNodeHandles->xtReg ;
LastGnUnitAssigned = UnCharG ;
LastOtUnitAssigned = UnCharO ;
ElementOfRegion = NULL;
/*----- should CUT into a method -----*/
/*----- should CUT into a method -----*/
RegList = RegionNodeHandles->xtReg ;
/* fill Geo Regions    FILL */
do {
    /* fill Geo Regions    FILL */
    printf("RegionFILL %2d  %4d %4d ", RegList->Id, RegList->EquipGn, RegList->EquipOt );
    if ( RegList != NULL ) {

        EquipGnatLevel = RegList->EquipGn ;
        if ( LastGnUnitAssigned != NULL && EquipGnatLevel > 0 ) {
            ElementOfRegion = NULL;

            LastGnUnitAssigned = FillRegion( stdout,
                                             EquipGnatLevel, &RegList->EquipGn,
                                             RegList, LastGnUnitAssigned, &ElementOfRegion);
            RegList->xtGnRegEle = ElementOfRegion;

            //      pRegEle = ElementOfRegion;
            //      printf("G  ->xtGnRegEle    %s", pRegEle->UChrp->Name);

        }

        EquipOtatLevel = RegList->EquipOt ;
        if ( LastOtUnitAssigned != NULL && EquipOtatLevel > 0 ) {
            ElementOfRegion = NULL;

            LastOtUnitAssigned = FillRegion( stdout,
                                             EquipOtatLevel, &RegList->EquipOt,
                                             RegList, LastOtUnitAssigned, &ElementOfRegion);
            RegList->xtOtRegEle = ElementOfRegion;

            //      printf("O  RegList->xtOtRegEle    %s", RegList->xtOtRegEle->UChrp->Name);
        }
        printf("Region %2d  %4d %4d resulting \n", RegList->Id, RegList->EquipGn, RegList->EquipOt );
        RegList = RegList->xtReg;
    } while ( RegList != NULL); /* end of list */

    /*----- should CUT into a method -----*/
    /*----- should CUT into a method -----*/

    RegList = RegionNodeHandles->xtReg ;

    HighGn = HighOt = LowGn = LowOt = 0;
    do {
        /* fill Geo Regions    FILL */
        /* count units by category */
        if ( RegList->Category == 1 ) {
            HighGn += RegList->EquipGn ;
            HighOt += RegList->EquipOt ;
        }
        if ( RegList->Category == 2 ) {
            LowGn += RegList->EquipGn ;
            LowOt += RegList->EquipOt ;
        }
    }

```

```

    RegList = RegList->xtReg;
}    while ( RegList != NULL);    /* end of list */

RegList = RegionNodeHandles->xtReg;

printf("press enter High %5d %5d    Low %5d %5d \n",HighGn, HighOt, LowGn, LowOt );
printf("press enter High %5d %5d    Low %5d %5d \n",HighGn, HighOt, LowGn, LowOt );
//gets(str);

Nodes.HighGn = HighGn ;
Nodes.HighOt = HighOt ;
Nodes.LowGn = LowGn ;
Nodes.LowOt = LowOt ;
//Nodes.TotalGnEquip = HighGn + LowGn;          /* CREATE NODES */
//Nodes.TotalOtEquip = HighOt + LowOt ;          /* CREATE NODES */
Nodes.High = SCENARIOHigh;                      /* CREATE NODES */
Nodes.HighEquipment = (HighGn + HighOt) / Nodes.High ; /* CREATE NODES */
Nodes.Low = SCENARIOLow ;
Nodes.LowEquipment = (LowGn + LowOt) / Nodes.Low ; /* CREATE NODES */

//PrintRegions( stdout, RegList, "GeoRegion1Before\n") ; /* CREATE NODES */

RegList = RegionNodeHandles->xtReg;

// PrintRegions( stdout, RegList, "GeoRegion1\n") ;

RegList = RegionNodeHandles->xtReg;

//FilteredUnits = CmdUnitNotInRegion(RegList->xtGnRegEle); /* COMMANDERS */

//printf("press enter CmdUnitNotInRegion( RegList->xtGnRegEle); /* GREEN COMMANDERS */ \n");
//PrintFilterList( stdout, FilteredUnits, "Greens Cmd\n");
//d_Filter_Unit_List( FilteredUnits ); /* Free allocated memory */

//FilteredUnits = CmdUnitNotInRegion(RegList->xtOtRegEle); /* COMMANDERS */

//printf("press enter CmdUnitNotInRegion( RegList->xtOtRegEle); /* OTHER COMMANDERS */ \n");
//PrintFilterList( stdout, FilteredUnits, "Others Cmd\n");
//d_Filter_Unit_List( FilteredUnits ); /* Free allocated memory */

/* ----- Build the overlapping regions */
/* ----- Build the overlapping regions */
/* ----- Build the overlapping regions */

RegList = RegionNodeHandles->xtReg; /* region 1 */

//FilteredUnits = PutAllInRegBySideInFilterList(1,RegList->xtGnRegEle); /* Side 1 */
i = RegList->EquipGn / 2;
printf("Region found %s    Equip %4d    select %4d\n", RegList->Name, RegList->EquipGn, i );

FilteredUnits = PutNumInRegBySideInFilterList(i,1,RegList->xtGnRegEle);

//sprintf( str,"Side 1 Reg %2d\n", RegList->Id );
//PrintFilterList( stdout, FilteredUnits, str);

RegList = FindRegion( 2, RegionNodeHandles->xtReg ) ; /* for region # 2 */

i = RegList->EquipGn / 2;
printf("Region found %s    Equip %4d    select %4d\n", RegList->Name, RegList->EquipGn, i );

FilteredNum = PutNumInRegBySideInFilterList(i,1,RegList->xtGnRegEle);

MergeFilterList( FilteredUnits, FilteredNum );

sprintf( str,"merged 1 Reg %2d\n", RegList->Id );

```

```

//PrintFilterList( stdout, FilteredUnits, str);
printf("Region found %s Equip %4d select %4d\n", RegList->Name, RegList->EquipGn, i );
printf("press enter at FindRegion\n");
//gets(str);

RegList = RegionNodeHandles->xtReg; /* region 1 */

AddNewRegion( NULL, 'G',RegList, FilteredUnits ); /* CREATE REGION 4 */

RegList = FindRegion( 4, RegionNodeHandles->xtReg ); /* for region # 4 */

//Print_Echeleon(stdout, UnLstG );
printf("Finished Adding region 4 %s Equip %4d select %4d\n", RegList->Name, RegList-
>EquipGn, i );
//printf("press enter at FindRegion\n");
//gets(str);
d_Filter_Unit_List( FilteredUnits ); /* Free allocated memory */
/* ----- Added GREEN force support units */

RegList = RegionNodeHandles->xtReg; /* region 1 */

//FilteredUnits = PutAllInRegBySideInFilterList(2, RegList->xtOtRegEle); /* Side 2 */
i = RegList->EquipOt / 2;
FilteredUnits = PutNumInRegBySideInFilterList(i,2,RegList->xtOtRegEle);

sprintf( str,"Side 2 Reg %2d\n", RegList->Id );
//PrintFilterList( stdout, FilteredUnits, str );

RegList = FindRegion( 3, RegionNodeHandles->xtReg ); /* for region # 2 */

i = RegList->EquipOt / 2;
printf("Region found %s Equip %4d select %4d\n", RegList->Name, RegList->EquipOt, i );

FilteredNum = PutNumInRegBySideInFilterList(i,2,RegList->xtOtRegEle);
//PrintFilterList( stdout, FilteredNum, str);

MergeFilterList( FilteredUnits, FilteredNum );
sprintf( str,"merged 2 Reg %2d\n", RegList->Id );
//PrintFilterList( stdout, FilteredUnits, str);
printf("Region found %s Equip %4d select %4d\n", RegList->Name, RegList->EquipOt, i );
printf("press enter at FindRegion\n");
//gets(str);

RegList = RegionNodeHandles->xtReg; /* region 1 */

AddNewRegion( NULL, 'O',RegList, FilteredUnits ); /* CREATE REGION 5 */
RegList = FindRegion( 5, RegionNodeHandles->xtReg ); /* for region # 5 */
/*AddNodeLinksToRegionsByUnit( stdout,
UnCharO,
RegList,
RegionNodeHandles->xtFed );

*/
//Print_Echeleon(stdout, UnLstO );
printf("Finished Adding region 5 %s Equip %4d select %4d\n", RegList->Name, RegList-
>EquipGn, i );
printf("press enter at FindRegion\n");
//gets(str);
d_Filter_Unit_List( FilteredUnits ); /* Free allocated memory */

/* ----- Added OTHER force support units */

CreateNodes( stdout, RegionNodeHandles ); /* Nodes */

Print_Echeleon( stdout, UnLstG );
Print_Echeleon( stdout, UnLstO );

```

```

PrintNodesOfFed( stdout, RegionNodeHandles->xtFed, " :By Nodes\n");
PrintRegionsNodes( stdout, RegionNodeHandles->xtReg, " :byRegions\n") ;

printf("intended setup Green High %5d Low %5d, TotalEquip %5d,\n",
      Nodes.HighGn, Nodes.LowGn, Nodes.TotalGnEquip );
printf("intended setup OTHER High %5d Low %5d, TotalEquip %5d,\n",
      Nodes.HighOt, Nodes.LowOt, Nodes.TotalOtEquip );
printf("Inteded setup for High Nodes: %3d equip per %5d \n",
      Nodes.High, Nodes.HighEquipment );
printf("Inteded setup for Low Nodes: %3d equip per %5d \n",
      Nodes.Low, Nodes.LowEquipment );

printf("press enter High %5d %5d Low %5d %5d \n",HighGn, HighOt, LowGn, LowOt );

printf("press enter at finish of construction of forces \n");
gets(str);
//PrintUnitsOfFed( stdout, RegionNodeHandles->xtFed, "list units\n" );

/*----- */
// LalaInit(1,1);
//sleep(1);

ViewNew();
//ViewEcheleonLeft( UnLstG );
ViewNew();
ViewNext();
//ViewEcheleonRight( UnLstO );

//Print_Echeleon( out, UnLstG );

//Print_Echeleon( out, UnLstO );
// fclose( out );
ViewNext();
//gets(str);
ViewRefresh( UnLstG );
//sleep(1);
UnChar = UnCharG ;
// do {
//     printf("%5d %-21s %5d %4d %3d\n", UnChar->Id, UnChar->Name,
//     UnChar->ViewHoriz, UnChar->ViewVert, UnChar->Echeleon );
//     LalaPlace( UnChar->ViewState, UnChar->ViewHoriz, UnChar->ViewVert );
//     UnChar = UnChar->ngep;
// } while ( UnChar != NULL );

GrowInterestGroup( UnCharG, UnCharO, 5 ); /* from, to, Max */
//Print_InterestGroup(stdout, UnCharG );
//Draw_InterestGroup( UnCharG );
//sleep(2);
GrowInterestGroup( UnCharO, UnCharG, 5 );
//Print_InterestGroup(stdout, UnCharO );
//Draw_InterestGroup( UnCharO );
//gets(str);
//out = fopen("GUnitIM.dat", "w");
//Print_InterestGroup(out, UnCharG );
//fclose(out);

//out = fopen("GUnitIM.dat", "w");
//Print_InterestGroup(out, UnCharO );
//fclose( out );

UnCharX = UnCharG ;
UnChar = UnCharO ;
//do {
//     printf("%5d %-21s %-21s %5d %4d %3d\n", UnChar->Id, UnChar->Name, UnCharX->Name,

```

30 June 1999

```

//          UnChar->ViewHoriz, UnChar->ViewVert, UnChar->Echeleon );
//
//      LalaPlace( UnChar->ViewColor,      UnChar->ViewHoriz, UnChar->ViewVert );
//
//      UnChar  = UnChar->ngep;
//      UnCharX = UnCharX->ngep;
//
//  } while ( UnChar != NULL ) ;
/* - */
printf("press enter  After Developing an Interest Group\n");
gets(str);

} /* end of grow main */

/* ----- DISTRIBUTIION ----- */
/*----- DocMethod */
extern double triangle(double c)
{
double dX, dU ;

    dU = erand48(TriangleRandU);

    if (dU < c ) { dX = sqrt( c*dU) ;          }
    else {        dX = 1 - sqrt((1-c)*(1-dU) ); }

    return(dX);
}
/*----- DocHeading */

```

```
/* file: InterestGroups.c */
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <math.h>
#include "soa_defs.h"
#include "soa_cnst.h"
```

```
/*----- InterestGroups.c ----- GrowInterestGroup f*/
/* This following stuff pertains to subsets with a region that is maintained by */
/* by the unit itself. Benefits pertain to extened entity state data. */
/* such as predictive contracts. */
/* 'If Unit knows those processed for a time interval then some local queries can */
/* can be avoided.' */
```

```
/*----- DocHeading */
```


30 June 1999

```

extern void GrowInterestGroup( struct Unit_Characteristics *UnCrA,
                             struct Unit_Characteristics *UnCrB, int MaxInList)
/*-----EndDocHead---*/
{
    struct Unit_Characteristics *A, *B, *T;
    struct InterestList      *Ip, *nIp ;
    struct InterestList      *c_InterestList(char *s);
    int MaxA_Ech, MaxB_Ech, InList, Account;
    extern int MaxEcheleon(FILE *o, struct Unit_Characteristics *A , char *s);
    int LimitEndOfB ;
    LimitEndOfB = 0;

    MaxA_Ech = MaxEcheleon( NULL, UnCrA , " ForceA ");
    MaxB_Ech = MaxEcheleon( NULL, UnCrB, " ForceB " );
    A = UnCrA;
    B = UnCrB;
    printf(" The maximum Echelons are %1d, %1d \n", MaxA_Ech, MaxB_Ech );
    Account = 0 ;
    while ( B->Echeleon < MaxB_Ech-1) { B = B->ngep ; }
    do {
        if ( A->Echeleon == MaxA_Ech && A->Designation == WarFighter ) {
            Account += 1;
            T = B ; InList = 0 ;
            do {
                if ( (B->Echeleon == MaxB_Ech || B->Echeleon == MaxB_Ech-1) &&
                    B->Designation == WarFighter ) {
                    Ip = c_InterestList( "Build Interest\n" );
                    Ip->UChrp = B ;
                    if ( A->InLstp == NULL ) { A->InLstp = Ip ; nIp = Ip ; }
                    else { nIp->ngep = Ip ; nIp = Ip ; }
                    if ( A->Echeleon == MaxA_Ech && B->Echeleon == MaxB_Ech ) {
                        InList +=1 ;
                    }
                }
                B = B->ngep ;
            } while ( InList <= MaxInList && B != NULL ) ;
            // nIp->ngep = A->InLstp ;
            B = T ;

            if ( Account >= MaxInList-1 ) { B = B->ngep ; /* allow advance */
                while ( B != NULL && B->Echeleon < MaxB_Ech ) { B = B->ngep ; }
            }
            if ( B == NULL ) { LimitEndOfB += 1; }
        }
        A = A->ngep ;
    } while ( A != NULL && B != NULL && LimitEndOfB < MaxInList );
}
/* if the MaxInList is used as a function for selection and assignment to sectors
   then we have the DDM regions. A selection function should account for density
   of units with regards to an activity level */

/*----- print Interest */
/*----- DocHeading */

```

```

extern void Print_InterestGroup(FILE *out, struct Unit_Characteristics *UnCrA )
/*-----EndDocHead---*/
{
    struct Unit_Characteristics *A;
    struct InterestList      *nIp ;
    int i ;
    i = 0;
    A = UnCrA;

    do {
        fprintf(out, "%4d %-19s: ", i++, A->Name );
        if ( A->InLstp != NULL ) { /* refined interest list within IML FQL */
            nIp = A->InLstp ;
            do { fprintf(out, "%4d:%17s,", i++, nIp->UChrp->Name );
                nIp = nIp->ngep ;
            } while ( nIp != NULL ) ;
        }
        fprintf(out, "\n" );
        A = A->ngep ;
    } while ( A != NULL );
}

/*----- DocMethod */
extern void Draw_InterestGroup(struct Unit_Characteristics *UnCrA )
/*-----EndDocHead---*/
{
    struct Unit_Characteristics *A ;
    struct InterestList      *nIp ;
    extern void LalaDrawLink( int State, int x1, int y1, int x2, int y2);
    int Xcolor ;
    A = UnCrA;

    do {
        // printf("%-19s: ", A->Name );
        if ( A->InLstp != NULL ) {
            nIp = A->InLstp ;
            do { /* printf("%17s,", nIp->UChrp->Name ); */
                if ( A->RegOfUnit != NULL ) { Xcolor = (A->Force-1) * 10 + A->RegOfUnit->xtReg-
>Id ;}
                else {Xcolor = A->Force ; }
                LalaDrawLink( Xcolor, A->ViewHoriz, A->ViewVert ,
                    nIp->UChrp->ViewHoriz, nIp->UChrp->ViewVert );
                nIp = nIp->ngep ;
            } while ( nIp != NULL ) ;
        }
        A = A->ngep ;
    } while ( A != NULL && A != UnCrA);
}

/*----- DocHeading */

```

30 June 1999

```
/* file: ReadOrdr.c */
/*****
--* int load_orders( char *filename, struct comm_pkt_ *pkt ) builds order list
--* * file load_ord.c
*****/
#include "soa_defs.h"
#include "soa_cnst.h"

#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <stdio.h>

/* FILE, *pkt */
int UniqueMsgId = 0 ;
extern struct Order_Packet *c_Order_Packet(char *s);
extern void Print_Order(FILE *out, struct Order_Packet *OPpkt, char *Emark );
extern int MsgIdTag(struct Order_Packet *pkt);
extern int csv(char *lstr, char *pieces[], char *delimiter);

/*----- ReadOrdr.c ----- ReadOrder f*/
/*----- DocHeading */
```

```

extern int ReadOrder( FILE *fptr, struct Order_Packet *pkt )
{
    int i,j ;
    // FILE *in ;
    char line[512];
    // extern int csv(char *lstr, char *pieces[], char *delimiter);
    int rstat ;
    char *list[128];
    //struct Order_Packet *lpkt ;
    // extern void Print_Order(FILE *out, struct Order_Packet *OPpkt, char *Emark );
    // extern int MsgIdTag(struct Order_Packet *pkt);

    /* printf( "fptr %08x \n", fptr ); */

    if ( fptr == NULL ) {
        printf( " load_orders: null File ptr \n");
        return(-1);
    }
    else {
        rstat = (int)fgets( line, 512, fptr ); line[ (sizeof(line) - 1) ] = 0 ;
        /* printf("r: %d, %s \n", rstat, line); */

        if ( rstat == NULL ) {
            fprintf( stderr, " ReadOrder: End Of File \n" );
        }
        else {
            j = csv( &line[0], list, "," ); /* should probably do while */

            if ( j > 0 ) {
                pkt->PcktType = CmdOrder ;
                pkt->TimeToSend = atof( list[0] );
                pkt->TimeToAct = atof( list[1] );
                pkt->ToActInHrs = atof( list[6] );
                pkt->Order = atoi( list[2] );
                pkt->Size = MaxSizeMsg ;
                pkt->Entities = 0.0 ;
                pkt->Priority = 0 ;
                pkt->Activity = 0 ; /* reassess for Order type */
                if ( strcmp(list[3], "b") == 0 ) { pkt->Force = Friends ; } /* green */
                else { pkt->Force = Others ; }
                pkt->To = atoi( list[4] );
                pkt->From = -1 ;
                pkt->nqep = NULL ;
                pkt->MsgId = MsgIdTag( NULL );
                for ( i=0; i<j; i++) { free( list[i] ); }
            }
        }

        /* potentially valid file ptr */
        return(rstat);
    } /* end of load_orders */

    /* ----- ReadOrder.c ----- f*/
    /*----- DocHeading */

```

30 June 1999

```

extern struct Order_Packet *MakeOrder(int aPcktType,
    int aForce,
    int aOrder,      /* MakeOrder f*/
    int aActivity,
    int aTo,         /* MakeOrder f*/
    int aFrom,
    int aDestSP,     /* MakeOrder f*/
    int aPriority,    /* MakeOrder f*/
    double aSize,
    double aEntities, /* MakeOrder f*/
    double aTimeToSend,
    double aTimeToAct ) /* MakeOrder f*/
{
    struct Order_Packet *pkt ;
    // extern struct Order_Packet *c_Order_Packet(char *s);
    // int MsgIdTag();

    pkt = c_Order_Packet( "MakeOrder" );

    if ( pkt != NULL ) {
        pkt->PcktType      = aPcktType      ;
        pkt->TimeToSend     = aTimeToSend ;
        pkt->TimeToAct      = aTimeToAct ;
        pkt->ToActInHrs     = aTimeToAct/3600.0 ;
        pkt->Order          = aOrder ;
        pkt->Size           = aSize ;
        pkt->Entities       = aEntities ;
        pkt->Priority        = aPriority ;
        pkt->Activity       = aActivity ; /* reassess for Order type */
        pkt->Force          = aForce ;
        pkt->To             = aTo ;
        pkt->From           = aFrom ;
        pkt->DestSP         = aDestSP ;
        pkt->MsgId          = MsgIdTag( NULL );
        pkt->nqep           = NULL ;

    }
    return( pkt ); /* potentially valid file ptr */
} /* end of MakeOrder */

/* ----- ReadOrdr.c ----- DuplicateOrder f*/
/* ----- DocHeading */

```

```
extern struct Order_Packet *DuplicateOrder( struct Order_Packet *pkt )
{
    struct Order_Packet *newPkt;
    // struct Order_Packet *c_Order_Packet();
    // int MsgIdTag();
    newPkt = c_Order_Packet( "MakeOrder" );

    if ( newPkt != NULL ) {
        newPkt->Size      = pkt->Size      ;
        newPkt->Entities   = pkt->Entities  ;
        newPkt->TimeToSend = pkt->TimeToSend;
        newPkt->TimeToAct  = pkt->TimeToAct ;
        newPkt->ToActInHrs = pkt->ToActInHrs;
        newPkt->TravelStart = pkt->TravelStart;
        newPkt->State      = pkt->State      ;
        newPkt->Semaphore  = pkt->Semaphore  ;
        newPkt->PcktType   = pkt->PcktType   ;
        newPkt->Force      = pkt->Force      ;
        newPkt->Order      = pkt->Order      ;
        newPkt->Activity   = pkt->Activity   ;
        newPkt->To         = pkt->To         ;
        newPkt->From       = pkt->From       ;
        newPkt->Priority    = pkt->Priority    ;
        newPkt->DestLocale = pkt->DestLocale ;
        newPkt->DestSP     = pkt->DestSP     ;
        newPkt->DestCS     = pkt->DestCS     ;
        newPkt->DestCpi    = pkt->DestCpi    ;
        newPkt->DestDVS    = pkt->DestDVS    ;
        newPkt->OrigLocale = pkt->OrigLocale ;
        newPkt->OrigSP     = pkt->OrigSP     ;
        newPkt->OrigCS     = pkt->OrigCS     ;
        newPkt->OrigCpi    = pkt->OrigCpi    ;
        newPkt->OrigDVS    = pkt->OrigDVS    ;
        newPkt->MsgId      = MsgIdTag( NULL );
        newPkt->PvTruthp   = pkt->PvTruthp   ;
        newPkt->nqep       = pkt->nqep       ;
    }
    return ( newPkt );
}
/* ----- ReadOrdr.c ----- SetMsgOrig f*/
/* ----- DocHeading */
```

```

extern int SetMsgOrig( struct Order_Packet *pkt,
                      int          rLocale, /* SetMsgOrig */
                      int          rSP,    /* SetMsgOrig */
                      int          rCS,    /* SetMsgOrig */
                      int          rCpi,   /* SetMsgOrig */
                      int          rDVS )  /* SetMsgOrig */
{
    pkt->OrigLocale = rLocale ;
    pkt->OrigSP     = rSP     ;
    pkt->OrigCS     = rCS     ;
    pkt->OrigCpi    = rCpi    ;
    pkt->OrigDVS    = rDVS    ;
    return (1);
}
/* ----- ReadOrdr.c ----- SetMsgDest f*/
/* ----- DocMethod */

extern int SetMsgDest( struct Order_Packet *pkt,
                      int          dLocale, /* SetMsgDest */
                      int          dSP,    /* SetMsgDest */
                      int          dCS,    /* SetMsgDest */
                      int          dCpi,   /* SetMsgDest */
                      int          dDVS )  /* SetMsgDest */
{
    pkt->DestLocale = dLocale ;
    pkt->DestSP     = dSP     ;
    pkt->DestCS     = dCS     ;
    pkt->DestCpi    = dCpi    ;
    pkt->DestDVS    = dDVS    ;
    return (1);
}
/* ----- ReadOrdr.c ----- MsgIdTag f*/
/* ----- DocMethod */

extern int MsgIdTag( struct Order_Packet *pkt )
{
    UniqueMsgId += 1 ;

    if (UniqueMsgId > 32700 ) { UniqueMsgId = 0 ; }
    if ( pkt != NULL ) {
        pkt->MsgId = UniqueMsgId ;
    }
    return ( UniqueMsgId );
}
/* ----- ReadOrdr.c ----- Print_LotI f*/
/* ----- DocMethod */

extern void Print_LotI( FILE *out,
                      struct Order_Packet *OPpkt, /* Print_LotI f*/
                      char *Emark )              /* Print_LotI f*/
{
    struct Order_Packet *lpkt;
    //double days, hours, minutes, seconds;
    //int ih, im ;

    lpkt = OPpkt ;

    fprintf( out, "%g,%g,%ld,", lpkt->TimeToSend,lpkt->TimeToAct, lpkt->Order);
    if ( lpkt->Force == Friends ) { fprintf(out,"b," ); }
    else { fprintf(out,"f," ); }
    fprintf( out, "%ld,%g,%g,,,,%s", lpkt->To, (lpkt->TimeToSend/3600.0),
            lpkt->ToActInHrs, Emark );
}
/* ----- ReadOrdr.c ----- Print_Route f*/
/* ----- DocMethod */

```

```
extern void Print_Route( FILE *out,
                        struct Order_Packet *OPpkt,      /* Print_Route f*/
                        char *Emark )                   /* Print_Route f*/
{
    struct Order_Packet *lpkt;
    lpkt = OPpkt ;
    if ( lpkt->PcktType == CmdOrder ) {
        fprintf( out, "Pckt:Cmd%ld,", lpkt->PcktType );
        if ( lpkt->Force == Friends ) { fprintf(out,"b," ); }
        else { fprintf(out,"f," ); }
    }
    else if ( lpkt->PcktType == DataPacket ) {
        fprintf( out, "Pckt:Dat,%2d,", lpkt->Force ); }
    else if ( lpkt->PcktType == CommPacket ) {
        fprintf( out, "Pckt:COM,%2d,", lpkt->Force ); }
    fprintf( out, "MsgID: %5d Orig:%2d.%2d.%2d.%2d.%2d Dest:%2d.%2d.%2d.%2d.%2d %s",
        lpkt->MsgId ,
        lpkt->OrigLocale ,
        lpkt->OrigSP ,      lpkt->OrigCS ,      lpkt->OrigCpi ,
        lpkt->OrigDVS ,
        lpkt->DestLocale ,
        lpkt->DestSP ,      lpkt->DestCS ,      lpkt->DestCpi ,
        lpkt->DestDVS , Emark );
}
/* ----- ReadOrdr.c ----- Print_Order f*/
/*----- DocHeading */
```


30 June 1999

```

extern void Print_Order( FILE *out,
                        struct Order_Packet *OPpkt,      /* Print_Order f*/
                        char *Emark )                   /* Print_Order f*/
{
    struct Order_Packet *lpkt;
    //double temp_time, hours, minutes, seconds ;
    //int   ih, im ;

    lpkt = OPpkt ;
    //temp_time = lpkt->TimeToSend * 24.0 ;
    //ih = (int)temp_time;
    //hours = (double)(int)(temp_time) ;
    //minutes = (double)(int)((temp_time - hours) * 60.0) ;
    //im = (int)minutes;
    //seconds = (((temp_time - hours) * 60) - minutes) * 60.0;
    if ( lpkt->PcktType == CmdOrder ) {
        fprintf( out, "Pckt:Cmd%1d,", lpkt->PcktType );
        if ( lpkt->Force == Friends ) { fprintf(out,"b," ); }
        else { fprintf(out,"f," ); }
    }
    else if ( lpkt->PcktType == DataPacket ) {
        fprintf( out, "Pckt:Dat,%2d,", lpkt->Force ); }
    else if ( lpkt->PcktType == CommPacket ) {
        fprintf( out, "Pckt:COM,%2d,", lpkt->Force ); }

    fprintf( out,
"%2d,%1d,%1d, To %4d,%4d, Cpu %2d, %2d, %6.1f,%6.1f,%12.6f,%12.6f,%9.6f,%s",
        lpkt->Order, lpkt->Force,
        lpkt->Activity,
        lpkt->To, lpkt->From,
        lpkt->DestSP,
        lpkt->Priority,
        lpkt->Size,
        lpkt->Entities,
        lpkt->TimeToSend,
        lpkt->TimeToAct,
        lpkt->ToActInHrs,
        Emark );

}
/* ----- ReadOrdr.c ----- Print_OrderQue f*/
/*----- DocHeading */

```

30 June 1999

```

extern int Print_OrderQue( FILE *out,
                          struct Order_Packet *OPpkt,      /* Print_OrderQue f*/
                          char *Emark )                    /* Print_OrderQue f*/
{
    struct Order_Packet *lpkt;
    // double temp_time, days, hours, minutes, seconds;
    int countQ ;
    countQ = 0 ;
    lpkt = OPpkt ;
    while ( lpkt != NULL ) {
        countQ += 1 ;
        fprintf( out, "          %4d", countQ );
        if ( lpkt->PcktType == CmdOrder ) {
            fprintf( out, " Pckt:Cmd%1d,", lpkt->PcktType );
            if ( lpkt->Force == Friends ) { fprintf( out, "b," ); }
            else { fprintf( out, "f," ); }
        }
        else if ( lpkt->PcktType == DataPacket ) {

            fprintf( out, "          Pckt:Dat,%2d,", lpkt->Force );
        }
        else if ( lpkt->PcktType == CommPacket ) {

            fprintf( out, "          Pckt:COM,%2d,", lpkt->Force );
        }

        fprintf( out,
            "%2d,%1d,%1d, To %2d,%2d, Cpu %2d, %1d, %g,%g,%8.6f,%8.6f,%8.6f,%s",
            lpkt->Order,
            lpkt->Force,
            lpkt->Activity,
            lpkt->To,
            lpkt->From,
            lpkt->DestSP,
            lpkt->Priority,
            lpkt->Size,
            lpkt->Entities,
            lpkt->TimeToSend,
            lpkt->TimeToAct,
            lpkt->ToActInHrs, Emark );

        lpkt = lpkt->nqep ;
    }
    return( countQ );
}
/* ----- ReadOrdr.c ----- Free_OrderQue f*/
/* ----- DocHeading */

```

30 June 1999

```

extern int Free_OrderQue( FILE *out,
                        struct Order_Packet **OPpkt, /* Free_OrderQue f*/
                        char *Emark ) /* Free_OrderQue f*/
{
    struct Order_Packet *lpkt, *opkt;
    //double temp_time, days, hours, minutes, seconds;
    int countQ ;
    countQ = 0 ;
    lpkt = *OPpkt ;
    while ( lpkt != NULL ) {
        countQ += 1 ;

        opkt = lpkt->nqep ;
        free( lpkt ); /* d_Order_Packet(lpkt); */
        lpkt = opkt ;
    }
    *OPpkt = NULL ;
    return( countQ );
}

/* ----- ReadOrdr.c ----- Print_Order f*/
/*----- DocHeading */

```

30 June 1999

```
extern void Print_NewOrder( FILE *out,
                           struct Order_Packet *OPpkt, /* Print_Order f*/
                           char *Emark ) /* Print_Order f*/
{
    struct Order_Packet *lpkt;

    lpkt = OPpkt ;

    fprintf( out, "%4.2f,%4.2f,%1d,", lpkt->TimeToSend,
             lpkt->TimeToAct, lpkt->Order );

    if ( lpkt->Force == Friends) { fprintf(out,"b," ); }
    else { fprintf(out,"f," ); }

    fprintf( out,
             "%1d,%g,%g,0.0,0.0,,, %s",
             lpkt->To,
             (lpkt->TimeToSend/3600.0),
             (lpkt->TimeToAct/3600.0),
             Emark );
}

/*----- DocHeading */
```

```
/* file: Regions.c */
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <math.h>
#include "soa_defs.h"
#include "soa_cnst.h"
#include "proto.h"
#include "rti_services.h"

#define      CompanyEcheleon      5
int instance_nbr = 0 ;
double      RegisterStart        = 0.1 ;
double      RegisterIncrement    = 0.001;
/*-----
```

DocHeading

All functions for Regions

```

extern void CreateRegions( FILE *out,
                          int TotGnEquip,
                          int TotOtEquip,
                          struct Region_Definition *Regions,
                          struct Region_Node_Handle *Region_Node_Handles )

extern void AddCommandRegions( FILE *out,
                              char Which, // 'G' or 'O'
                              struct Region_List *RegionList,
                              struct Filter_Unit_List *FilterList )

extern void AddNewRegion( FILE *out,
                          char Which,
                          struct Region_List *RegionList,
                          struct Filter_Unit_List *FilterList )

extern void AddRegionReference( struct Unit_Region_List **RegOfUnit,
                               struct Region_List *lxtReg)

extern void AddToRegion( FILE *out,
                        char Which,
                        struct Region_List *RegList,
                        struct Unit_Characteristics *UnitChar )

extern int AddToRegionElements( FILE *out,
                               struct Region_List *RegList,
                               struct Region_Element_List *RegEleList,
                               struct Unit_Characteristics *UnitChar )

extern struct Filter_Unit_List *CmdUnitNotInRegion(
                               struct Region_Element_List *RegEleList )

extern struct Unit_Characteristics *FillRegion( FILE *out,
                                                int ApproxEquip,
                                                int ActualEquip,
                                                struct Region_List *CurrentRegion,
                                                struct Unit_Characteristics *UnCharOrigin,
                                                struct Region_Element_List **ElementOfRegion )

extern struct Filter_Unit_List *PutAllInRegBySideInFilterList(
    int Side,
    struct Region_Element_List *RegEleList )

extern struct Filter_Unit_List *PutCoInRegionInFilterList(
    struct Region_Element_List *RegEleList )

extern struct Filter_Unit_List *PutNumInRegBySideInFilterList(
    int Number,
    int Side,
    struct Region_Element_List *RegEleList )

extern struct Filter_Unit_List *PutRegionInFilterList(
    struct Region_Element_List *RegEleList )

extern int RemoveRegionReference(
    struct Unit_Characteristics *UnitChar,
    struct Region_List *lxtReg)

extern void PrintRegionElements( FILE *out,
                                struct Region_Element_List *RegEle,
                                char *Emark )

extern void PrintRegionsNodes( FILE *out,
                               struct Region_List *RegList,
                               char *Emark )

```

30 June 1999

```
extern void PrintRegions( FILE *out,  
    struct Region_List *RegList,  
    char *Emark )
```

```
extern void PrintOneRegion( FILE *out,  
    struct Region_List *RegList,  
    char *Emark )
```

```
*/
```

```
/*----- GrowHier.c ----- CreateRegions f*/  
/*----- DocHeading */
```

```
extern void CreateRegions( FILE *out,
                          int TotGnEquip,
                          int TotOtEquip,
                          struct Region_Definition *Regions,
                          struct Region_Node_Handle *Region_Node_Handles )
/*-----EndDocHead-----*/
{
    int i,j,k, GnEquipPerRegion, OtEquipPerRegion;
    struct Region_List *lxtReg ;
    char Name[Serv_Nam_M] ;

    Region_Node_Handles->xtFed = NULL ;

    GnEquipPerRegion = ((int)((double)TotGnEquip * Regions->Commit)) / Regions->high ;
    OtEquipPerRegion = ((int)((double)TotOtEquip * Regions->Commit)) / Regions->high ;

    for(i=0; i< Regions->high; i++ ) {

        if ( Region_Node_Handles->xtReg == NULL || i == 0 ) {
            if (Region_Node_Handles->xtReg == NULL ) {
                Region_Node_Handles->xtReg = c_Region_List(" CreateRegions ShrI");
            }
            lxtReg = Region_Node_Handles->xtReg;
        }
        else {
            lxtReg->xtReg = c_Region_List(" CreateRegions Shr");
            lxtReg = lxtReg->xtReg ;
        }

        lxtReg->EquipGn = GnEquipPerRegion ;
        lxtReg->EquipOt = OtEquipPerRegion ;
        lxtReg->Id = i + 1;

        sprintf( Name,"RegXHigh%3.3d", lxtReg->Id );
        strcpy(lxtReg->Name, Name);
        //printf("Create %3d %8.8x \n", lxtReg->Id, lxtReg);

        lxtReg->Category = 1 ; /* high */
        printf("Create %3d %8.8x \n", lxtReg->Id, lxtReg);
        lxtReg->SubInteract[OrderSOM] = 1 ;
        lxtReg->SubInteract[ReportSOM] = 1 ;
        lxtReg->SubInteract[FireSOM] = 1 ;
        lxtReg->SubInteract[SenseSOM] = 1 ;
        lxtReg->SubInteract[SupplySOM] = 1 ;

        lxtReg->PubInteract[OrderSOM] = 1 ;
        lxtReg->PubInteract[ReportSOM] = 1 ;
        lxtReg->PubInteract[FireSOM] = 1 ;
        lxtReg->PubInteract[SenseSOM] = 1 ;
        lxtReg->PubInteract[SupplySOM] = 0 ;

        lxtReg->SubObjects[RouteSOM] = 1 ;
        lxtReg->SubObjects[MissionSOM] = 1 ;
        lxtReg->SubObjects[UnitTypeSOM] = 1 ;
        lxtReg->SubObjects[PlanSOM] = 1 ;
        lxtReg->SubObjects[HealthSOM] = 1 ;

        lxtReg->PubObjects[RouteSOM] = 1 ;
        lxtReg->PubObjects[MissionSOM] = 1 ;
        lxtReg->PubObjects[UnitTypeSOM] = 1 ;
        lxtReg->PubObjects[PlanSOM] = 1 ;
        lxtReg->PubObjects[HealthSOM] = 1 ;

    }

    GnEquipPerRegion = ((int)((double)TotGnEquip * (1.0-Regions->Commit))) / Regions->other ;
}
```


30 June 1999

```
OtEquipPerRegion = ((int)((double)TotOtEquip * (1.0-Regions->Commit))) / Regions->other ;
```

```
/*
// For the non-warfighter units:
//     only put them in separate regions since support is for your own force.
//
//     Later the support regions can be extended to include the warfighter regions.
//
//     This current allocation arrangement approximates a geographical distribution.
//
//     Density in Geo regions should pertain to a f(algorithms & CPU)
*/
k = i+1;

for(j=i; j< Regions->other+i; j++ ) {

    lxtReg->xtReg = c_Region_List(" CreateRegions Gn");
    lxtReg = lxtReg->xtReg ;
    if ( j+1 == Regions->other+i ) { /* this is the last Geo region set large */
        lxtReg->EquipGn    = 10*GnEquipPerRegion ;
        lxtReg->EquipOt    = 0 ;    }
    else {
        lxtReg->EquipGn    = GnEquipPerRegion ;
        lxtReg->EquipOt    = 0 ;
    }
    lxtReg->Id            = k;

    sprintf( Name,"RegGnLow%3.3d", lxtReg->Id );
    strcpy(lxtReg->Name, Name);
    k += 1 ;
    lxtReg->Category      = 2 ;    /* not high */

    printf("Create %3d %8.8x %8.8x \n", lxtReg->Id, lxtReg, lxtReg->xtReg);

    lxtReg->xtReg = c_Region_List(" CreateRegions Ot");
    lxtReg = lxtReg->xtReg ;
    if ( j+1 == Regions->other+i ) { /* this is the last Geo region set large */
        lxtReg->EquipGn    = 0 ;
        lxtReg->EquipOt    = 10*OtEquipPerRegion ; }
    else {
        lxtReg->EquipGn    = 0 ;
        lxtReg->EquipOt    = OtEquipPerRegion ;
    }
    lxtReg->Id            = k;
    sprintf( Name,"RegOtLow%3.3d", lxtReg->Id );
    strcpy(lxtReg->Name, Name);
    k += 1 ;
    lxtReg->Category      = 2 ;    /* not high */
    printf("Create %3d %8.8x %8.8x \n", lxtReg->Id, lxtReg, lxtReg->xtReg);
    lxtReg->SubInteract [OrderSOM]      = 1 ;
    lxtReg->SubInteract [ReportSOM]     = 0 ;
    lxtReg->SubInteract [FireSOM]       = 0 ;
    lxtReg->SubInteract [SenseSOM]      = 0 ;
    lxtReg->SubInteract [SupplySOM]     = 1 ;

    lxtReg->PubInteract [OrderSOM]      = 0 ;
    lxtReg->PubInteract [ReportSOM]     = 1 ;
    lxtReg->PubInteract [FireSOM]       = 0 ;
    lxtReg->PubInteract [SenseSOM]      = 0 ;
    lxtReg->PubInteract [SupplySOM]     = 1 ;

    lxtReg->SubObjects [RouteSOM]       = 1 ;
    lxtReg->SubObjects [MissionSOM]     = 1 ;
    lxtReg->SubObjects [UnitTypeSOM]    = 0 ;
    lxtReg->SubObjects [PlanSOM]        = 0 ;
    lxtReg->SubObjects [HealthSOM]      = 1 ;
```

```
lxtReg->PubObjects[RouteSOM]      = 1 ;
lxtReg->PubObjects[MissionSOM]     = 0 ;
lxtReg->PubObjects[UnitTypeSOM]    = 0 ;
lxtReg->PubObjects[PlanSOM]        = 0 ;
lxtReg->PubObjects[HealthSOM]      = 1 ;
}

}

/*----- Regions.c ----- AddCommandRegions( f*/
/*----- DocHeading */
```

30 June 1999

```

extern void AddCommandRegions( FILE *out,
                               char      Which, /* 'G' or 'O' */
                               struct Region_List *RegionList,
                               struct Filter_Unit_List *FilterList )
/*-----EndDocHead---*/
{
    struct Region_List      *lRegList ;
    struct Region_Element_List *pxtEle;
    struct Filter_Unit_List *pFilUnit ;
    struct Unit_Characteristics *UnChr ;
    struct Unit_List      *ULstp;
    char  Name[Serv_Nam_M];
    int Id, Equipment;
    Equipment = 0 ;

    if ( FilterList != NULL ) {
        if ( RegionList == NULL ) { printf("RegionList NULL AddCommandRegion\n"); exit(-86); }
        else{
            lRegList = RegionList ;

            while( lRegList->xtReg != NULL ) { lRegList = lRegList->xtReg ; }

            lRegList->xtReg = c_Region_List("AddCommandRegions");
            Id = lRegList->Id + 1 ;
            lRegList = lRegList->xtReg ;
            lRegList->Id = Id ;
            sprintf( Name,"Reg%c_Cmd%3.3d", Which, lRegList->Id );
            strcpy(lRegList->Name, Name );
            pFilUnit = FilterList ;
            /* Note EVERY command level could be a unique region! */
            do {

                UnChr = pFilUnit->UChrp ; /* put unit an immediate subordinates in a new region*/

                if ( UnChr->ULstp->Subrdp != NULL &&
                    UnChr->ULstp->Subrdp->UChrp != NULL ) {

                    if ( Which == 'G' && lRegList->xtGnRegEle == NULL ) {
                        lRegList->xtGnRegEle = c_Region_Element_List( "AddCommandRegions");
                        pxtEle = lRegList->xtGnRegEle ;
                    }
                    else if ( Which == 'O' && lRegList->xtOtRegEle == NULL ) {
                        lRegList->xtOtRegEle = c_Region_Element_List( "AddCommandRegions");
                        pxtEle = lRegList->xtOtRegEle ;
                    }
                    else {

                        if ( pxtEle == NULL ) { printf("Error:AddCommandRegions\n"); exit(-86); }
                        pxtEle->xtEle = c_Region_Element_List( "AddCommandRegions");
                        pxtEle = pxtEle->xtEle ;
                    }

                    pxtEle->Id      = Id ;
                    pxtEle->UChrp = UnChr ;
                    Equipment += UnChr->Equipment;
                    AddRegionReference( &UnChr->RegOfUnit, lRegList); /*commander */
                    ULstp = UnChr->ULstp->Subrdp ;
                    /* Region_Element_List extention of subordinates */
                    do {
                        UnChr = ULstp->UChrp ;

                        pxtEle->xtEle = c_Region_Element_List( "AddCommandRegions");
                        pxtEle = pxtEle->xtEle ;
                        pxtEle->Id      = Id ;
                        pxtEle->UChrp = UnChr ;
                        Equipment += UnChr->Equipment;
                    }
                }
            } while ( pFilUnit = pFilUnit->next );
        }
    }
}

```

30 June 1999

```

        AddRegionReference( &UnChr->RegOfUnit, lRegList); /* subordinates */
        ULstp = ULstp->nrep;

        } while( ULstp != NULL);
    }
        pFilUnit = pFilUnit->nxtFilteredUnitp ;
    } while ( pFilUnit != NULL);

    if ( Which == 'G' ){ lRegList->EquipGn = Equipment ; }
    else {               lRegList->EquipOt = Equipment ; }

}
} /* end of if ( FilterList != NULL ) { */

}

/*----- Regions.c ----- AddCommandRegions( f*/
/*----- DocHeading */

```

```

extern void AddNewRegion( FILE *out,
                        char Which, /* 'G' or 'O' */
                        struct Region_List *RegionList,
                        struct Filter_Unit_List *FilterList )
/*-----EndDocHead-----*/
{
    struct Region_List *lRegList ;
    struct Region_Element_List *pxtEle;
    struct Filter_Unit_List *pFilUnit ;
    struct Unit_Characteristics *UnChr ;
    struct Unit_List *ULstp;
    char Name[Serv_Nam_M];
    int Id, Equipment;
    Equipment = 0 ;

    if ( FilterList != NULL ) {
        if ( RegionList == NULL ) { printf("RegionList NULL AddCommandRegion\n"); exit(-86); }
        else{
            lRegList = RegionList ;

            while( lRegList->xtReg != NULL ) { lRegList = lRegList->xtReg ; }

            lRegList->xtReg = c_Region_List("AddNewRegion");
            Id = lRegList->Id + 1 ;
            lRegList = lRegList->xtReg ;
            lRegList->Id = Id ;
            lRegList->SubInteract[OrderSOM] = 1 ;
            lRegList->SubInteract[ReportSOM] = 0 ;
            lRegList->SubInteract[FireSOM] = 0 ;
            lRegList->SubInteract[SenseSOM] = 0 ;
            lRegList->SubInteract[SupplySOM] = 1 ;

            lRegList->PubInteract[OrderSOM] = 0 ;
            lRegList->PubInteract[ReportSOM] = 1 ;
            lRegList->PubInteract[FireSOM] = 0 ;
            lRegList->PubInteract[SenseSOM] = 0 ;
            lRegList->PubInteract[SupplySOM] = 1 ;

            lRegList->SubObjects[RouteSOM] = 1 ;
            lRegList->SubObjects[MissionSOM] = 1 ;
            lRegList->SubObjects[UnitTypeSOM] = 0 ;
            lRegList->SubObjects[PlanSOM] = 0 ;
            lRegList->SubObjects[HealthSOM] = 1 ;

            lRegList->PubObjects[RouteSOM] = 1 ;
            lRegList->PubObjects[MissionSOM] = 0 ;
            lRegList->PubObjects[UnitTypeSOM] = 0 ;
            lRegList->PubObjects[PlanSOM] = 0 ;
            lRegList->PubObjects[HealthSOM] = 1 ;

            sprintf( Name,"Reg%c_Adl%3.3d", Which, lRegList->Id );
            strcpy(lRegList->Name, Name );
            pFilUnit = FilterList ;
            /* Note EVERY command level could be a unique region! */
            do {

                UnChr = pFilUnit->UChrp ; /* put unit in the new region*/

                if ( UnChr != NULL ) {

                    if ( Which == 'G' && lRegList->xtGnRegEle == NULL ) {
                        lRegList->xtGnRegEle = c_Region_Element_List( "AddNewRegion");
                        pxtEle = lRegList->xtGnRegEle ;
                    }
                    else if ( Which == 'O' && lRegList->xtOtRegEle == NULL ) {
                        lRegList->xtOtRegEle = c_Region_Element_List( "AddNewRegion");
                    }
                }
            } while ( pFilUnit = pFilUnit->xtFilUnit );
        }
    }
}

```

30 June 1999

```

        pxtEle = lRegList->xtOtRegEle ;
    }
    else {

        if ( pxtEle == NULL ) { printf("Error:AddNewRegion\n" ); exit(-86); }
        pxtEle->xtEle = c_Region_Element_List( "AddNewRegion");
        pxtEle = pxtEle->xtEle ;
    }

    pxtEle->Id      = Id ;
    pxtEle->UChrp  = UnChr ;
    Equipment += UnChr->Equipment;

    AddRegionReference( &UnChr->RegOfUnit, lRegList); /*commander */
}
    pFilUnit = pFilUnit->nxtFilteredUnitp ;
} while ( pFilUnit != NULL);

if ( Which == 'G' ){ lRegList->EquipGn = Equipment ; }
else {              lRegList->EquipOt = Equipment ; }

}
} /* end of if ( FilterList != NULL ) { */

}

/*----- Grow.c ----- AddRegionReference f*/
/*----- DocHeading */

```

30 June 1999

```

extern void AddRegionReference( struct Unit_Region_List **RegOfUnit,
                               struct Region_List      *lxtReg)
/*-----EndDocHead---*/
{
    struct Unit_Region_List      *curRegOfUnit ;

    if ( *RegOfUnit == NULL ) { /* point from Unit Char to the region */
        *RegOfUnit = c_Unit_Region_List( " FillRegions");
        curRegOfUnit = *RegOfUnit;
        curRegOfUnit->xtReg = lxtReg ;
        // printf("added region ptr to UnChar %3d \n", curRegOfUnit->xtReg->Id );
    }
    else { /* go to end and add another region link to this Unit */
        curRegOfUnit = *RegOfUnit;
        while( curRegOfUnit->nxtRegOfUnit != NULL &&
               curRegOfUnit->xtReg->Id != lxtReg->Id ) {
            curRegOfUnit = curRegOfUnit->nxtRegOfUnit ; }

        if ( curRegOfUnit->xtReg->Id != lxtReg->Id ) {

            curRegOfUnit->nxtRegOfUnit = c_Unit_Region_List( " AddRegionR");
            curRegOfUnit->nxtRegOfUnit->xtReg = lxtReg ;

        }
        else {

            printf(
                "Warning Attempt to add duplicate region to a Unit_Region_List(AddRegionReference)\n");

        }
    }
}

/*----- GrowHier.c ----- AddToRegions( f*/
/*----- DocHeading */

```

```

extern void AddToRegion( FILE      *out,
                        char       Which,
                        struct Region_List *RegList,
                        struct Unit_Characteristics *UnitChar )
/*-----EndDocHead-----*/
{
    struct Unit_Characteristics *UnChr ;
    struct Region_Element_List *pxtEle;

    UnChr = UnitChar;

    if ( Which == 'G' ) {          /* could be first entry */
        if ( RegList->xtGnRegEle == NULL ) {

            RegList->xtGnRegEle = c_Region_Element_List( "AddToRegion");
            pxtEle = RegList->xtGnRegEle ;
        }
        else {
            pxtEle = RegList->xtGnRegEle;
        }
    }
    else if ( Which == 'O' ) {          /* could be first entry */
        if ( RegList->xtOtRegEle == NULL ) {

            RegList->xtOtRegEle = c_Region_Element_List( "AddToRegion");
            pxtEle = RegList->xtOtRegEle ;
        }
        else {
            pxtEle = RegList->xtOtRegEle;
        }
    }
    while ( pxtEle->xtEle != NULL && pxtEle->UChrp != UnitChar) {
        pxtEle = pxtEle->xtEle ;
    }
    if ( pxtEle->UChrp != UnitChar ) {

        pxtEle->xtEle = c_Region_Element_List( "AddToRegion");
        pxtEle = pxtEle->xtEle ;
        pxtEle->Id      = RegList->Id ;
        pxtEle->UChrp   = UnChr ;
        RegList->EquipGn += UnChr->Equipment;

        AddRegionReference( &UnChr->RegOfUnit, RegList);
    }
}
/*----- GrowHier.c ----- AddToRegionElements( f*/
/*----- DocHeading */

```


30 June 1999

```

extern int AddToRegionElements( FILE *out,
                                struct Region_List *RegList,
                                struct Region_Element_List *RegEleList,
                                struct Unit_Characteristics *UnitChar )
/*-----EndDocHead-----*/
{
    struct Unit_Characteristics *UnChr ;
    struct Region_Element_List *pxtEle;

    UnChr = UnitChar;

    if (      RegEleList != NULL ) {
        pxtEle = RegEleList ;
    }

    /* Go to the end of the list */
    while ( pxtEle->xtEle != NULL ) { pxtEle = pxtEle->xtEle ; }

    if ( pxtEle->UChrp != UnitChar ) {

        pxtEle->xtEle = c_Region_Element_List( "AddToRegion");
        pxtEle = pxtEle->xtEle ;

        pxtEle->Id      = RegList->Id ;
        pxtEle->UChrp   = UnChr ;

        //      RegList->EquipGn += UnChr->Equipment;

        AddRegionReference( &UnChr->RegOfUnit, RegList);
    }
    return( UnChr->Equipment );
}

/*----- Grow.c ----- *CmdUnitNotInRegion f*/
/*----- DocHeading */

```

30 June 1999

```

extern struct Filter_Unit_List *CmdUnitNotInRegion(
    struct Region_Element_List *RegEleList )
/*-----EndDocHead-----*/
{
    struct Region_Element_List    *pRegEle ;
    struct Filter_Unit_List      *Top, *cur;

    Top = NULL;
    pRegEle = RegEleList;

    if ( pRegEle != NULL ) {

        do {                                /* go through region element list */

            if ( pRegEle->UChrp != NULL ) {

                if ( pRegEle->UChrp->ULstp != NULL &&
                    pRegEle->UChrp->ULstp->UCmdp != NULL &&
                    pRegEle->UChrp->ULstp->UCmdp->UChrp != NULL ) {

                    if ( pRegEle->UChrp->ULstp->UCmdp->UChrp->RegOfUnit == NULL ) {

                        if ( Top == NULL ) {

                            Top = c_Filter_Unit_List( "CmdUnitNotInRegion");
                            cur = Top;

                        }

                        else {

                            cur->nxtFilteredUnitp = c_Filter_Unit_List( "CmdUnitNotInRegion");
                            cur = cur->nxtFilteredUnitp;

                        }

                        cur->UChrp = pRegEle->UChrp->ULstp->UCmdp->UChrp ;

                    }

                }

            }

            pRegEle = pRegEle->xtEle;

        } while ( pRegEle != NULL );      /* end of list */

    }

    /* if ( pRegEle != NULL ) { */
    return(Top);
    /*      *CmdUnitNotInRegion */

}

/*----- GrowHier.c ----- FillRegions f*/
/*----- DocHeading */

```

30 June 1999

```

extern struct Unit_Characteristics *FillRegion( FILE *out,
        int                ApproxEquip,
        int                *ActualEquip,
        struct Region_List *CurrentRegion,
        struct Unit_Characteristics *UnCharOrigin,
        struct Region_Element_List **ElementOfRegion )
/*-----EndDocHead-----*/
{
    /* This filling addresses a Geo distribution - non overlapping */

    struct Filter_Unit_List *FilterList, *tempF ;
    int GoalEquip, EquipSoFar;
    int MaxEch, AddEquip ;
    int CountSubrEquip();
    struct Region_Element_List *RegElements;
    struct Unit_Characteristics *lUnChr;
    struct Region_Element_List *lxtEle;

    MaxEch = MaxEcheleon(NULL, UnCharOrigin, " FillRegions" );
    lUnChr = UnCharOrigin;

    // lxtEle = c_Region_Element_List(" FillRegions ");

    GoalEquip = ApproxEquip;
    EquipSoFar = 0 ;
    do {
        if ( lUnChr->Echeleon == MaxEch - 1 || lUnChr->Echeleon == MaxEch ) { /* candidate */
            if ( *ElementOfRegion == NULL ) {
                *ElementOfRegion = c_Region_Element_List(" FillRegions ");
                lxtEle = *ElementOfRegion;
                // printf(" Start Elements of FillRegion %8.8x \n", lxtEle );
            }
            else {
                //printf("      FillRegions Unit Name %21s %8.8x\n", lUnChr->Name, lxtEle);
                lxtEle->xtEle = c_Region_Element_List(" FillRegions ");
                lxtEle = lxtEle->xtEle ;
            }
            if ( CurrentRegion->Category == 1 ) { /* high */
                lUnChr->Designation = WarFighter ;
            }
            lxtEle->UChrp = lUnChr;
            //printf("      FillRegions Unit Name %21s %-21s\n", lUnChr->Name, lxtEle->UChrp-
>Name );
            lxtEle->Id = CurrentRegion->Id;
            AddRegionReference( &lUnChr->RegOfUnit, CurrentRegion ) ; /* had to add lxtReg
possible problem*/

            EquipSoFar += lUnChr->Equipment ; /* Number in the region */

            //fprintf(out," %3d unit %18s level %1d Equip %4d Goal %4d \n",
            //CurrentRegion->Id,lUnChr->Name, lUnChr->Echeleon, EquipSoFar, GoalEquip );

        }

        lUnChr = lUnChr->ngep;
    } while( lUnChr != NULL && ( GoalEquip > (EquipSoFar + lUnChr->Equipment/2)) );

    // && pUnChr != lUnChr
    // Since the above fills by second to lowest echeleon in the
    // grown scenario the commanders or these units should be in same
    // same region. (Generally commander echeleon is BN distribution by Co)
    // But this is being done by relationship to maintain flexibility

    AddEquip = 0;
    RegElements = *ElementOfRegion;
    FilterList = CmdUnitNotInRegion( RegElements );
    /* add to region */

```

30 June 1999

```

tempF = FilterList ;
do{
    AddEquip += AddToRegionElements( out, CurrentRegion,
                                     RegElements, tempF->UChrp );
    tempF = tempF->nxtFilteredUnitp;
} while( tempF != NULL);
*ActualEquip = EquipSoFar + AddEquip;

fprintf(out, "in Region %2d Equipment placed %5d for Goal %5d %5d  AddEquip %5d \n",
        CurrentRegion->Id, EquipSoFar, ApproxEquip, *ActualEquip, AddEquip );

return( lUnChr );

// fprintf(out, " total  %5d \n", EquipSoFar );

}
/* end of FillRegion ===== */

/*----- Grow.c ----- PrintRegions f*/
/*----- DocHeading */

```

30 June 1999

```

extern struct Region_List *FindRegion( int RegId,
                                     struct Region_List *RegList )
/*-----EndDocHead---*/
{
    struct Region_List *RegionPtr; /* nodes where region resides */

    RegionPtr = NULL ;

    if ( RegList != NULL ) {

        do {                                     /* find Region Id */

            if ( RegList->Id == RegId ) {
                RegionPtr = RegList ;
            }
            RegList = RegList->xtReg;
        } while ( RegList != NULL && RegionPtr == NULL);          /* end of list */

    } /* if ( RegList != NULL ) { */
    return(RegionPtr);
}

/*          Put a Company in a Filter List */
/*-----DocHeading */

```

```

extern struct Filter_Unit_List *PutCoInRegionInFilterList( /* Regions.c */
    struct Region_Element_List *RegEleList )
/*-----EndDocHead---*/
{
    struct Region_Element_List *pRegEle ;
    struct Filter_Unit_List *Top, *cur;

    Top = NULL;
    pRegEle = RegEleList;

    if ( pRegEle != NULL ) {

        do {
            /* go through region element list */
            if ( pRegEle->UChrp != NULL ) {

                if ( pRegEle->UChrp->Echeleon == CompanyEcheleon ) { /* companys */

                    if ( Top == NULL ) {

                        Top = c_Filter_Unit_List( "PutCoInRegionInFilterList " );
                        cur = Top;

                    }

                    else {

                        cur->nxtFilteredUnitp = c_Filter_Unit_List( " PutCoInRegionInFilterList");
                        cur = cur->nxtFilteredUnitp;

                    }

                    cur->UChrp = pRegEle->UChrp ;

                }

            }

            pRegEle = pRegEle->xtEle;

        } while ( pRegEle != NULL ) ; /* end of list */

    }

    return(Top);
} /* *PutCoInRegionInFilterList( struct Region_Element_List *RegEleList ) */

/* Put All in Region by Side in a Filter List */
/*-----DocHeading */

```

30 June 1999

```

extern struct Filter_Unit_List *PutAllInRegBySideInFilterList( int Side,
    struct Region_Element_List *RegEleList )
/*-----EndDocHead---*/
{
    struct Region_Element_List *pRegEle ;
    struct Filter_Unit_List *Top, *cur;

    Top = NULL;
    pRegEle = RegEleList;

    if ( pRegEle != NULL ) {

        do {
            /* go through region element list */
            if ( pRegEle->UChrp != NULL ) {

                if ( pRegEle->UChrp->Force == Side ) { /* companys */

                    if ( Top == NULL ) {

                        Top = c_Filter_Unit_List( "PutAllInRegBySideInFilterList " );
                        cur = Top;

                    }

                    else {

                        cur->nxtFilteredUnitp = c_Filter_Unit_List("PutAllInRegBySideInFilterList");
                        cur = cur->nxtFilteredUnitp;

                    }
                    cur->UChrp = pRegEle->UChrp ;

                }

            }
            pRegEle = pRegEle->xtEle;

        } while ( pRegEle != NULL ) ; /* end of list */

    }
    return(Top);
} /* *PutAllInRegBySideInFilterList( int Side, struct Region_Element_List *RegEleList ) */

/* Put Number in Region by Side in a Filter List */
/*-----DocHeading */

```

30 June 1999

```

extern struct Filter_Unit_List *PutNumInRegBySideInFilterList( int Number,
    int Side,
    struct Region_Element_List *RegEleList )
/*-----EndDocHead---*/
{
    struct Region_Element_List *pRegEle ;
    struct Filter_Unit_List *Top, *cur;
    int localcount;
    localcount = 0 ;
    Top = NULL;
    pRegEle = RegEleList;

    if ( pRegEle != NULL ) {

        do {
            if ( pRegEle->UChrp != NULL ) {
                /* go through region element list */

                if ( pRegEle->UChrp->Force == Side && Number > localcount ) { /* companys */
                    printf(" pRegEle->UChrp->Name %s %3d %3d \n",
                        pRegEle->UChrp->Name, Number, localcount);
                    if ( Top == NULL ) {
                        Top = c_Filter_Unit_List( "PutAllInRegBySideInFilterList " );
                        cur = Top;
                    }

                    else {
                        cur->nxtFilteredUnitp = c_Filter_Unit_List("PutAllInRegBySideInFilterList");
                        cur = cur->nxtFilteredUnitp;
                    }
                    cur->UChrp = pRegEle->UChrp ;
                    localcount += pRegEle->UChrp->Equipment;
                }
            }
            pRegEle = pRegEle->xtEle;

        } while ( pRegEle != NULL ) ; /* end of list */

    }
    return(Top);
} /* *PutNumInRegBySideInFilterList( int Side, struct Region_Element_List *RegEleList ) */

/*----- DocHeading */

```


30 June 1999

```

extern struct Filter_Unit_List *PutRegionInFilterList(
    struct Region_Element_List *RegEleList )
/*-----EndDocHead---*/
{
    struct Region_Element_List    *pRegEle ;
    struct Filter_Unit_List       *Top, *cur;

    Top = NULL;
    pRegEle = RegEleList;
    if ( pRegEle != NULL ) {

do {                                     /* go through region element list */
    if ( pRegEle->UChrp != NULL ) {
        if ( Top == NULL ) {
            Top = c_Filter_Unit_List( "PutRegionInFilterList " );
            cur = Top;
        }
        else {
            cur->nxtFilteredUnitp = c_Filter_Unit_List( " PutRegionInFilterList");
            cur = cur->nxtFilteredUnitp;
        }
        cur->UChrp = pRegEle->UChrp ;
    }
    pRegEle = pRegEle->xtEle;
} while ( pRegEle != NULL) ;           /* end of list */
}
return(Top);
} /* *PutRegionInFilterList( struct Region_Element_List *RegEleList ) */

/*----- Grow.c ----- RemoveRegionReference( f*/
/*----- DocHeading */

```

30 June 1999

```

extern int RemoveRegionReference(
    struct Unit_Characteristics *UnitChar,
    struct Region_List *lxtReg)
/*-----EndDocHead-----*/
{
    struct Unit_Region_List d_Unit_Region_List(struct Unit_Region_List *s);
    struct Unit_Region_List *curRegOfUnit ;
    struct Unit_Region_List *preRegOfUnit ;
    int Success ;
    struct Region_Element_List *curxtEle;
    struct Region_Element_List *prextEle;
    extern struct Region_Element_List d_Region_Element_List( struct Region_Element_List *s);

    Success = 0 ;

    if ( UnitChar->RegOfUnit != NULL ) { /* point from Unit Char to the region */
        curRegOfUnit = UnitChar->RegOfUnit;

        if ( curRegOfUnit->xtReg->Id == lxtReg->Id ) {

            UnitChar->RegOfUnit = curRegOfUnit->nxtRegOfUnit ;
            d_Unit_Region_List( curRegOfUnit );
            Success = 1 ;
        }
        else {

            preRegOfUnit = curRegOfUnit ;

            while( curRegOfUnit != NULL ) {
                if ( curRegOfUnit->xtReg->Id == lxtReg->Id ) {

                    preRegOfUnit->nxtRegOfUnit = curRegOfUnit->nxtRegOfUnit ;
                    d_Unit_Region_List( curRegOfUnit );
                    Success = 1 ;
                    curRegOfUnit = preRegOfUnit ;
                }
                else {
                    preRegOfUnit = curRegOfUnit ;
                }
                curRegOfUnit = curRegOfUnit->nxtRegOfUnit ;
            }
        }
    }
    if ( lxtReg->xtGnRegEle != NULL ) {
        curxtEle = lxtReg->xtGnRegEle ;

        if ( curxtEle->UChrp == UnitChar ) {

            lxtReg->xtGnRegEle = curxtEle->xtEle;
            d_Region_Element_List(curxtEle);
            Success += 1 ;
        }
        else {
            prextEle = curxtEle ;

            while( curxtEle != NULL ) {
                if ( curxtEle->UChrp == UnitChar ) {
                    prextEle->xtEle = curxtEle->xtEle ;
                    d_Region_Element_List( curxtEle );
                    Success += 1;
                    curxtEle = prextEle ;
                }
                else {
                    prextEle = curxtEle ;
                }
                curxtEle = curxtEle->xtEle ;
            }
        }
    }
}

```

```
    }  
  }  
}  
return(Success);  
}          /* RemoveRegionReference */
```

```
/*----- Grow.c ----- AddRegionReference f*/  
/*----- DocHeading */
```

30 June 1999

```

extern void PrintRegionElements( FILE *out,
                                struct Region_Element_List *RegEle,
                                char *Emark )
/*-----EndDocHead---*/
{
    struct Region_Element_List *pRegEle;

    pRegEle = RegEle ;

    if ( pRegEle != NULL ){
        do {
            fprintf(out, "Region %3d %21s %s",
                    pRegEle->Id, pRegEle->UChrp->Name, Emark );

            pRegEle = pRegEle->xtEle ;
        } while( pRegEle != NULL );
    }
}

/*----- Grow.c ----- PrintRegions f*/
/*----- DocHeading */

```

30 June 1999

```

extern void PrintRegionsNodes( FILE *out,
                             struct Region_List *RegList,
                             char *Emark )
/*-----EndDocHead---*/
{
    struct Nodes_wrt_Region_List *NodeWRTRegion ; /* nodes where region resides */

    if ( RegList != NULL ) {

        do {
            /* fill Geo Regions FILL */

            fprintf(out, "%s %2d Equip Gn %4d Ot %4d Nodes:",
                    RegList->Name, RegList->Id, RegList->EquipGn, RegList->EquipOt );

            if ( RegList->NodeWRTRegion != NULL ) {

                NodeWRTRegion = RegList->NodeWRTRegion;
                do {

                    fprintf(out, " %s", NodeWRTRegion->NodeOfFed->Name );

                    NodeWRTRegion = NodeWRTRegion->xtNodeWRTRegion;

                } while( NodeWRTRegion != NULL );

            }
            fprintf(out, "%s", Emark);

            RegList = RegList->xtReg;
        } while ( RegList != NULL); /* end of list */

    } /* if ( RegList != NULL ) { */

}

/*----- Grow.c ----- PrintRegions f*/
/*----- DocHeading */

```

```

extern void PrintRegions( FILE *out,      /* Regions.c */
                        struct Region_List *RegList,
                        char *Emark )

/*-----EndDocHead-----*/
{
struct Region_Element_List      *pRegEle;
int i ;
i = 0 ;
if ( RegList != NULL ) {
    fprintf(out,"Regionl %2d Equip Gn %4d Ot %4d \n",
            RegList->Id, RegList->EquipGn, RegList->EquipOt );
    do {
        /* fill Geo Regions FILL */

        if ( RegList->xtGnRegEle != NULL ) {

            pRegEle = RegList->xtGnRegEle;

            do {
                if ( pRegEle->UChrp != NULL || i == 1 ) {
                    fprintf(out, "      Region Gn %3d Cat %1d %21s %s",
                            pRegEle->Id,RegList->Category, pRegEle->UChrp->Name, Emark );
                }
                else {
                    fprintf(out, "      Region Gn %3d Cat %1d %21s %s",
                            pRegEle->Id,RegList->Category, pRegEle->UChrp->Name, Emark );
                }

                pRegEle = pRegEle->xtEle ;
            } while( pRegEle != NULL );
        }
        if ( RegList->xtOtRegEle != NULL ) {

            pRegEle = RegList->xtOtRegEle;

            do {
                if ( pRegEle->UChrp != NULL || i == 1 ) {
                    fprintf(out, "      Region Ot %3d Cat %1d %21s %s",
                            pRegEle->Id,RegList->Category, pRegEle->UChrp->Name, Emark );
                }
                else {
                    fprintf(out, "      Region Ot %3d Cat %1d %21s %s",
                            pRegEle->Id,RegList->Category, pRegEle->UChrp->Name, Emark );
                }

                pRegEle = pRegEle->xtEle ;
            } while( pRegEle != NULL );
        }

        RegList = RegList->xtReg;
    } while ( RegList != NULL); /* end of list */
} /* if ( RegList != NULL ) {*/

}

/*----- Grow.c -----          PrintOneRegion f*/
/*-----                      DocHeading */

```

```

extern void PrintOneRegion( FILE *out,
                           struct Region_List *RegList,
                           char *Emark )
/*-----EndDocHead-----*/
{
  struct Region_Element_List *pRegEle;
  int i ;

  i = 0 ;

  /* fill Geo Regions FILL */
  if ( RegList != NULL ) {

    fprintf(out,"Region2 %2d Equip Gn %4d Ot %4d \n",
            RegList->Id, RegList->EquipGn, RegList->EquipOt );

    if ( RegList->xtGnRegEle != NULL ) {

      pRegEle = RegList->xtGnRegEle;

      do {
        if ( pRegEle->UChrp != NULL || i == 1 ) {
          fprintf(out, "      Region Gn %3d Cat %1d %21s %s",
                  pRegEle->Id,RegList->Category, pRegEle->UChrp->Name, Emark );
        }
        else {
          fprintf(out, "      Region Gn %3d Cat %1d %21s %s",
                  pRegEle->Id,RegList->Category, pRegEle->UChrp->Name, Emark );
        }

        pRegEle = pRegEle->xtEle ;
      } while( pRegEle != NULL );
    }
    if ( RegList->xtOtRegEle != NULL ) {

      pRegEle = RegList->xtOtRegEle;

      do {
        if ( pRegEle->UChrp != NULL || i == 1 ) {
          fprintf(out, "      Region Ot %3d Cat %1d %21s %s",
                  pRegEle->Id,RegList->Category, pRegEle->UChrp->Name, Emark );
        }
        else {
          fprintf(out, "      Region Ot %3d Cat %1d %21s %s",
                  pRegEle->Id,RegList->Category, pRegEle->UChrp->Name, Emark );
        }

        pRegEle = pRegEle->xtEle ;
      } while( pRegEle != NULL );
    }
  }
}

/*----- Grow.c ----- PrintRegions f*/
/*----- DocHeading */

```

```
extern int RegisterRegions( FILE *out,
                          struct Region_List *RegList )
/*-----EndDocHead-----*/
{
    struct Region_Element_List      *pRegEle;
    struct Event_Message            *NewMsg ;
    int obj_class_nbr, j ;
    int kTotal ;
    int inventory;
    int count= 1;
    double RegTime, IncrTime ;
    RegTime = RegisterStart ;
    /* Register & Associate the Instance of an Object with a Region */
    inventory = 0 ;
    kTotal = 0 ;

    if ( RegList != NULL ) {
        fprintf(out, "Region3 %2d Equip Gn %4d Ot %4d \n",
                RegList->Id, RegList->EquipGn, RegList->EquipOt );
        do {
            /* fill Geo Regions FILL */
            inventory = 0 ;
            count = 1;
            if ( RegList->xtGnRegEle != NULL ) {
                //fprintf(out, "registering objects for region:%d \n", RegList->Id);
                pRegEle = RegList->xtGnRegEle; /* get Green Region elements */
                do {
                    if ( pRegEle->UChrp != NULL ) {
                        //fprintf(out, "green region:%d \n", pRegEle->Id);

                        if ( pRegEle->UChrp->FedNode > 0 ) {
                            //fprintf(out, "registering starting instances:%d for green federate:%d \n",
                            //      instance_nbr, pRegEle->UChrp->FedNode);

                            for ( j=0; j<ObjectsInSOM; j++ ) { /* obj class */
                                if ( RegList->SubObjects[j] > 0 && /* The Region takes this object */
                                    pRegEle->UChrp->Objects[j] > 0 ) { /* and the Unit has this object */
                                    obj_class_nbr = j + OffsetObject ;
                                    pRegEle->UChrp->ObjectInstance[j] = instance_nbr ;
                                    IncrTime = RegisterIncrement * (double)instance_nbr;
                                    NewMsg = SetExtendEventMessage(
                                        1, /* RTIcommand, */
                                        1, /* SIMcommand, */
                                        RTI_REGISTER_INST, /* Action, */
                                        1, /* fedrtn_exname,*/
                                        pRegEle->UChrp->FedNode, /* Federate */
                                        obj_class_nbr, /* obj_class_nbr, */
                                        (instance_nbr++), /* obj_instance_nbr,*/
                                        0, /* interact_class_nbr, */
                                        0, /* interact_instance_nbr */
                                        0.0, /* fedrtn_time */
                                        RegList->Id, /* region_nbr, */
                                        0, /* routing_space_nbr */
                                        0, /* nbr_rcvd_msgs */
                                        0, /* nbr_sent_msgs */
                                        0.0, /* LBTS_time */
                                        (RegTime+IncrTime), /* lPhysicalTime, */
                                        (RegTime+IncrTime), /* lVirtualTime,*/
                                        "Init" ); /* just a note */
                                    NewMsg->Sim.UnitId = pRegEle->UChrp->Id;
                                    NewMsg->Sim.ExtentOfEffect = 0 ;
                                    NewMsg->Sim.InteraClass = 0 ;
                                    NewMsg->Sim.ObjectClass = 0 ;
                                    AddEvent( stdout, "InToRTI", NewMsg );
                                    inventory += 1 ;
                                } /*end if Object type is to exist in Region for this instance*/
                            }
                        }
                    }
                }
            }
        }
    }
}
```



```

        kTotal += 1 ;
        count ++;
    } /* end if federate */
} /* end if green region element */

    pRegEle = pRegEle->xtEle ;
} while( pRegEle != NULL ) ; /* && count < 50 ); */
}
//#if 0
    if ( RegList->xtOtRegEle != NULL ) {

        pRegEle = RegList->xtOtRegEle;

        do {
            if ( pRegEle->UChrp != NULL ) {
                if ( pRegEle->UChrp->FedNode > 0 ) {
                    for ( j=0; j<ObjectsInSOM; j++ ) { /* int class */
                        if ( RegList->SubObjects[j] > 0 && /* The Region takes this object */
                            pRegEle->UChrp->Objects[j] > 0 ) { /* and the Unit has this object */
                            obj_class_nbr = j + OffsetObject ;
                            pRegEle->UChrp->ObjectInstance[j] = instance_nbr ;
                            IncrTime = RegisterIncrement * (double)instance_nbr ;
                            NewMsg = SetExtendEventMessage(
                                1, /* RTIcommand, */
                                1, /* SIMcommand, */
                                RTI_REGISTER_INST, /* Action, */
                                1, /* fedrtn_exname,*/
                                pRegEle->UChrp->FedNode, /* Federate */
                                j, /* obj_class_nbr, */
                                (instance_nbr++), /* obj_instance_nbr,*/
                                0, /* interact_class_nbr, */
                                0, /* interact_instance_nbr */
                                0.0, /* fedrtn_time */
                                RegList->Id, /* region_nbr, */
                                0, /* routing_space_nbr */
                                0, /* nbr_rcvd_msgs */
                                0, /* nbr_sent_msgs */
                                0.0, /* LBTS_time */
                                (RegTime+IncrTime), /* lPhysicalTime, */
                                (RegTime+IncrTime), /* lVirtualTime, */
                                "Init" ); /* just a note */
                            NewMsg->Sim.UnitId = pRegEle->UChrp->Id;
                            NewMsg->Sim.ExtentOfEffect = 0 ;
                            NewMsg->Sim.InteraClass = 0 ;
                            NewMsg->Sim.ObjectClass = 0 ;
                            AddEvent( stdout, "InToRTI", NewMsg );
                            inventory += 1 ;
                        }
                    }
                }
                kTotal += 1 ;
            }
        }

        pRegEle = pRegEle->xtEle ;
    } while( pRegEle != NULL );

}
//#endif
    fprintf(out,"ForRegion %2d Registers %3d \n", RegList->Id, inventory );

    RegList = RegList->xtReg;
} while ( RegList != NULL ); /* end of regions list */
} /* if ( RegList != NULL ) {*/
return(kTotal);
}
/*----- DocHeading */

```

```
/* file: SOAcreat.c */
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include "soa_defs.h"
#include "soa_cnst.h"

#define UpperEventLimit 100000;

int CurNodesInFederation = 0;
int Duplicate_Event = 0 ;
unsigned short int SOAcXsubi[3];
int EventIdCounter = 0;
extern struct Event_Message *c_Event_Message( char *sptr ) ;

/* func_Name                *c_Comm_Net_Association          f*/
/*----- DocHeading */
```

```

extern struct Comm_Net_Association *c_Comm_Net_Association( char *sptr )
{
    struct Comm_Net_Association *tmp_ref;
    extern double erand48(unsigned short int X[3]);

    tmp_ref = ( struct Comm_Net_Association *)malloc( sizeof( struct Comm_Net_Association ) ) ;
    if ( errno == EINVAL ) {
        fprintf( stderr, " %s: (EINVAL) malloc has requested 0 bytes. c_Comm_Net_Association\n",sptr );
    }
    if ( errno == ENOMEM ) {
        fprintf( stderr, " %s: (ENOMEM) not enough storage space was available.
c_Comm_Net_Association\n", sptr );
        exit(-1); }

    tmp_ref->MaxSize      = MaxSizeMsg + MaxSizeMsg * erand48( SOAcXsubi) ;
    tmp_ref->MaxTime      = MaxReply - FuzzyReply * erand48( SOAcXsubi) ; /* * 3600.0 ; */
    tmp_ref->ReplyToCommandAt = -1.0 ;
    tmp_ref->ReplyToPeerAt   = -1.0 ;
    tmp_ref->SubModeChangeAt = -1.0 ;
    tmp_ref->SubModeOfActivity = 0 ;
    tmp_ref->SpecificUnit    = 0 ;
    tmp_ref->CmdOrdrp       = NULL ;
    tmp_ref->PeerOrdrp      = NULL ;
    tmp_ref->UChrp          = NULL;
    tmp_ref->ngep           = NULL;
    tmp_ref->CmNLP         = NULL;
    return (tmp_ref);
}

/* func_Name                      *c_Comm_Net_List          f*/
/*----- DocMethod */
extern struct Comm_Net_List *c_Comm_Net_List( char *sptr )
{
    struct Comm_Net_List *tmp_ref;
    tmp_ref = (struct Comm_Net_List *)malloc( sizeof( struct Comm_Net_List ) ) ;
    if ( errno == EINVAL ) {
        fprintf( stderr, " %s: (EINVAL) malloc has requested 0 bytes. c_Comm_Net_List\n",sptr );
    }
    if ( errno == ENOMEM ) {
        fprintf( stderr, " %s: (ENOMEM) not enough storage space was available.c_Comm_Net_List\n",
        sptr);
        exit(-1);
    }
    tmp_ref->ngep      = tmp_ref;
    tmp_ref->nOrder    = NULL;
    tmp_ref->ULstp     = NULL;
    return (tmp_ref);
}

/* func_Name                      *c_Truth_Group_List        f*/
/*----- DocMethod */
extern struct Truth_Group_List *c_Truth_Group_List( char *sptr )
{
    struct Truth_Group_List *tmp_ref;
    tmp_ref = (struct Truth_Group_List *)malloc( sizeof( struct Truth_Group_List ) ) ;
    if ( errno == EINVAL ) {
        fprintf( stderr, " %s: (EINVAL) malloc has requested 0 bytes. c_Truth_Group_List\n",sptr );
    }
    if ( errno == ENOMEM ) {
        fprintf( stderr, " %s: (ENOMEM) not enough storage space was available. c_Truth_Group_List\n",
        sptr );
        exit(-1); }
    tmp_ref->ngep      = tmp_ref;
    tmp_ref->TGLp      = tmp_ref;
    tmp_ref->NtDp      = NULL ;
    tmp_ref->ULstp     = NULL ;
    return (tmp_ref);
}

```

```

}
/* func_Name                                *c_Node_Table_Def      f*/
/*----- DocMethod */
extern struct Node_Table_Def *c_Node_Table_Def( char *sptr )
{
struct Node_Table_Def *tmp_ref;
tmp_ref = (struct Node_Table_Def *)malloc( sizeof( struct Node_Table_Def ) );
if ( errno == EINVAL ) {
    fprintf( stderr, " %s: (EINVAL) malloc has requested 0 bytes. c_Node_Table_Def\n", sptr ); }
if ( errno == ENOMEM ) {
    fprintf( stderr, " %s: (ENOMEM) not enough storage space was available. c_Node_Table_Def\n",
    sptr );
    exit(-1); }
tmp_ref->TGLst      = NULL;
tmp_ref->NdLstActp   = NULL;
tmp_ref->NLstp       = NULL;
tmp_ref->Id          = 0 ;
tmp_ref->Objects     = 0.0 ;
tmp_ref->NumCategorys = 1;
tmp_ref->Subscribed  = 0.0;
tmp_ref->Cpufactor   = 0.0 ;
return (tmp_ref);
}
/* func_Name                                *c_Node_List        f*/
/*----- DocMethod */
extern struct Node_List *c_Node_List( char *sptr )
{
struct Node_List *tmp_ref;
tmp_ref = (struct Node_List *)malloc( sizeof( struct Node_List ) );
if ( errno == EINVAL ) {
    fprintf( stderr, " %s: (EINVAL) malloc has requested 0 bytes. c_Node_List\n", sptr ); }
if ( errno == ENOMEM ) {
    fprintf( stderr, " %s: (ENOMEM) not enough storage space was available. c_Node_List\n", sptr );
    exit(-1); }
tmp_ref->NTindex     = -1 ;
tmp_ref->Subscribe    = 0.0;
tmp_ref->Reflect      = 0.0;
tmp_ref->Category     = 0 ;
tmp_ref->nlep         = NULL;
return (tmp_ref);
}
/* func_Name                                *c_Unit_Characteristics f*/

extern struct Unit_Characteristics *c_Unit_Characteristics( char *sptr ) /* SOAcreat.c pf*/
{
struct Unit_Characteristics *tmp_ref;
tmp_ref = (struct Unit_Characteristics *)malloc( sizeof( struct Unit_Characteristics ) );
if ( errno == EINVAL ) {
    fprintf( stderr, " %s: (EINVAL) malloc has requested 0 bytes. c_Unit_Characteristics\n", sptr );
}
if ( errno == ENOMEM ) {
    fprintf( stderr, " %s: (ENOMEM) not enough storage space was available.
c_Unit_Characteristics\n", sptr );
    exit(-1);
}
strcpy(tmp_ref->Name, "\0" );
tmp_ref->ReportRate   = 15.0 ; /* erand48( SOAcXsubi) ;*/
tmp_ref->OrderRate    = 3.0 ;
tmp_ref->FireRate     = 8.0 ;
tmp_ref->SenseRate    = 2.0 ;
tmp_ref->InLstp       = NULL ;
tmp_ref->CmNetp       = NULL ;
tmp_ref->Truthp       = NULL ;
tmp_ref->SvStkp       = NULL ;

```

```

tmp_ref->AltSvLp      = NULL ;
tmp_ref->ServLp       = NULL ;
tmp_ref->ULstp        = NULL ;
tmp_ref->RegOfUnit    = NULL ;
tmp_ref->ngep         = tmp_ref ;
return (tmp_ref);
}

/* func_Name                      *c_Unit_List      f*/
/*----- DocMethod */
extern struct Unit_List *c_Unit_List( char *sptr )
{
    struct Unit_List *tmp_ref;
    tmp_ref = (struct Unit_List *)malloc( sizeof( struct Unit_List ) ) ;
    if ( errno == EINVAL ) {
        fprintf( stderr, " %s: (EINVAL) malloc has requested 0 bytes. c_Unit_List\n", sptr ); }
    if ( errno == ENOMEM ) {
        fprintf( stderr, " %s: (ENOMEM) not enough storage space was available. c_Unit_List\n", sptr );
        exit(-1); }
    tmp_ref->nrep      = tmp_ref ;
    tmp_ref->UChrp     = NULL ;
    tmp_ref->UCmdp     = NULL ;
    tmp_ref->Subrdp    = NULL ;
    return (tmp_ref);
}

/* func_Name                      *c_InterestList    f*/
/*----- DocMethod */
extern struct InterestList *c_InterestList( char *sptr )
{
    struct InterestList *tmp_ref;
    tmp_ref = (struct InterestList *)malloc( sizeof( struct InterestList ) ) ;
    if ( errno == EINVAL ) {
        fprintf( stderr, " %s: (EINVAL) malloc has requested 0 bytes. c_InterestList\n", sptr ); }
    if ( errno == ENOMEM ) {
        fprintf( stderr, " %s: (ENOMEM) not enough storage space was available. c_InterestList\n",
        sptr );
        exit(-1); }
    tmp_ref->ngep      = NULL ;
    tmp_ref->UChrp     = NULL ;
    return (tmp_ref);
}

/* func_Name                      *c_Order_Packet    f*/
/*----- DocMethod */
extern struct Order_Packet *c_Order_Packet( char *sptr )
{
    struct Order_Packet *tmp_ref;
    tmp_ref = (struct Order_Packet *)malloc( sizeof( struct Order_Packet ) ) ;
    if ( errno == EINVAL ) {
        fprintf( stderr, " %s: (EINVAL) malloc has requested 0 bytes. c_Order_Packet\n", sptr ); }
    if ( errno == ENOMEM ) {
        fprintf( stderr, " %s: (ENOMEM) not enough storage space was available. c_Order_Packet\n",
        sptr );
        exit(-1); }
    tmp_ref->TimeToSend = 0.0 ;
    tmp_ref->TimeToAct  = 0.0 ;
    tmp_ref->ToActInHrs = 0.0 ;
    tmp_ref->PcktType   = 0 ; /* if not changed then Order will be dropped */
    tmp_ref->Size       = 0.0 ;
    tmp_ref->Entities   = 0.0 ;
    tmp_ref->TimeOnQueue = 0.0 ;
    tmp_ref->TravelStart = 0.0 ;
    tmp_ref->Priority    = 0 ;
    tmp_ref->To         = 0 ;
    tmp_ref->From       = 0 ;
    tmp_ref->Activity   = 3 ;

```

30 June 1999

```

tmp_ref->Order      = 0 ;
tmp_ref->Force       = 0 ;
tmp_ref->XferType    = 0 ;
tmp_ref->McNet       = 0 ;
tmp_ref->DestLocale  = 0 ;
tmp_ref->DestSP      = 0 ;
tmp_ref->DestCS      = 0 ;
tmp_ref->DestCpi     = 0 ;
tmp_ref->DestDVS     = 0 ;
tmp_ref->OrigLocale  = 0 ;
tmp_ref->OrigSP      = 0 ;
tmp_ref->OrigCS      = 0 ;
tmp_ref->OrigCpi     = 0 ;
tmp_ref->OrigDVS     = 0 ;
tmp_ref->MsgId       = 0 ;

tmp_ref->PvTruthp    = NULL ;
tmp_ref->nqep         = NULL ; /* These are used for queues (FIFO for now) */

return (tmp_ref);
}
/* func_Name                      *c_Task_List          f*/
/*----- DocMethod */
extern struct Task_List *c_Task_List( char *sptr )
{
struct Task_List *tmp_ref;
tmp_ref = (struct Task_List *)malloc( sizeof( struct Task_List ) ) ;
if ( errno == EINVAL ) {
    fprintf( stderr, " %s: (EINVAL) malloc has requested 0 bytes. c_Task_List\n", sptr ); }
if ( errno == ENOMEM ) {
    fprintf( stderr, " %s: (ENOMEM) not enough storage space was available. c_Task_List\n", sptr );
    exit(-1); }
tmp_ref->nqep         = NULL ;
tmp_ref->AppListp     = NULL ;
tmp_ref->Periodicp    = NULL ;
tmp_ref->Mode         = 0 ;
tmp_ref->Code         = 0 ;
return (tmp_ref);
}
/* func_Name                      c_Serv_Characteristics  f*/
/*----- DocMethod */
extern struct Serv_Characteristics *c_Serv_Characteristics( char *sptr )
{
struct Serv_Characteristics *tmp_ref;
tmp_ref = (struct Serv_Characteristics *)malloc( sizeof( struct Serv_Characteristics ) ) ;
if ( errno == EINVAL ) {
    fprintf( stderr, " %s: (EINVAL) malloc has requested 0 bytes. c_Serv_Characteristics\n", sptr );
    exit(-1); }
if ( errno == ENOMEM ) {
    fprintf( stderr, " %s: (ENOMEM) not enough storage space was available.
c_Serv_Characteristics\n", sptr );
    exit(-1); }
tmp_ref->nqep         = NULL ;
return (tmp_ref);
}
/* func_Name                      *c_Serv_List          f*/
/*----- DocMethod */
extern struct Serv_List *c_Serv_List( char *sptr )
{
struct Serv_List *tmp_ref;
tmp_ref = (struct Serv_List *)malloc( sizeof( struct Serv_List ) ) ;
if ( errno == EINVAL ) {
    fprintf( stderr, " %s: (EINVAL) malloc has requested 0 bytes. c_Serv_List\n", sptr ); }
if ( errno == ENOMEM ) {

```

30 June 1999

```

    fprintf( stderr, " %s: (ENOMEM) not enough storage space was available. c_Serv_List\n", sptr
);
    exit(-1); }
tmp_ref->nlep      = tmp_ref ;
tmp_ref->SChrp     = NULL ;
tmp_ref->TskLstp   = NULL ;
return (tmp_ref);
}
/* func_Name      *c_Serv_Stack      f*/
/*----- DocMethod */
extern struct Serv_Stack *c_Serv_Stack( char *sptr )
{
    struct Serv_Stack *tmp_ref;
    tmp_ref = (struct Serv_Stack *)malloc( sizeof( struct Serv_Stack ) ) ;
    if ( errno == EINVAL ) { fprintf( stderr, " %s: (EINVAL) malloc has requested 0 bytes.
c_Serv_Stack\n", sptr ); }
    if ( errno == ENOMEM ) { fprintf( stderr, " %s: (ENOMEM) not enough storage space was
available. c_Serv_Stack\n", sptr );
        exit(-1);
    }
    tmp_ref->SvLstp = NULL ;
    tmp_ref->nskp   = NULL ;
    return (tmp_ref);
}
/*-----*/
/*-----*/
/*-----*/
/* func_Name      *c_Region_Definition      f*/
/*----- DocMethod */
extern struct Region_Definition *c_Region_Definition( char *sptr )
{
    struct Region_Definition *tmp_ref;
    tmp_ref = (struct Region_Definition *)malloc( sizeof( struct Region_Definition ) ) ;
    if (errno == EINVAL){ fprintf(stderr," %s: (EINVAL) malloc has requested 0 bytes.
c_Region_Definition\n", sptr ); }
    if (errno == ENOMEM){ fprintf(stderr," %s: (ENOMEM) not enough storage space was available.
c_Region_Definition\n", sptr );
        exit(-1); }
    tmp_ref->high    = 0 ;
    tmp_ref->Commit  = 0.0 ;
    tmp_ref->other   = 0 ;
    return (tmp_ref);
}

/* func_Name      *c_Region_Node_Handle      f*/
/*----- DocMethod */
extern struct Region_Node_Handle *c_Region_Node_Handle( char *sptr )
{
    struct Region_Node_Handle *tmp_ref;
    tmp_ref = (struct Region_Node_Handle *)malloc( sizeof( struct Region_Node_Handle ) ) ;
    if (errno == EINVAL){ fprintf(stderr," %s: (EINVAL) malloc has requested 0 bytes.
c_Region_Node_Handle\n", sptr ); }
    if (errno == ENOMEM){ fprintf(stderr," %s: (ENOMEM) not enough storage space was available.
c_Region_Node_Handle\n", sptr );
        exit(-1); }
    tmp_ref->xtFed    = NULL ;
    tmp_ref->xtReg    = NULL ;
    return (tmp_ref);
}

/* func_Name      *c_Units_on_Node_List      f*/
/*----- DocMethod */
extern struct Units_on_Node_List *c_Units_on_Node_List( char *sptr )
{
    struct Units_on_Node_List *tmp_ref;
    tmp_ref = (struct Units_on_Node_List *)malloc( sizeof( struct Units_on_Node_List ) ) ;

```

30 June 1999

```

if (errno == EINVAL){fprintf(stderr," %s: (EINVAL) malloc has requested 0 bytes.
c_Units_on_Node_List\n",sptr ); }
if (errno == ENOMEM){fprintf(stderr," %s: (ENOMEM) not enough storage space was available.
c_Units_on_Node_List\n",sptr);
    exit(-1); }
tmp_ref->UChrp                = NULL ;
tmp_ref->xtUnitOnNode          = NULL ;
return (tmp_ref);
}

```

```

/* func_Name                *c_Nodes_wrt_Region_List                f*/
/*----- DocMethod */
extern struct Nodes_wrt_Region_List *c_Nodes_wrt_Region_List( char *sptr )
{
    struct Nodes_wrt_Region_List *tmp_ref;
    tmp_ref = (struct Nodes_wrt_Region_List *)malloc( sizeof( struct Nodes_wrt_Region_List ) );
    if (errno == EINVAL){fprintf(stderr," %s: (EINVAL) malloc has requested 0 bytes.
c_Nodes_of_Region_List\n",sptr ); }
    if (errno == ENOMEM){fprintf(stderr," %s: (ENOMEM) not enough storage space was available.
c_Nodes_of_Region_List\n",sptr);
        exit(-1); }
    tmp_ref->boundary          = 0 ;
    tmp_ref->NodeId            = 0 ;
    tmp_ref->NodeOfFed         = NULL ;
    tmp_ref->xtNodeWRTRegion   = NULL ;
    return (tmp_ref);
}

```

```

/* func_Name                *c_Nodes_of_Fed_List                f*/
/*----- DocMethod */
extern struct Nodes_of_Fed_List *c_Nodes_of_Fed_List( char *sptr )
{
    char string[Serv_Nam_M];
    struct Nodes_of_Fed_List *tmp_ref;
    tmp_ref = (struct Nodes_of_Fed_List *)malloc( sizeof( struct Nodes_of_Fed_List ) );
    if (errno == EINVAL){fprintf(stderr," %s: (EINVAL) malloc has requested 0 bytes.
c_Nodes_of_Fed_List\n",sptr ); }
    if (errno == ENOMEM){fprintf(stderr," %s: (ENOMEM) not enough storage space was available.
c_Nodes_of_Fed_List\n",sptr);
        exit(-1); }
    tmp_ref->Ratio = 0.0 ;
    tmp_ref->NumGn = 0 ;
    tmp_ref->NumOt = 0 ;
    tmp_ref->Initialized = 0 ;
    tmp_ref->NodeId = CurNodesInFederation + 1 ;
    CurNodesInFederation = tmp_ref->NodeId ;
    tmp_ref->box[0][0] = 0 ;
    tmp_ref->box[0][1] = 0 ;
    tmp_ref->box[1][0] = 0 ;
    tmp_ref->box[1][1] = 15 ;
    tmp_ref->box[2][0] = 10 ;
    tmp_ref->box[2][1] = 10 ;
    tmp_ref->box[3][0] = 10 ;
    tmp_ref->box[3][1] = 0 ;

```

```

    tmp_ref->Interact[OrderSOM] = 1 ;
    tmp_ref->Interact[ReportSOM] = 1 ;
    tmp_ref->Interact[FireSOM] = 1 ;
    tmp_ref->Interact[SenseSOM] = 1 ;
    tmp_ref->Interact[SupplySOM] = 1 ;

```

```

    tmp_ref->Objects[RouteSOM] = 1 ;
    tmp_ref->Objects[MissionSOM] = 1 ;
    tmp_ref->Objects[UnitTypeSOM] = 1 ;
    tmp_ref->Objects[PlanSOM] = 1 ;
    tmp_ref->Objects[HealthSOM] = 1 ;

```



```

tmp_ref->RegOnNode      = NULL ;
tmp_ref->UnitOnNode     = NULL ;
tmp_ref->xtNodeOfFed    = NULL ;
sprintf(string, "FedNode%3.3d",tmp_ref->NodeId );
strcpy(tmp_ref->Name, string);
return (tmp_ref);
}

```

```

/* func_Name          *c_Region_List          f*/
/*----- DocMethod */
extern struct Region_List *c_Region_List( char *sptr )
{
    struct Region_List *tmp_ref;
    tmp_ref = (struct Region_List *)malloc( sizeof( struct Region_List ) ) ;
    if (errno == EINVAL){fprintf(stderr," %s: (EINVAL) malloc has requested 0 bytes.
c_Region_List\n",sptr ); }
    if (errno == ENOMEM){fprintf(stderr," %s: (ENOMEM) not enough storage space was available.
c_Region_List\n",sptr);
    exit(-1);}
    tmp_ref->EquipGn      = 0 ;
    tmp_ref->EquipOt      = 0 ;
    tmp_ref->Id           = 0 ;
    tmp_ref->Category     = 0 ;
    tmp_ref->box[0][0]    = 0 ;
    tmp_ref->box[0][1]    = 0 ;
    tmp_ref->box[1][0]    = 0 ;
    tmp_ref->box[1][1]    = 15 ;
    tmp_ref->box[2][0]    = 10 ;
    tmp_ref->box[2][1]    = 10 ;
    tmp_ref->box[3][0]    = 10 ;
    tmp_ref->box[3][1]    = 0 ;
    tmp_ref->SubInteract [OrderSOM]      = 1 ;
    tmp_ref->SubInteract [ReportSOM]     = 1 ;
    tmp_ref->SubInteract [FireSOM]       = 1 ;
    tmp_ref->SubInteract [SenseSOM]      = 1 ;
    tmp_ref->SubInteract [SupplySOM]     = 1 ;

    tmp_ref->PubInteract [OrderSOM]      = 1 ;
    tmp_ref->PubInteract [ReportSOM]     = 1 ;
    tmp_ref->PubInteract [FireSOM]       = 1 ;
    tmp_ref->PubInteract [SenseSOM]      = 1 ;
    tmp_ref->PubInteract [SupplySOM]     = 1 ;

    tmp_ref->SubObjects [RouteSOM]       = 1 ;
    tmp_ref->SubObjects [MissionSOM]     = 1 ;
    tmp_ref->SubObjects [UnitTypeSOM]    = 1 ;
    tmp_ref->SubObjects [PlanSOM]        = 1 ;
    tmp_ref->SubObjects [HealthSOM]      = 1 ;

    tmp_ref->PubObjects [RouteSOM]       = 1 ;
    tmp_ref->PubObjects [MissionSOM]     = 1 ;
    tmp_ref->PubObjects [UnitTypeSOM]    = 1 ;
    tmp_ref->PubObjects [PlanSOM]        = 1 ;
    tmp_ref->PubObjects [HealthSOM]      = 1 ;
    tmp_ref->NodeWRTRegion = NULL ;
    tmp_ref->xtGnRegEle   = NULL ;
    tmp_ref->xtOtRegEle   = NULL ;
    tmp_ref->xtReg        = NULL ;
    return (tmp_ref);
}

```

```

/* func_Name          *c_Region_Element_List    f*/
/*----- DocMethod */
extern struct Region_Element_List *c_Region_Element_List( char *sptr )
{

```

30 June 1999

```

struct          Region_Element_List  *tmp_ref;
tmp_ref = (struct Region_Element_List *)malloc( sizeof( struct Region_Element_List ) );
if (errno == EINVAL){fprintf(stderr, " %s: (EINVAL) malloc has requested 0 bytes.
c_Region_Element_List\n",sptr ); }
if (errno == ENOMEM){fprintf(stderr, " %s: (ENOMEM) not enough storage space was available.
c_Region_Element_List\n",sptr);
exit(-1);}
tmp_ref->Id          = 0 ;
tmp_ref->UChrp       = NULL ;
tmp_ref->xtEle       = NULL ;
return (tmp_ref);
}

```

```

/* func_Name          *c_Unit_Region_List          f*/
/*----- DocMethod */

```

```

extern struct Unit_Region_List *c_Unit_Region_List( char *sptr )

```

```

{
struct          Unit_Region_List  *tmp_ref;
tmp_ref = (struct Unit_Region_List *)malloc( sizeof( struct Unit_Region_List ) );
if (errno == EINVAL){fprintf(stderr, " %s: (EINVAL) malloc has requested 0 bytes.
c_Region_Element_List\n",sptr ); }
if (errno == ENOMEM){fprintf(stderr, " %s: (ENOMEM) not enough storage space was available.
c_Unit_Region_List\n",sptr);
exit(-1);}
tmp_ref->xtReg          = NULL ;
tmp_ref->nxtRegOfUnit = NULL ;
return (tmp_ref);
}

```

```

/* func_Name          *c_Filter_Unit_List          f*/
/*----- DocMethod */

```

```

extern struct Filter_Unit_List *c_Filter_Unit_List( char *sptr )

```

```

{
struct          Filter_Unit_List *tmp_ref;
tmp_ref = (struct Filter_Unit_List *)malloc( sizeof( struct Filter_Unit_List ) );
if ( errno == EINVAL ) {fprintf( stderr, " %s:(EINVAL)malloc has requested 0 bytes.
c_Filter_Unit_List\n",sptr ); }
if ( errno == ENOMEM ) {fprintf(stderr, " %s:(ENOMEM)not enough storage space was available.
c_Filter_Unit_List\n", sptr );
exit(-1); }
tmp_ref->UChrp          = NULL ;
tmp_ref->nxtFilteredUnitp = NULL ;
return (tmp_ref);
}

```

```

/* func_Name          *c_Federate_Destination      f*/
/*----- DocMethod */

```

```

extern struct Federate_Destination *c_Federate_Destination( char *sptr )

```

```

{
struct          Federate_Destination *tmp_ref;
tmp_ref = (struct Federate_Destination *)malloc( sizeof( struct Federate_Destination ) );
if ( errno == EINVAL ) {fprintf( stderr, " %s:(EINVAL)malloc has requested 0 bytes.
c_Federate_Destination\n",sptr ); }
if ( errno == ENOMEM ) {fprintf(stderr, " %s:(ENOMEM)not enough storage space was available.
c_Federate_Destination\n", sptr );
exit(-1); }
tmp_ref->federate       = 0 ;
tmp_ref->next           = NULL ;
return (tmp_ref);
}

```

```

/* func_Name          *c_Event_Message          f*/
/*----- DocMethod */

```

```

extern struct Event_Message *c_Event_Message( char *sptr )

```

```

{
struct          Event_Message  *tmp_ref;
tmp_ref = (struct Event_Message *)malloc( sizeof( struct Event_Message ) );

```

30 June 1999

```

if ( errno == EINVAL ) {fprintf( stderr," %s:(EINVAL)malloc has requested 0 bytes.
c_Event_Message\n",sptr ); }
if ( errno == ENOMEM ) {fprintf(stderr, " %s:(ENOMEM)not enough storage space was available.
c_Event_Message\n", sptr );
    exit(-1); }

```

```

tmp_ref->Rti.rti_svc_nbr          = 0; /* action to perform */
tmp_ref->Rti.fedrtn_exname        = 7 * 70; /* name is number */
tmp_ref->Rti.federate_name        = 0; /* name is number of node */
tmp_ref->Rti.fedrtn_type          = 0;
tmp_ref->Rti.fedrtn_save_label    = 0; /* name is number */
tmp_ref->Rti.obj_class_nbr        = 0;
tmp_ref->Rti.obj_instance_nbr     = 0;
tmp_ref->Rti.interact_class_nbr   = 0;
tmp_ref->Rti.interact_instance_nbr= 0;
tmp_ref->Rti.tag_name             = 0; /* name is number */
tmp_ref->Rti.passive_subscription_indicator = 0;
tmp_ref->Rti.fedrtn_time          = 0;
tmp_ref->Rti.transportation_type  = 0;
tmp_ref->Rti.routing_space_nbr    = 0;
tmp_ref->Rti.region_nbr           = 0;
tmp_ref->WhoGetsIt.RTI            = 0;
tmp_ref->WhoGetsIt.SIM            = 0;
tmp_ref->Color.LowestUnprocessedTSO = 0.0 ;
tmp_ref->Color.ColorTag           = 0;
tmp_ref->Color.Sent                = 0;
tmp_ref->Color.Received            = 0;
tmp_ref->Color.Boundary            = 0;

```

```

tmp_ref->Time.PhysicalTime        = 0.0 ;
tmp_ref->Time.VirtualTime         = 0.0 ;
tmp_ref->Time.Label               = 0;
if ( Duplicate_Event != 1 ) {
    EventIdCounter = (EventIdCounter + 1) % UpperEventLimit ;
}
tmp_ref->Time.UniqueMsgId          = EventIdCounter ;
tmp_ref->Time.OutEnter             = 0.0 ;
tmp_ref->Time.OutService           = 0.0 ;
tmp_ref->Time.OutComplete          = 0.0 ;
tmp_ref->Time.RTIEnter            = 0.0 ;
tmp_ref->Time.RTIService           = 0.0 ;
tmp_ref->Time.RTIComplete          = 0.0 ;
tmp_ref->Time.TsoEnter            = 0.0 ;
tmp_ref->Time.TsoRtService         = 0.0 ;
tmp_ref->Time.TsoService           = 0.0 ;
tmp_ref->Time.TsoComplete          = 0.0 ;
tmp_ref->Time.TsoRtComplete        = 0.0 ;
tmp_ref->destinations_list         = NULL ;
tmp_ref->Sim.UnitId               = 0;
tmp_ref->Sim.Effect                = 0;
tmp_ref->Sim.ExtentOfEffect        = 0;
tmp_ref->Sim.InteraClass           = 0;
tmp_ref->Sim.ObjectClass           = 0;
tmp_ref->nqep                     = NULL ;
tmp_ref->Marked_Delete             = 0;
tmp_ref->marker_mode               = 0;
return (tmp_ref);
}

```

```

/*----- MsgQPckt.c ----- SelectMsg f*/

```

```

/*----- DocMethod */

```

```

extern struct Event_Message *c_Duplicate_Event_Message( struct Event_Message *A)
{
    struct Event_Message *Dupl ;
    Duplicate_Event = 1 ;
    if ( A != NULL ) {

```

30 June 1999

```
Dupl = c_Event_Message( "Dupl" );
Dupl->Color= A->Color;
Dupl->WhoGetsIt = A->WhoGetsIt ;
Dupl->Rti  = A->Rti  ;
Dupl->Sim  = A->Sim  ;
Dupl->Time = A->Time ;
}
Duplicate_Event = 0 ;
return( Dupl );
}
/* ----- DocHeading */
```

```

/* file: SOAdestr.c */
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include "soa_defs.h"
#include "soa_cnst.h"

/*
/* ----- DocMethod */
extern void d_Comm_Net_Association(struct Comm_Net_Association *dptr )
{
    free( dptr );
}
/* ----- DocMethod */
extern void d_Comm_Net_List( struct Comm_Net_List *sptr )
{
    free( sptr );
}
/* ----- DocMethod */
extern void d_Truth_Group_List( struct Truth_Group_List *sptr )
{
    free( sptr );
}
/* ----- DocMethod */
extern void d_Node_Table_Def( struct Node_Table_Def *sptr )
{
    free( sptr );
}
/* ----- DocMethod */
extern void d_Node_List( struct Node_List *sptr )
{
    free( sptr );
}
/* ----- DocMethod */
extern void d_Unit_Characteristics( struct Unit_Characteristics *sptr )
{
    free( sptr );
}
/* ----- DocMethod */
extern void d_Unit_List( struct Unit_List *sptr )
{
    free( sptr );
}
/* ----- DocMethod */
extern void d_Event_Message( struct Event_Message *sptr )
{
    free( sptr );
}
/* ----- DocMethod */
extern void d_Order_Packet( struct Order_Packet *sptr )
{
    struct Order_Packet *pPkt, *cPkt;
    if ( sptr->nqep != NULL ) {
        cPkt = sptr ;
        while ( cPkt->nqep != NULL ) {
            pPkt = cPkt->nqep;
            free(cPkt);
            cPkt = pPkt;
        }
        free(cPkt);
    }
    else { free(sptr); }
    sptr = NULL ;
}
/* ----- DocMethod */
extern void d_Filter_Unit_List( struct Filter_Unit_List *sptr )
{
    struct Filter_Unit_List *pPkt, *cPkt;
    if (
        sptr->nxtFilteredUnitp != NULL &&
        sptr != sptr->nxtFilteredUnitp ) { /* initialed to point to self */
        cPkt = sptr ;
        do {
            pPkt = cPkt->nxtFilteredUnitp;
            free(cPkt);
            cPkt = pPkt;
        } while (
            cPkt->nxtFilteredUnitp != NULL &&
            cPkt != cPkt->nxtFilteredUnitp ) ;
        free(cPkt);
    }
    else { free(sptr); }
}

```

30 June 1999

```

    sptr = NULL ;
}
/* ----- DocMethod */
extern void d_Unit_Region_List( struct Unit_Region_List *sptr )
{
    free( sptr );
}
/* ----- DocMethod */
extern void d_Region_Element_List( struct Region_Element_List *sptr )
{
    free( sptr );
}

/* ----- DocMethod */
extern void d_Task_List( struct Task_List *sptr )
{
    free( sptr );
}
/* ----- DocMethod */
extern void d_Serv_Characteristics( struct Serv_Characteristics *sptr )
{
    free( sptr );
}
/* ----- DocMethod */
extern void d_Serv_List( struct Serv_List *sptr )
{
    free( sptr );
}
/* ----- DocMethod */
extern void d_Serv_Stack( struct Serv_Stack *sptr )
{
    free( sptr );
}
/* ----- DocMethod */
extern void d_Federate_Destination( struct Federate_Destination *sptr )
{
    struct Federate_Destination *ptr, *fptr ;
    ptr = sptr ;
    while( ptr != NULL ) {
        fptr = ptr;
        ptr = ptr->next ;
        free(fptr);
    }
}
/*----- DocHeading ---*

```

/* file: Sim.c */

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <math.h>
#include <unistd.h>
#include "soa_defs.h"
#include "soaGcnst.h"
#include "proto.h"
#include "rti_services.h"
```

```
extern      void      rtimgr_cleanup();
extern      void      statsmgr_setup_stats_tables();
extern      void      statsmgr_print_accum_totals();
FILE        *QueuesAtEpoch ;
```

```
static      double    DeltaQueues = 2.0 ;
```

```
/*-----                               DocHeading      Grow                               */
```

30 June 1999

```

main(int argc, char *argv[]) /* int GreenBattalions, int OtherBattalions)*/
{
/*-----EndDocHead-----*/
/*----- cut here for main -----*/
/*----- cut here for main -----*/
/*----- cut here for main -----*/
/*----- cut here for main -----*/
//struct Filter_Unit_List *FilteredUnits, *FilteredList;
struct Event_Message *NewMsg ;
struct Unit_Characteristics *UnCharG, *UnCharO, *UnChar;
//struct Unit_Characteristics *LastGnUnitAssigned, *LastOtUnitAssigned ;
//struct Unit_Characteristics *UnChar, *UnCharX;
//FILE *out;
//FILE *in_orders;
//struct Region_Definition Regions;
struct Region_Node_Handle RegionNodeHandles;
//struct Region_List *RegList ;
//struct Region_Element_List *ElementOfRegion, *pRegEle ;

//int EquipOtatLevel, EquipGnatLevel;
int Total_Force_Battalions ;
int i,j,k,l,m,n,o,p,q,r,s,t,u,v ;
int QueueEmpty ;
//FILE *in_orders;
int AfterStart ;
double ptemp, dtemp, intervaltemp;
//char justtext[128];
char str[128] ;
char ch;
struct Event_Message *EVptr;
void ResetEcheleon();
struct Unit_Region_List *RegOfUnit ;
extern void QueuesTest() ;
FILE *LogFile, *UtilFile ;
//struct Event_Message *NextEvent(FILE *out, double *R, double S); /* NextMsg MsgQPckt.c pf*/
extern void PrintQueueHistory(FILE *out);
double RealSystemTime, CpuTimeOfService;
double OnceADelta ;
Region_List_Type *regions;
int instance_nbr = 0;
int count = 1;
int Epochs = 1;

OnceADelta = START ;

i=j=k=l=m=n=o=p=q=r=s=t=u=v=1;
i=j+k+l+m+n+o+p+q+r+s+t+u+v;

printf("%3d\n",i);
QueuesAtEpoch = fopen("QueuesAtEpoch.out", "w");

LogFile = fopen("QueuesAtDelta.out", "w");
UtilFile = fopen("ResourceUtil.out", "w");
fprintf( LogFile, " opened file \n" );
Total_Force_Battalions = MAXBATTALIONS; /* TMP changed to 1 from 12 */

/* printf("%8x b\n", &UnitList ); */
if ( argc < 2) { printf("grow 'Force 1 Battalions' ::: Default to 12 ::: ) n");
/* exit(-86); */

Total_Force_Battalions = 12;
Total_Force_Battalions = MAXBATTALIONS;
}
else {
Total_Force_Battalions = atoi( argv[1] );
printf("Force 1 %3d battalions \n", Total_Force_Battalions);
}
}

```


30 June 1999

```

LalaInit(10, 0);
QueuesInitialize();

/* initialize statistics tables */
statsmgr_setup_stats_tables();

/* Army->Corps->Division->Brigade->Battalion->Company->Platoon->Squad */

GrowInitArmy( Total_Force_Battalions, Total_Force_Battalions,
              &RegionNodeHandles) ;

/* Initialize RTI HERE PUBLISH & SUBSCRIBE */
//NewMsg = SetEventMessage( int Action, int Federate,
//                          int RTIcommand, int SIMcommand, double lPhysicalTime,
//                          double lVirtualTime, char *sptr ) ;
/*extern struct Event_Message *SetExtendEventMessage(
    1      int RTIcommand,
    2      int SIMcommand,
    3      int Action,                      rti_svc_nbr
    4      int fedrtn_exname,
    5      int Federate,                    federate_name
    6      int obj_class_nbr,
    7      int obj_instance_nbr,
    8      int interact_class_nbr,
    9      int interact_instance_nbr ,
    10     double fedrtn_time ,
    1      int region_nbr,
    2      int routing_space_nbr ,
    3      int nbr_rcvd_msgs ,
    4      int nbr_sent_msgs ,
    5      double LBTS_time ,
    6      double lPhysicalTime,
    7      double lVirtualTime,          char *sptr )
*/

Initialize_RTI();

/* CREATE a FEDERATION */
NewMsg = SetExtendEventMessage(
    1,                                     /* RTIcommand, */
    1,                                     /* SIMcommand, */
    RTI_CREATE_FEDEX,                    /* Action, */
    1,                                     /* fedrtn_exname,*/
    1,                                     /* Federate */
    j,                                     /* obj_class_nbr, */
    0,                                     /* obj_instance_nbr,*/
    0,                                     /* interact_class_nbr, */
    0,                                     /* interact_instance_nbr */
    0.0,                                  /* fedrtn_time */
    0,                                     /* region_nbr, */
    0,                                     /* routing_space_nbr */
    0,                                     /* nbr_rcvd_msgs */
    0,                                     /* nbr_sent_msgs */
    0.0,                                  /* LBTS_time */
    0.0,                                  /* lPhysicalTime, */
    0.0,                                  /* lVirtualTime,*/
    "Init" ) ;                           /* just a note */
AddEvent( stdout, "InToRTI", NewMsg );

/* temp set to 5- join fedex, for federate 1 to 10      TEN federates for paper */
for ( i=1; i<= SCENARIOSlimitsOnFederates; i++) {
NewMsg = SetExtendEventMessage(1,1,RTI_JOIN_FEDEX,1,i, 0,0, 0,0, 0.0,
    0,0,0,0, 0.001, 0.001, 0.0, "Init" ) ;
AddEvent( stdout, "InToRTI", NewMsg );
}
/* create regions      13 regions; TMP only 4 */

```

30 June 1999

```

for ( i=1; i<=SCENARIOLimitsOnFederates; i++) { /* TMP make only 5 regions for now */
NewMsg = SetExtendEventMessage(1,1, RTI_CREATE_UPDATE_REGION,1,1, 0,0, 0,0, 0,0,
    i,0,0,0, 0.01, 0.01, 0.0, "Init" );
AddEvent( stdout, "InToRTI", NewMsg );
}
/* publish ;          i for Fed j for Object class for K regions for now */

k = PublishByFederate(stdout, RegionNodeHandles.xtFed );

printf( "PublishbyFedNode Number of   %5d \n", k );
printf("Press enter to Continue\n");
gets(str);

// for ( i=1; i<6; i++) { /* TMP only 5 federates; each federate */

//     for ( j=0; j<5; j++) { /* each object class */
//         NewMsg = SetExtendEventMessage(1,1,RTI_PUBLISH_OBJCLSS,1,i, j,0, 0,0, 0.0, k,0,0,0, 0.0,
// 0.0, 0.0, "Init" );
//         AddEvent( stdout, "InToRTI", NewMsg );
//     } /* end for object classes */

//     for ( j=5; j<10; j++) { /* each interact class */
//         NewMsg = SetExtendEventMessage(1,1, RTI_PUBLISH_INTCLSS,1,i, 0,j, 0,0,0.0, k,0,0,0, 0.0,
// 0.0, 0.0, "Init" );
//         AddEvent( stdout, "InToRTI", NewMsg );
//     } /* end for interact classes */

// } /* end for federates */

/* subscribe,          i for Fed j for Object class for K regions for now */

k = SubscribeByFederate(stdout, RegionNodeHandles.xtFed );

printf( "SubscribeNode Number of subscriptions %5d \n", k );
printf("Press enter to Continue\n");
gets(str);

/* REGISTER instances */
UnChar = RegionNodeHandles.UnitGn;

k = RegisterRegions( stdout, RegionNodeHandles.xtReg );
printf( " Number of entities registered %5d \n", k );
printf("Press enter to Continue\n");
gets(str);

/* Initialize RTI HERE REGISTER WITH REGIONS */

/* end of initialize RTI HERE */
printf("\n");
//QueuesTest();
InitSimModel( &RegionNodeHandles );
QueuesPrint(stdout, -1);
printf("Press enter to Continue\n");
gets(str);
i=0;
AfterStart = 1 ;
dtemp = -1.0;
intervaltemp = START ;
QueueEmpty = 1 ;
fprintf(QueuesAtEpoch,"Replicate %3d time %12.5f events %9d at start \n",
    Epochs,dtemp, count);
    QueuesPrint(QueuesAtEpoch,Epochs) ;
    QueuesPrint(QueuesAtEpoch,Epochs) ;
Epochs +=1 ;
do {
    /* multiple replication loop */
    i=0;

```

```

do { /* the EPOCH loop */
    i +=1;
    ptemp = dtemp ;
    dtemp = EventManager(stdout, LogFile, &QueueEmpty, &RegionNodeHandles );

    //PrintQueueHistory(stdout);
    if ( i > 1000 ) {
        printf(
            " %s Cycle in the Main # %5d   time %8.3f \n",
            " . . . . . ", count, dtemp );
        i = 0;
        QueuesPrint(stdout,0) ;
        if ( AfterStart && dtemp > START ) { /* this only happens once */
            QueuesPrint(QueuesAtEpoch,EPOCHS) ;
            AfterStart = 0 ;
            SetBaseResourceTime();
        }
    }
    if ( dtemp > OnceADelta ) {
        QueuesPrint(LogFile, Epochs) ;
        OnceADelta += DeltaQueues;
        PrintUtilizationResourceTime(UtilFile,DeltaQueues,EPOCHS);
    }
    //PrintEventsProcessed( stdout);
    //PrintEventsInSystem( stdout);

    // QueuesPrint(stdout, -1) ;
    // printf("Press enter to Continue\n");
    // gets(str);

    count++;

} while( QueueEmpty    && dtemp <  intervaltemp + EPOCH );

printf("Press enter to TERMINATE \n");
//gets(str);
fprintf(QueuesAtEpoch,"Replicate %3d   time %12.5f   events %9d \n",
    Epochs,dtemp, count);
    QueuesPrint(QueuesAtEpoch, Epochs) ;
    Epochs +=1 ;

intervaltemp = intervaltemp + EPOCH ;

// PrintRTIEchelon( stdout,   RegionNodeHandles.UnListGn);
// PrintRTIEchelon( stdout,   RegionNodeHandles.UnListOt);

PrintEventsProcessed( stdout);

statsmgr_print_accum_totals();
printf("Run another interval? Press Y or N\n");
printf("Run another interval? Press Y or N\n");
printf("Run another interval? Press Y or N\n");
i = 0 ;
// ch = getchar();
if (ch == 'n' || ch== 'N')
{ i = 1 ; }
else { rtimgr_clear(); }
} while ( dtemp <  ENDitALL && i == 0 ) ;

    rtimgr_final_cleanup(); /* cristl's cleanup rti rtn */

PrintEventsProcessed( stdout);
PrintRTIIInstanceEchelon( stdout, RegionNodeHandles.UnListGn);
PrintRTIIInstanceEchelon( stdout, RegionNodeHandles.UnListOt);

} /* end of grow main */
/*-----

```

DocHeading */

```

/* file: SimModel.c */
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <math.h>
#include "soa_defs.h"
#include "soa_cnst.h"
#include "proto.h"
#include "statsmgr.h"
#include "rti_services.h"

#define REQUESTOR 4
#define WAITonTIME 0

extern void tm_clear_LBTS_info();
extern void rtimgr_clear_reduction_network_info();
extern double erand48(unsigned short int X[3]);
unsigned int FederationPastColor = 0;
static int AdvanceRequest = 0 ;
static int InitSimOnce = 1 ;
static unsigned short int SimXsubi[3];
static double RepeatAdvance = START ;
static int AdvanceRequestGranted[SCENARIOILimitsOnFederates+1] ;
static double StartAdvanceRequest = START;
static int CollectionInterval = 1 ;
STATISTIC_CPM_TYPE TimeAdvance;
STATISTIC_CPM_TYPE ObjectUpdate[ SCENARIOILimitsOnFederates+1 ];
STATISTIC_CPM_TYPE ObjectInteraction[ SCENARIOILimitsOnFederates+1 ];

static int TimeStatistic ;
static int UpdateStatistic[SCENARIOILimitsOnFederates+1] ;
static int InteraStatistic[SCENARIOILimitsOnFederates+1] ;

extern void NextCollectionInterval() {
    CollectionInterval += 1;
}
/*----- EventManager.c ----- f*/
/*----- DocHeading */

```

30 June 1999

```

extern double SimModel( struct Event_Message *ptr, double PhysicalTime,
                        struct Region_Node_Handle *RNH )
{
    struct Event_Message      *NewPtr ;
    struct Unit_Characteristics *UnChar ;
    double    ServiceTime, dtemp , EventTime;
    struct Nodes_of_Fed_List  *FedPtr ;
    int                      i,j, ThisFed, aFed;
    unsigned int              CurColor;
    struct Units_on_Node_List *UnitOnNode;
    struct Unit_Region_List   *RegOfUnit ;
    int    NumberCreated ;
    char str[12] ;

    NumberCreated = 0 ;
    ThisFed = ptr->Rti.federate_name ;
    CurColor = GetColorTag(ThisFed);

    if ( StartAdvanceRequest + TIMEadvTIMEOUT > PhysicalTime ) {
        AdvanceRequest = 0 ;
        /* reset reporting counts and status */
        //      rtimgr_clear_reduction_network_info();
        //      tm_clear_LBTS_info();
    }

    //if ( ptr->Color.ColorTag != CurColor) { and So what only matters in RTI
    //  printf("ColorTag Does not Match Fed %2d Color %3d Is %3d %8.3f\n",
    //    ThisFed, CurColor, ptr->Color.ColorTag, PhysicalTime );
    //}

    if ( ptr->Rti.rti_svc_nbr == RTI_DISCVR_OBJ ) {
        if (ptr->Sim.UnitId > 0 ) {
            UnChar = FindUnit( ptr->Sim.UnitId );
            if ( UnChar != NULL && ThisFed == UnChar->FedNode ) { UnChar->Discovered += 1 ; }
            if ( InitSimOnce ) {
                for ( j=0; j<=SCENARIOILimitsOnFederates; j++) {
                    AdvanceRequestGranted[ j ] = 1 ;
                }
                InitSimOnce = 0 ;
            }
            ServiceTime = 0.0005;
        }
    }
    else if ( ptr->Rti.rti_svc_nbr == RTI_QUERY_FED_LBTS ) {
        if (ptr->Sim.UnitId > 0 ) {
            UnChar = FindUnit( ptr->Sim.UnitId );
            if ( UnChar != NULL && ThisFed == UnChar->FedNode ) { UnChar->Updated += 1 ; }
            ServiceTime = 0.002;
        }
        d_Event_Message( ptr );
    }
    else if ( ptr->Rti.rti_svc_nbr == RTI_REFLECT_ATTRIB ) {
        if (ptr->Sim.UnitId > 0 ) {
            UnChar = FindUnit( ptr->Sim.UnitId );
            if ( UnChar != NULL && ThisFed == UnChar->FedNode ) {
                UnChar->Updated -= 1 ;
                dtemp = PhysicalTime - ptr->Time.VirtualTime ;
                statsmgr_collect_statistic( UpdateStatistic[ThisFed], dtemp );
                //      fprintf(stdout,
                //        "RUnitId%02d, %5d, : %s, at, %9.4f, on,%2d, updt, %9.4f, Delta, %9.4f, Obj Reflected
                //        \n",
                //        CollectionInterval, UnChar->Id, UnChar->Name, PhysicalTime, ThisFed, ptr-
                //        >Time.VirtualTime, dtemp );
            }
            ServiceTime = 0.002;
        }
        d_Event_Message( ptr );
    }
}

```

30 June 1999

```

}
else if ( ptr->Rti.rti_svc_nbr == RTI_RECEIVE_INT ) {
    if (ptr->Sim.UnitId > 0 ) {
        UnChar = FindUnit( ptr->Sim.UnitId );
        if ( UnChar != NULL && ThisFed == UnChar->FedNode ) {
            UnChar->Interaction += 1 ;
            dtemp = PhysicalTime - ptr->Time.VirtualTime ;
            statmgr_collect_statistic( InteraStatistic[ThisFed], dtemp);
            // fprintf(stdout,
            // "VUnitId%02d, %5d, : %s, at, %9.4f, on,%2d, evnt, %9.4f, Delta, %9.4f, Obj Received
\n",
            // CollectionInterval,UnChar->Id, UnChar->Name, PhysicalTime,ThisFed, ptr-
>Time.VirtualTime, dtemp );
        }
        ServiceTime = 0.002;
    }
    d_Event_Message( ptr );
}
else if ( ptr->Rti.rti_svc_nbr == RTI_TIME_ADV_GRANT ) {
    if ( FederationPastColor < CurrentFederationColor() ) {
        TestForDiffColor( CurrentFederationColor() );
        FederationPastColor = CurrentFederationColor() ;
    }
    fprintf(stdout,
    "TmUnitId%02d, TimeAdvance granted at, %8.5f, start, %8.5f, diff, %10.5f,
GRANTED\n",
    CollectionInterval, PhysicalTime, StartAdvanceRequest, PhysicalTime-
StartAdvanceRequest);
    if ( WAITonTIME && ThisFed == REQUESTOR ) fprintf(stdout," Press ENTER to continue\n" );
    if ( WAITonTIME && ThisFed == REQUESTOR ) gets(str);
    AdvanceRequestGranted[ThisFed] = 1 ;
    if ( ThisFed == REQUESTOR ) {
        AdvanceRequest = 0 ; /* to reset after it has been granted */
    }
    ServiceTime = 0.00124;
    dtemp = PhysicalTime - StartAdvanceRequest;
    statmgr_collect_statistic( TimeStatistic, dtemp );
    if (ptr->Sim.UnitId > 0 ) {
        UnChar = FindUnit( ptr->Sim.UnitId );
        // AdvanceRequest = 0 ; /* to reset after it has been granted */
    }
    d_Event_Message( ptr );
}
else if ( ptr->Sim.Effect == ORDER ) {
    if (ptr->Sim.UnitId > 0 ) { UnChar = FindUnit( ptr->Sim.UnitId); }
    ServiceTime = 0.002 ;
    if ( UnChar != NULL && ThisFed == UnChar->FedNode && UnChar ) {
        if ( UnChar->Designation == WarFighter ) {
            dtemp = 1.4425*60.0 / HIGHFACTOR; }
        else { dtemp = 1.4425*60.0 ; }
        UnChar->OrderRate = PhysicalTime + dtemp * triangle( 0.92 ) ;
        ptr->Time.PhysicalTime = UnChar->OrderRate ;
        ptr->Time.VirtualTime = UnChar->OrderRate ;
        ptr->Rti.fedrtm_time = UnChar->OrderRate ;
        CreateInteraction( ptr, PhysicalTime, RNH, UnChar->OrderRate,
            UnChar, ServiceTime, OrderSOM ) ;
        AddEvent(stdout, "TSO", ptr ); /* Unit Next Event Time */
        UnChar->LastTime = PhysicalTime ;
    }
    else { d_Event_Message( ptr ); } /* eliminate this message */
}
else if ( ptr->Sim.Effect == REPORT ) { /* this is a SUPPLY interaction */
    if (ptr->Sim.UnitId > 0 ) { UnChar = FindUnit( ptr->Sim.UnitId); }
    ServiceTime = 0.002 ;
    if ( UnChar != NULL && ThisFed == UnChar->FedNode ) {
        if ( UnChar->Designation == WarFighter ) {
            dtemp = 1.4425*60.0 / HIGHFACTOR; }
        else { dtemp = 1.4425*60.0 ; }
    }
}

```

30 June 1999

```

        else { dtemp = 1.4425*60.0 ; }
        dtemp = UnChar->ReportRate;
        UnChar->ReportRate = PhysicalTime + dtemp * triangle( 0.92 ) ;
        ptr->Time.PhysicalTime = UnChar->ReportRate ;
        ptr->Time.VirtualTime = UnChar->ReportRate ;
        ptr->Rti.fedrtm_time = UnChar->ReportRate ;
        CreateInteraction( ptr, PhysicalTime, RNH, dtemp,
                           UnChar, ServiceTime, SupplySOM ) ;
        AddEvent(stdout, "TSO", ptr ); /* Unit Next Event Time */
        UnChar->LastTime = PhysicalTime ;
    }
    else { d_Event_Message( ptr ); } /* eliminate this message */
}
else if ( ptr->Sim.Effect == FIRE ) {
    if (ptr->Sim.UnitId > 0 ) { UnChar = FindUnit( ptr->Sim.UnitId);}
    ServiceTime = 0.002 ;
    if ( UnChar != NULL && ThisFed == UnChar->FedNode ) {
        if ( UnChar->Designation == WarFighter ) {
            dtemp = 1.802*60.0 / HIGHFACTOR; }
        else { dtemp = 1.802*60.0 ; }
        dtemp = UnChar->FireRate;
        UnChar->FireRate = PhysicalTime + dtemp * triangle( 0.92 ) ;
        ptr->Time.PhysicalTime = UnChar->FireRate ;
        ptr->Time.VirtualTime = UnChar->FireRate ;
        ptr->Rti.fedrtm_time = UnChar->FireRate ;
        CreateInteraction( ptr, PhysicalTime, RNH, dtemp,
                           UnChar, ServiceTime, FireSOM ) ;
        AddEvent(stdout, "TSO", ptr ); /* Unit Next Event Time */
        UnChar->LastTime = PhysicalTime ;
    }
    else { d_Event_Message( ptr ); } /* eliminate this message */
}
else if ( ptr->Sim.Effect == SENSE ) {
    if (ptr->Sim.UnitId > 0 ) { UnChar = FindUnit( ptr->Sim.UnitId);}
    ServiceTime = 0.002 ;
    if ( UnChar != NULL && ThisFed == UnChar->FedNode ) {
        if ( UnChar->Designation == WarFighter ) {
            dtemp = 0.636*60.0 / HIGHFACTOR; }
        else { dtemp = 0.636*60.0 ; }
        dtemp = UnChar->SenseRate;
        UnChar->SenseRate = PhysicalTime + dtemp * triangle( 0.92 ) ;
        ptr->Time.PhysicalTime = UnChar->SenseRate ;
        ptr->Time.VirtualTime = UnChar->SenseRate ;
        ptr->Rti.fedrtm_time = UnChar->SenseRate ;
        CreateInteraction( ptr, PhysicalTime, RNH, dtemp,
                           UnChar, ServiceTime, SenseSOM ) ;
        AddEvent(stdout, "TSO", ptr ); /* Unit Next Event Time */
        UnChar->LastTime = PhysicalTime ;
    }
    else { d_Event_Message( ptr ); } /* eliminate this message */
}
else if ( ptr->Sim.Effect == MOVE ) {
    if (ptr->Sim.UnitId > 0 ) { UnChar = FindUnit( ptr->Sim.UnitId);}
    ServiceTime = 0.002 ;
    if ( UnChar != NULL && ThisFed == UnChar->FedNode ) {
        if ( UnChar->Designation == WarFighter ) {
            dtemp = 0.778*60 / HIGHFACTOR; }
        else { dtemp = 0.778*60 ; }
        dtemp = UnChar->MoveRate;
        UnChar->MoveRate = PhysicalTime + dtemp * triangle( 0.92 ) ;
        ptr->Time.PhysicalTime = UnChar->MoveRate ;
        ptr->Time.VirtualTime = UnChar->MoveRate ;
        ptr->Rti.fedrtm_time = UnChar->MoveRate ;
        UpdateEntity( ptr, PhysicalTime, RNH, dtemp, UnChar, ServiceTime ) ;
        AddEvent(stdout, "TSO", ptr ); /* Unit Next Event Time */
        UnChar->LastTime = PhysicalTime ;
    }
}

```

30 June 1999

```

    else { d_Event_Message( ptr ); } /* eliminate this message */
}
else if ( ptr->Sim.Effect == SUPPLY ) {
    if ( ptr->Sim.UnitId > 0 ) { UnChar = FindUnit( ptr->Sim.UnitId ); }
    if ( UnChar == NULL ) { d_Event_Message( ptr ); }
    ServiceTime = 0.0005 ;
}
// else if ( ptr->Sim.InteraClass > 0 ) {
//     ServiceTime = 0.0005 ;
//     d_Event_Message( ptr );
// }
// else if ( ptr->Sim.ObjectClass > 0 ) {
//     ServiceTime = 0.0005 ;
//     d_Event_Message( ptr );
// }
else if ( ptr->Sim.UnitId > 0 ) {
    UnChar = FindUnit( ptr->Sim.UnitId );
    NumberCreated = 0 ;
    ServiceTime = 0.00259;
    if ( UnChar != NULL && ThisFed == UnChar->FedNode ) {
        fprintf(stdout,
            "IDONLYdelete, %5d, : %s, at, %9.4f, on,%2d, next, %9.4f, Delta, %9.4f, \n",
            UnChar->Id, UnChar->Name, PhysicalTime, ThisFed, UnChar->LastTime, dtemp );
        d_Event_Message( ptr );
    }
}
else {
    aFed = ptr->Rti.federate_name ;
    //fprintf(stdout, "SimModel message just an Interval Mark on node %3d \n", aFed );
    ptr->WhoGetsIt.RTI = 0 ;
    ptr->WhoGetsIt.SIM = 1 ;
    ptr->Color.ColorTag = GetColorTag( ptr->Rti.federate_name );
    ptr->Time.PhysicalTime = PhysicalTime + REPEATTIMEADVANCE ;
    ptr->Time.VirtualTime = PhysicalTime + REPEATTIMEADVANCE ;
    AddEvent(stdout, "TSO", ptr ); /* Time Step */

    if ( AdvanceRequest == 0 && ptr->Rti.federate_name == REQUESTOR &&
        RepeatAdvance <= PhysicalTime ) {
        AdvanceRequest = 1 ;
        /* reset reporting counts and status just to be sure */
        // rtimgr_clear_reduction_network_info();
        // tm_clear_LBTS_info();
        RepeatAdvance = PhysicalTime + REPEATTIMEADVANCE ;
        StartAdvanceRequest = PhysicalTime ;
        NewPtr = SetExtendEventMessage(
            1, /* RTIcommand, */
            0, /* SIMcommand, */
            RTI_TIME_ADV_RQST, /* Action, */
            1, /* fedrtn_exname, */
            ptr->Rti.federate_name, /* Federate */
            3, /* obj_class_nbr, */
            0, /* obj_instance_nbr, */
            0, /* interact_class_nbr, */
            0, /* interact_instance_nbr */
            PhysicalTime+0.0005, /* fedrtn_time */
            0, /* region_nbr, */
            0, /* routing_space_nbr */
            0, /* nbr_rcvd_msgs */
            0, /* nbr_sent_msgs */
            0.0, /* LBTS_time */
            PhysicalTime+0.0005, /* lPhysicalTime, */
            PhysicalTime+0.0005, /* lVirtualTime, */
            "Init" ); /* just a note */

        AddEvent( stdout, "InToRTI", NewPtr );

        //QueuesPrint( stdout, -2);

```


30 June 1999

```
        //fprintf(stdout," Press Enter After TIME ADVANCE REQUEST    for fed %3d \n",
aFed);
        // gets(str);
        }
        ServiceTime = 0.0005          ;

        //  for ( aFed = 1; aFed <=SCENARIOLimitsOnFederates; aFed++ ) {
        //  aFed = PickSome(1,5) ;
```

```
    } /* for the else */
```

```
                                //return( SimModelTEST( ptr) );
return(  ServiceTime );
}
```

```
/*----- EventManager.c ----- f*/
/*-----                               DocHeading */
```

30 June 1999

```

extern void UpdateEntity( struct Event_Message *ptr, double PhysicalTime,
                        struct Region_Node_Handle *RNH, double EventTime,
                        struct Unit_Characteristics *UnChar, double ServiceTime )
{
    struct Event_Message      *NewPtr ;
    double      dtemp;
    struct Nodes_of_Fed_List  *FedPtr ;
    int      i,j, ThisFed, aFed;
    unsigned int      CurColor;
    struct Units_on_Node_List *UnitOnNode;
    struct Unit_Region_List  *RegOfUnit ;
    int      NumberCreated ;
    char str[12] ;

    NumberCreated = 0 ;
    ThisFed = ptr->Rti.federate_name ;
    CurColor = GetColorTag(ThisFed);
    aFed = ptr->Rti.federate_name;

    UnChar = FindUnit( ptr->Sim.UnitId );
    dtemp = PhysicalTime - UnChar->LastTime;

    //      fprintf(stdout,
    //      "UUnitId, %5d, %s, at, %9.4f, on,%2d, next, %9.4f, Delta, %9.4f, Obj ",
    //      UnChar->Id, UnChar->Name, PhysicalTime, ThisFed, UnChar->LastTime, dtemp );
    UnChar->Updated += 1 ;

    aFed = ptr->Rti.federate_name;
    FedPtr = FindNode( aFed, RNH->xtFed );
    for( j = 0; j < ObjectsInSOM; j++ ) {
        if ( UnChar->ObjectInstance[j] > 0 &&
            FedPtr->Objects[j] > 0 ) {
            //      fprintf( stdout, "%1d:%5d,", j, UnChar->ObjectInstance[j] );
            NewPtr = SetExtendEventMessage(
                1, /* RTIcommand, */
                1, /* SIMcommand, */
                RTI_UPDATE_ATTRIB, /* Action, */
                1, /* fedrtn_exname,*/
                aFed, /* Federate */
                (OffsetObject+j), /* obj_class_nbr, */
                UnChar->ObjectInstance[j], /* ?obj_instance_nbr,*/
                0, /* interact_class_nbr, */
                0, /* interact_instance_nbr */
                PhysicalTime, /* fedrtn_time */
                0, /* region_nbr, */
                0, /* routing_space_nbr */
                0, /* nbr_rcvd_msgs */
                0, /* nbr_sent_msgs */
                0.0, /* LBTS_time */
                PhysicalTime + ServiceTime, /* lPhysicalTime, */
                EventTime, /* lVirtualTime,*/
                "Init" ); /* just a note */
            NewPtr->Sim.UnitId = UnChar->Id ;
            AddEvent( stdout, "InToRTI", NewPtr );
            NumberCreated += 1;
        } /* if an instance for this object exists */
        //      else { fprintf(stdout,"-," ); }
    }
    //      fprintf(stdout,"Updates %2d \n", NumberCreated );
}
/*----- EventManager.c ----- f*/
/*----- DocHeading */

```

30 June 1999

```

extern void CreateInteraction( struct Event_Message *ptr, double PhysicalTime,
                             struct Region_Node_Handle *RNH, double EventTime,
                             struct Unit_Characteristics *UnChar, double ServiceTime,
                             int Type )
{
    struct Event_Message      *NewPtr ;
    double      dtemp;
    struct Nodes_of_Fed_List  *FedPtr ;
    int      i,j, ThisFed, aFed;
    unsigned int      CurColor;
    struct Units_on_Node_List *UnitOnNode;
    struct Unit_Region_List  *RegOfUnit ;
    int      NumberCreated ;
    char str[12] ;

    NumberCreated = 0 ;
    ThisFed = ptr->Rti.federate_name ;
    CurColor = GetColorTag(ThisFed);

    // fprintf(stdout,
    // "IUnitId, %5d,: %s, at, %9.4f, on,%2d, next, %9.4f, Delta, %9.4f, Int ",
    //          UnChar->Id, UnChar->Name, PhysicalTime, ThisFed, EventTime, dtemp );
    aFed = ptr->Rti.federate_name;
    FedPtr = FindNode( aFed, RNH->xtFed );

    for ( j=0; j< InteractionsInSOM; j++){
        // fprintf(stdout,"%1d:", (OffsetInteraction+j) );
        if ( UnChar->RegOfUnit != NULL &&
            FedPtr->Interact[j] > 0    && j == Type ) {
            RegOfUnit = UnChar->RegOfUnit;
            while ( RegOfUnit != NULL ) {
                // fprintf(stdout,"%1d:",RegOfUnit->xtReg->Id);
                NewPtr = SetExtendEventMessage(
                    1,                                /* RTIcommand, */
                    1,                                /* SIMcommand, */
                    RTI_SEND_INT,                     /* Action, */
                    1,                                /* fedrtn_exname,*/
                    aFed,                              /* Federate */
                    0,                                /* obj_class_nbr, */
                    0,                                /* obj_instance_nbr,*/
                    (OffsetInteraction+j),             /* interact_class_nbr, */
                    0,                                /* interact_instance_nbr */
                    PhysicalTime,                     /* fedrtn_time */
                    RegOfUnit->xtReg->Id,              /* region_nbr, */
                    0,                                /* routing_space_nbr */
                    0,                                /* nbr_rcvd_msgs */
                    0,                                /* nbr_sent_msgs */
                    0.0,                              /* LBTS_time */
                    PhysicalTime + ServiceTime,        /* lPhysicalTime, */
                    EventTime,                        /* lVirtualTime,*/
                    "Sch" );                          /* just a note */
                NewPtr->Sim.UnitId = UnChar->Id ;
                AddEvent( stdout, "InToRTI", NewPtr );
                NumberCreated += 1;
                RegOfUnit = RegOfUnit->nxtRegOfUnit ;    /* Next Region NEXT */
            } /* while applies to any regaion */
        } /* if this interaction applies */
        // else { fprintf(stdout,"--," );
        // }
    } /* for all interactions */
    // fprintf(stdout,"SendInt %2d, \n", NumberCreated );

}
/*----- EventManager.c ----- f*/
/*----- DocHeading */

```

30 June 1999

```

extern void InitSimModel( struct Region_Node_Handle *RNH )
{
    struct Event_Message      *ptr, *NewPtr ;
    int                        j, aFed, NumberCreated;
    char str[12];
    struct Unit_Characteristics *UnChar ;
    struct Nodes_of_Fed_List   *FedPtr ;
    struct Units_on_Node_List  *UnitOnNode;
    struct Unit_Region_List    *RegOfUnit ;

    NumberCreated = 0;

    TimeStatistic = initialize_statistic(&TimeAdvance,      "TimeAdvFED", 0 );

    for( j=0; j<SCENARIOILimitsOnFederates ; j++) {

        UpdateStatistic[j+1] = initialize_statistic(&ObjectUpdate[j+1],      "ObjUpdaFED", j );
        InteraStatistic[j+1] = initialize_statistic(&ObjectInteraction[j+1], "ObjInteFED", j );

        ptr = SetEventMessage( 0, (j+1), 0,1, START,
                                START, "InSM");

        ptr->Sim.UnitId          = -1 ;
        ptr->Sim.Effect          = -1 ;

        ptr->Sim.ExtentOfEffect  = -1 ;
        ptr->Sim.InteraClass     = 0 ;
        ptr->Sim.ObjectClass     = 0 ;
        ptr->Color.Boundary      = 25 +j ; /* ColorTag */
        AddEvent( stdout, "TSO" , ptr );
        aFed = j +1 ;
        FedPtr = FindNode( aFed, RNH->xtFed );
        if ( FedPtr != NULL ) {
            UnitOnNode = FedPtr->UnitOnNode;
            while( UnitOnNode != NULL ) {
                NumberCreated += 1;
                if ( UnitOnNode->UChrp->Echeleon >= 4 && UnitOnNode->UChrp->Echeleon < 6 ) {
                    /* create an event for each ORDER REPORT FIRE SENSE SUPPLY */
                    UnitOnNode->UChrp->LastTime = UnitOnNode->UChrp->OrderRate ;
                    //
                    fprintf(stdout,
                        //      "CreateEvent %5d for fed %02d at %9.3f      %s      %4d for future %9.3f \n",
                        //          NumberCreated,aFed, 0.0, UnitOnNode->UChrp->Name,
                        //              UnitOnNode->UChrp->Equipment, UnitOnNode->UChrp->LastTime);
                    if ( UnitOnNode->UChrp->Designation == WarFighter ) {
                        UnitOnNode->UChrp->OrderRate = UnitOnNode->UChrp->OrderRate / HIGHFACTOR;
                    }
                    NewPtr = SetExtendEventMessage(
                        0,                                /* RTIcommand, */
                        1,                                /* SIMcommand, */
                        77,                               /* Action, not for RTI */
                        1,                                /* fedrtn_exname,*/
                        aFed,                             /* Federate */
                        0,                                /* obj_class_nbr, */
                        0,                                /* obj_instance_nbr,*/
                        -1,                               /* interact_class_nbr, */
                        0,                                /* interact_instance_nbr */
                        UnitOnNode->UChrp->OrderRate ,     /* fedrtn_time */
                        -1,                               /* region_nbr, */
                        0,                                /* routing_space_nbr */
                        0,                                /* nbr_rcvd_msgs */
                        0,                                /* nbr_sent_msgs */
                        0.0,                              /* LBTS time */
                        UnitOnNode->UChrp->OrderRate+START, /* lPhysicalTime, */
                        UnitOnNode->UChrp->OrderRate+START, /* lVirtualTime,*/
                        "Sch" ) ;                          /* just a note */
                    UnitOnNode->UChrp->OrderRate          += START ;
                }
                UnitOnNode = UnitOnNode->NextUnitOnNode;
            }
        }
    }
}

```

```

/* ORDER REPORT FIRE SENSE SUPPLY */
NewPtr->Sim.Effect = ORDER ;
NewPtr->Sim.UnitId = UnitOnNode->UChrp->Id ;
AddEvent( stdout, "TSO", NewPtr );
}
/* REPORT */
if ( UnitOnNode->UChrp->Echeleon == 6 ) {
    if ( UnitOnNode->UChrp->Designation == WarFighter ) {
        UnitOnNode->UChrp->ReportRate = UnitOnNode->UChrp->ReportRate / HIGHFACTOR;
        UnitOnNode->UChrp->FireRate = UnitOnNode->UChrp->FireRate / HIGHFACTOR;
        UnitOnNode->UChrp->SenseRate = UnitOnNode->UChrp->SenseRate / HIGHFACTOR;
        UnitOnNode->UChrp->MoveRate = UnitOnNode->UChrp->MoveRate / HIGHFACTOR;
    }
    // fprintf(stdout,
    //      "CreateEvent %5d for fed %02d at %9.3f      %s      %4d for future %9.3f \n",
    //      NumberCreated,aFed, 0.0, UnitOnNode->UChrp->Name,
    //      UnitOnNode->UChrp->Equipment, UnitOnNode->UChrp->ReportRate);
    UnitOnNode->UChrp->LastTime = UnitOnNode->UChrp->ReportRate ;
    NewPtr = SetExtendEventMessage(
        0, /* RTIcommand, */
        1, /* SIMcommand, */
        77, /* Action, not for RTI */
        1, /* fedrtn_exname,*/
        aFed, /* Federate */
        0, /* obj_class_nbr, */
        0, /* obj_instance_nbr,*/
        -1, /* interact_class_nbr, */
        0, /* interact_instance_nbr */
        UnitOnNode->UChrp->ReportRate , /* fedrtn_time */
        -1, /* region_nbr, */
        0, /* routing_space_nbr */
        0, /* nbr_rcvd_msgs */
        0, /* nbr_sent_msgs */
        0.0, /* LBTS_time */
        UnitOnNode->UChrp->ReportRate +START, /* lPhysicalTime, */
        UnitOnNode->UChrp->ReportRate +START, /* lVirtualTime,*/
        "Sch" ) ; /* just a note */
    UnitOnNode->UChrp->ReportRate += START ;
    NewPtr->Sim.Effect = REPORT ;
    NewPtr->Sim.UnitId = UnitOnNode->UChrp->Id ;
    AddEvent( stdout, "TSO", NewPtr );
}
/* FIRE */
if ( UnitOnNode->UChrp->FireRate > UnitOnNode->UChrp->LastTime ) {
    UnitOnNode->UChrp->LastTime = UnitOnNode->UChrp->FireRate ;
}
NewPtr = SetExtendEventMessage(
    0, /* RTIcommand, */
    1, /* SIMcommand, */
    77, /* Action, not for RTI */
    1, /* fedrtn_exname,*/
    aFed, /* Federate */
    0, /* obj_class_nbr, */
    0, /* obj_instance_nbr,*/
    -1, /* interact_class_nbr, */
    0, /* interact_instance_nbr */
    UnitOnNode->UChrp->FireRate , /* fedrtn_time */
    -1, /* region_nbr, */
    0, /* routing_space_nbr */
    0, /* nbr_rcvd_msgs */
    0, /* nbr_sent_msgs */
    0.0, /* LBTS_time */
    UnitOnNode->UChrp->FireRate +START, /* lPhysicalTime, */
    UnitOnNode->UChrp->FireRate +START, /* lVirtualTime,*/
    "Sch" ) ; /* just a note */
    UnitOnNode->UChrp->FireRate += START ;
    NewPtr->Sim.Effect = FIRE ;
    NewPtr->Sim.UnitId = UnitOnNode->UChrp->Id ;

```

30 June 1999

```

AddEvent( stdout, "TSO", NewPtr );

/* SENSE */
if ( UnitOnNode->UChrp->SenseRate > UnitOnNode->UChrp->LastTime ) {
    UnitOnNode->UChrp->LastTime = UnitOnNode->UChrp->SenseRate ;
}
NewPtr = SetExtendEventMessage(
    0, /* RTIcommand, */
    1, /* SIMcommand, */
    77, /* Action, not for RTI */
    1, /* fedrtn_exname, */
    aFed, /* Federate */
    0, /* obj_class_nbr, */
    0, /* obj_instance_nbr, */
    -1, /* interact_class_nbr, */
    0, /* interact_instance_nbr */
    UnitOnNode->UChrp->SenseRate, /* fedrtn_time */
    -1, /* region_nbr, */
    0, /* routing_space_nbr */
    0, /* nbr_rcvd_msgs */
    0, /* nbr_sent_msgs */
    0.0, /* LBTS time */
    UnitOnNode->UChrp->SenseRate+START, /* lPhysicalTime, */
    UnitOnNode->UChrp->SenseRate+START, /* lVirtualTime, */
    "Sch" ) ; /* just a note */
UnitOnNode->UChrp->SenseRate += START ;
NewPtr->Sim.Effect = SENSE ;
NewPtr->Sim.UnitId = UnitOnNode->UChrp->Id ;
AddEvent( stdout, "TSO", NewPtr );

/* MOVE */
if ( UnitOnNode->UChrp->MoveRate > UnitOnNode->UChrp->LastTime ) {
    UnitOnNode->UChrp->LastTime = UnitOnNode->UChrp->MoveRate ;
}
NewPtr = SetExtendEventMessage(
    0, /* RTIcommand, */
    1, /* SIMcommand, */
    77, /* Action, not for RTI */
    1, /* fedrtn_exname, */
    aFed, /* Federate */
    0, /* obj_class_nbr, */
    0, /* obj_instance_nbr, */
    -1, /* interact_class_nbr, */
    0, /* interact_instance_nbr */
    UnitOnNode->UChrp->MoveRate , /* fedrtn_time */
    -1, /* region_nbr, */
    0, /* routing_space_nbr */
    0, /* nbr_rcvd_msgs */
    0, /* nbr_sent_msgs */
    0.0, /* LBTS time */
    UnitOnNode->UChrp->MoveRate +START, /* lPhysicalTime, */
    UnitOnNode->UChrp->MoveRate +START, /* lVirtualTime, */
    "Sch" ) ; /* just a note */
UnitOnNode->UChrp->MoveRate += START ;
NewPtr->Sim.Effect = MOVE ;
NewPtr->Sim.UnitId = UnitOnNode->UChrp->Id ;
AddEvent( stdout, "TSO", NewPtr );
} /* Platoon Level of the Echelon */
/* for Echelon */
UnitOnNode = UnitOnNode->xtUnitOnNode ;
} /* while a unit on the node */

// QueuesPrint( stdout,-2);
// fprintf(stdout," Press Enter during Event Loading of Federates %4d for fed %3d \n",
NumberCreated, aFed);
// gets(str);
} /* if node selected for nodes */
} /* for all Federates */
QueuesPrint( stdout,-2);

```

30 June 1999

```
//      fprintf(stdout," Press Enter After Event Loading of Federates      %4d for fed %3d \n",
NumberCreated, aFed);
//      gets(str);
```

```
}
```

```
/*----- SimModel.c ----- f*/
/*----- DocHeading */
```

30 June 1999

```

extern void InitSimModelTest( struct Region_Node_Handle *RNH )
{
    struct Event_Message      *ptr, *NewPtr ;
    int                        j, aFed, NumberCreated;
    char str[12];
    struct Unit_Characteristics *UnChar ;
    struct Nodes_of_Fed_List   *FedPtr ;
    struct Units_on_Node_List  *UnitOnNode;
    struct Unit_Region_List    *RegOfUnit ;

    NumberCreated = 0;
    for( j=0; j<SCENARIOILimitsOnFederates ; j++) {
        ptr = SetEventMessage( 0, (j+1), 0,1, 2.50,
                               2.50, "InSM");

        ptr->Sim.UnitId      = -1 ;
        ptr->Sim.Effect      = -1 ;

        ptr->Sim.ExtentOfEffect = -1 ;
        ptr->Sim.InteraClass   = 0 ;
        ptr->Sim.ObjectClass   = 0 ;
        ptr->Color.Boundary    = 25 +j ; /* ColorTag */
        AddEvent( stdout, "TSO" , ptr );
        aFed = j +1 ;
        FedPtr = FindNode( aFed, RNH->xtFed );
        if ( FedPtr != NULL ) {
            UnitOnNode = FedPtr->UnitOnNode;
            while(UnitOnNode != NULL) {
                NumberCreated += 1;
                if ( UnitOnNode->UChrp->Echeleon >= 4 ) {
                    // fprintf(stdout,
                    //      "CreateEvent %5d for fed %02d at %9.3f      %s      %4d for future %9.3f \n",
                    //      NumberCreated,aFed, 0.0, UnitOnNode->UChrp->Name,
                    //      UnitOnNode->UChrp->Equipment, UnitOnNode->UChrp->LastTime);
                    NewPtr = SetExtendEventMessage(
                        0, /* RTIcommand, */
                        1, /* SIMcommand, */
                        77, /* Action, not for RTI */
                        1, /* fedrtn_exname,*/
                        aFed, /* Federate */
                        0, /* obj_class_nbr, */
                        0, /* obj_instance_nbr,*/
                        -1, /* interact_class_nbr, */
                        0, /* interact_instance_nbr */
                        UnitOnNode->UChrp->LastTime, /* fedrtn_time */
                        -1, /* region_nbr, */
                        0, /* routing_space_nbr */
                        0, /* nbr_rcvd_msgs */
                        0, /* nbr_sent_msgs */
                        0.0, /* LBTS_time */
                        UnitOnNode->UChrp->LastTime+START, /* lPhysicalTime, */
                        UnitOnNode->UChrp->LastTime+START, /* lVirtualTime,*/
                        "Sch" ) ; /* just a note */
                    UnitOnNode->UChrp->LastTime += START ;
                    NewPtr->Sim.UnitId = UnitOnNode->UChrp->Id ;
                    AddEvent( stdout, "TSO", NewPtr );

                } /* for Echelon */
                UnitOnNode = UnitOnNode->xtUnitOnNode ;
            } /* while a unit on the node */

            QueuesPrint( stdout,-2);
            // fprintf(stdout," Press Enter during Event Loading of Federates      %4d for fed %3d \n",
            NumberCreated, aFed);
            // gets(str);
            } /* if node selected for nodes */

```


30 June 1999

```
} /* for all Federates */  
//   QueuesPrint( stdout,-2);  
//   fprintf(stdout," Press Enter After Event Loading of Federates    %4d for fed %3d \n",  
NumberCreated, aFed);  
//   gets(str);
```

```
}  
/*----- DocHeading */
```

30 June 1999

```

/* file: UnitCharFile.c */
/*-- UnitCharacter */
/*-----EndDocHead---*/

#include "soa_defs.h"
#include "soa_cnst.h"

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>
/* direct to Units with ids */
extern int Level ;
int First_Friendly = 1 ;
int Num_Friends = 0 ;
int Num_Others = 0 ;
int First_Other = 1 ;
struct Unit_Characteristics *FriendsIndex[Friendlies] ; /* Global to these methods */
struct Unit_Characteristics *OtherIndex[Not_So];

int ucLcLcounter, UnitChar_num ;
/*----- GrowHier.c ----- MaxEchelon f*/
/*----- DocHeading */

```

```

extern int MaxEcheleon(FILE *out, struct Unit_Characteristics *UnCrA, char *str)
/*-----EndDocHead---*/
{
    struct Unit_Characteristics *A;
    int i, MaxA_Ech ;

    A = UnCrA;
    i = 0 ;
    do { if ( A->Echeleon > i && A->Echeleon < 8 ) {
        i = A->Echeleon;
        if (out != NULL) {fprintf(out, "%20s %17s %1d\n", str, A->Name, A->Echeleon );}
    }
        A = A->ngep ;
    } while ( A != NULL );
    MaxA_Ech = i ; i = 0 ;
    return (MaxA_Ech);
}

/*----- UnitChar.c ----- Initialize_Friends f*/
/*----- DocMethod */
extern void Initialize_Friends()
/*-----EndDocHead---*/
{ int i; First_Friendly = 0; for(i=0; i<Friendlies; i++ ) { FriendsIndex[i] = NULL; } }

/*----- UnitChar.c ----- Initialize_Others f*/
/*----- DocMethod */
extern void Initialize_Others()
/*-----EndDocHead---*/
{ int i; First_Other = 0; for(i=0; i<Not_So; i++ ) { OtherIndex[i] = NULL; } }

/*----- UnitChar.c ----- UnitCharacter f*/
/*----- DocMethod */
extern int UnitCharacter( struct Unit_Characteristics *uptr,
                        char *line, int Who_is_it )
/*-----EndDocHead---*/
{
    int i,j;
    FILE *in ;
    extern int csv(char *s, char *p[], char *d);
    char *list[128];
    extern void Print_UnitC(FILE *out,
        struct Unit_Characteristics *luptr, char *Emark );

    ucLcLcounter = 0;
    iUcp fprintf( stderr, "%3d UnitChar:%s \n", UnitChar_num, line);
    if (First_Friendly == 1) { Initialize_Friends(); }
    if (First_Other == 1) { Initialize_Others(); }

    j = csv( line, list, "," );
    if ( j > 0 ) {
        /* Yes the values need to be right */
        uptr->Id = atoi( list[ 1 ] ) ;
        uptr->Force = Who_is_it ;
        uptr->Activity = atoi( list[ 2 ] ) ;
        uptr->Designation = atoi( list[ 3 ] ) ;
        uptr->Subordinates = atoi( list[ 4 ] ) ;
        uptr->Equipment = atoi( list[ 5 ] ) ;
        uptr->Personnel = atoi( list[ 6 ] ) ;
        uptr->Rolled_Equipment = atoi( list[ 7 ] ) ;
        uptr->Rolled_Personnel = atoi( list[ 8 ] ) ;
        uptr->DataSize = atoi( list[ 9 ] ) ;
        uptr->ReportRate = atof( list[ 10 ] ) ;
        uptr->CpuNode = atoi( list[ 11 ] ) ;
        uptr->Controller = atoi( list[ 12 ] ) ;
        uptr->Echeleon = atoi( list[ 13 ] ) ;
    }
}

```

30 June 1999

```

/* if ( uptr->Equipment > 0 && uptr->Personnel > 0 ) {
    uptr->CrewByEntity = ((double)uptr->Personnel )/ ((double)uptr->Equipment); }
else if ( uptr->Rolled_Equipment > 0 ) {
    if ( uptr->Rolled_Personnel > 0 ) {
        uptr->CrewByEntity =
            ((double)uptr->Rolled_Personnel )/ ((double)uptr->Rolled_Equipment); }
    else {
        uptr->CrewByEntity = ((double)uptr->Rolled_Equipment) / 4.0 ;    }
    }
*/
strcpy( uptr->Name,          list[ 0 ] );
UnitChar_num += 1;
/* printf("UnitCharacter:");    Print_UnitC ( stderr, uptr, "\n" ); */

for ( i=0; i<j; i++) { free( list[i] ); }

/* build the direct access vector based on force type */

if ( Who_is_it == Friends ) {
    if ( uptr->Id > 0 && uptr->Id < Friendlies ) {
        FriendsIndex[uptr->Id] = uptr ; /*printf(" Wrote to friends direct inde \n");*/
    }
    else if (uptr->Id > Friendlies ) {
        fprintf( stderr,
            " UnitCharacteristics(%3d) direct index id is too big %1d Friendly\n",
            UnitChar_num, uptr->Id );
        exit(-100);
    }
}
else {
    if ( uptr->Id > 0 && uptr->Id < Not_So ) {
        OtherIndex[uptr->Id] = uptr ;
    }
    else if (uptr->Id > Not_So ) {
        fprintf( stderr,
            " UnitCharacteristics(%3d) direct index id is too big %1d Other \n",
            UnitChar_num, uptr->Id );
        exit(-100);
    }
}
} /* there was data */

return(j);
} /* end of UnitCharacter */

/*----- UnitChar.c ----- Print_UnitChar_File f*/
/*----- DocHeading */

```

30 June 1999

```

extern void Print_UnitC_File( char filename[],
                             struct Unit_Characteristics *uptr )
/*-----EndDocHead---*/
{
FILE *out ;
struct Unit_Characteristics *luptr;

out = fopen(filename, "w");
luptr = uptr ;

while ( luptr != NULL) {

iUCp { fprintf(out, "%8.8x %8.8x %8.8x %8.8x %8.8x %8.8x ",
               luptr,
               luptr->CmNetp,
               luptr->Truthp,
               luptr->ServLp,
               luptr->ULstp,
               luptr->ngep ); }

fprintf(out,
        "Id %4d Act %1d, Dsg %1d, S %4d, Eq %3d, P %3d, REq%4d, RPe%4d, Sz%4d, Cr%7.5f, %2d, Ctl %2d
        Ech %1d %s\n",
        luptr->Id,
        luptr->Activity,
        luptr->Designation,
        luptr->Subordinates,
        luptr->Equipment,
        luptr->Personnel,
        luptr->Rolled_Equipment,
        luptr->Rolled_Personnel,
        luptr->DataSize,
        luptr->ReportRate,
        luptr->FedNode,
        luptr->Controller,
        luptr->Echeleon,
        luptr->Name );

        luptr = luptr->ngep ;
    }
fclose(out);
}

/*----- UnitChar.c ----- Print_UnitC f*/
/*----- DocMethod */
extern void Print_UnitC( FILE *out,
                        struct Unit_Characteristics *luptr, char *Emark )
/*-----EndDocHead---*/
{
    if (luptr != NULL) {
        iUCp { fprintf(out, "%8.8x %8.8x %8.8x %8.8x %8.8x %8.8x ",
                       luptr,
                       luptr->CmNetp,
                       luptr->Truthp,
                       luptr->ServLp,
                       luptr->ULstp,
                       luptr->ngep ); }

        fprintf(out,
            "Id %4d Act %1d, Dsg %1d, S %4d, Eq %3d, P %3d, REq%4d, RPe%4d, Sz%4d, Cr%7.5f, %2d, Ctl %2d
            %s%s",
            luptr->Id,
            luptr->Activity,
            luptr->Designation,
            luptr->Subordinates,
            luptr->Equipment,
            luptr->Personnel,
            luptr->Rolled_Equipment,

```

30 June 1999

```

        luptr->Rolled_Personnel,
        luptr->DataSize,
        luptr->ReportRate,
        luptr->FedNode,
        luptr->Controller, luptr->Name,  Emark );
    }
}

/*----- UnitChar.c ----- Print_UnitC_comma f*/
/*----- DocMethod */
extern void Print_UnitC_comma( FILE *out,
                             struct Unit_Characteristics *luptr, char *Emark )
/*-----EndDocHead---*/
{

    fprintf(out, "%s,%ld,%ld,%ld,%ld,%ld,%ld,%ld,%g,%g,%g,%g,%ld,%ld,%s",
        luptr->Name,
        luptr->Id,
        luptr->Activity,
        luptr->Force, /* is not in input file */
        luptr->Designation,
        luptr->Subordinates,
        luptr->Equipment,
        luptr->DataSize,
        luptr->ReportRate,
        luptr->OrderRate,
        luptr->FireRate,
        luptr->SenseRate ,
        luptr->FedNode,
        luptr->Echeleon,  Emark );
}
/*----- */
/*----- DocHeading */

```

30 June 1999

/* file: UnitEcheleon.c */

```

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <math.h>
#include "soa_defs.h"
#include "soa_cnst.h"

int Horiz = 10, Vert = 10, VertK = 5;
int maxHoriz=0;
int minHoriz=0;
unsigned int XWidth ;
unsigned int XLength ;
unsigned short int xsubi[3];
unsigned int TotalEquip[9], TotalPerson[9];
unsigned int TotalR_Equip[9], TotalR_Person[9];
unsigned long Color = 28;
int level;
int Level ;
int UELcLcounter ;
int Kcounter = 0 ;
int KunitId = 0 ;
int Count_ActLevel = 0 ;
int UnitEquip[4] = { 0, 0, 0, 0};
int UnitPers[4] = { 0, 0, 0, 0};

/*----- UnitEcheleon.c ----- f*/

/* ----- UnitEcheleon.c ----- View_Refresh f*/
/*----- DocHeading */

```

30 June 1999

```

extern int CountSubrEquip( FILE *out,
struct Unit_List      *ULp)
/*-----EndDocHead---*/
{
struct Unit_List *cULp;
//int i ;
//char   tempstr[64];
int Total, KeepTotal;
//unsigned long C ;
Total = 0;  KeepTotal = 0 ;
cULp = ULp ;
if (  ULp != NULL ) {
do {
//printf("CountSubrEquip  %5d, %17s %1d\n", Total, cULp->UChrp->Name, cULp->UChrp->Echeleon
);

    if ( cULp->Subrdp != NULL ) {
        KeepTotal = Total ;
        Total = CountSubrEquip( out, cULp->Subrdp );
        Total += KeepTotal ;
    }
    else {
        Total += cULp->UChrp->Equipment ;
    }
    //printf("CountSubrEquip  %5d, %17s %1d\n", Total, cULp->UChrp->Name, cULp->UChrp->Echeleon
);

    cULp = cULp->nrep ;
} while (cULp != NULL ) ;
}
return(Total);
}

/*-----DocMethod */
extern void ResetEcheleon(int Iset)
/*-----EndDocHead---*/
{
    Kcounter = 0 ;
    KunitId  = Iset ;
    Count_ActLevel = 0 ;
}

/* ----- UnitEcheleon.c -----TallyEcheleon f*/
/*-----DocMethod */
extern void TallyEcheleon( struct Unit_List  *ULp) /* UnitEcheleon.c */
/*-----EndDocHead---*/
{
struct Unit_List *cULp ;
//int i ;
//char   tempstr[64];
//extern void   LalaPlace( int State, int X, int Y );
int Total, aLevel ;

Total = 0;
cULp = ULp ;
aLevel = cULp->UChrp->Echeleon;
do {

    if ( cULp->Subrdp != NULL ) {
        TallyEcheleon(cULp->Subrdp );
    }
    Total += cULp->UChrp->Equipment ;

    TotalEquip[aLevel] += cULp->UChrp->Equipment ;
    TotalPerson[aLevel] += cULp->UChrp->Personnel ;
}

```


30 June 1999

```

    TotalR_Equip[aLevel] += cULp->UChrp->Rolled_Equipment ;
    TotalR_Person[aLevel] += cULp->UChrp->Rolled_Personnel ;

    cULp = cULp->nrep ;
} while (cULp != NULL) ;

printf("%1d %5d, ", aLevel, Total );
if ( aLevel <= 4 ) { printf("\n" ); }
}

/*----- UnitEcheleon.c -----*/
/*----- DocMethod */
extern void TallyClearEch()
/*-----EndDocHead---*/
{
    int i;
    for (i=0; i<9; i++ ) {
        TotalEquip[i] = 0 ;
        TotalPerson[i] = 0 ;
        TotalR_Equip[i] = 0 ;
        TotalR_Person[i] = 0 ;
    }
}

/*----- UnitEcheleon.c -----*/
/*----- DocMethod */
extern void TallyPrintEch(FILE *out, char *label )
/*-----EndDocHead---*/
{
    int i; unsigned int T;
    fprintf(out,"%-20s,          Army,   Corp,   Div,   Brig,   Bat,   Co,   Plt, Squad,
Total,\n", label);
    fprintf(out,"%-20s, Equipment,", label); T = 0;
    for (i=0; i<8; i++ ) { fprintf(out,"%5d, ",TotalEquip[i] ) ; T+= TotalEquip[i]; }
    fprintf(out,"%6d,\n",T); TotalEquip[8] = T;
    fprintf(out,"%-20s, Personnel,", label); T = 0;
    for (i=0; i<8; i++ ) { fprintf(out,"%5d, ",TotalPerson[i] ) ; T+= TotalPerson[i]; }
    fprintf(out,"%6d,\n",T); TotalPerson[8] = T;
    fprintf(out,"%-20s, Rolled_Eq,", label); T = 0;
    for (i=0; i<8; i++ ) { fprintf(out,"%5d, ",TotalR_Equip[i] ) ; T+= TotalR_Equip[i]; }
    fprintf(out,"%6d,\n",T); TotalR_Equip[8] = T;
    fprintf(out,"%-20s, Rolled_Pn,", label); T = 0;
    for (i=0; i<8; i++ ) { fprintf(out,"%5d, ",TotalR_Person[i] ) ; T+= TotalR_Person[i]; }
    fprintf(out,"%6d,\n",T); TotalR_Person[8] = T;
}

/*----- UnitEcheleon.c -----*/
/*----- DocMethod */
extern int GetTotalEquipByLevel(int i)
/*-----EndDocHead---*/
{
    int j ;
    j = (int)TotalEquip[i] ;
    return(j);
}

/*----- UnitEcheleon.c -----*/
/*----- DocMethod */
extern unsigned int GetTotalEquip()
/*-----EndDocHead---*/
{
    return((int)TotalEquip[8]);
}

/*----- UnitEcheleon.c -----*/
/*----- DocMethod */

```

30 June 1999

```
extern int GetTotalPersonByLevel(int i)
```

```
/*-----EndDocHead---*/
{
    return(TotalPerson[i]);
}
```

```
/*----- UnitEcheleon.c ----- Print_EchSummary f*/
/*----- DocMethod */
```

```
extern void Print_EchSummary( FILE *out )
```

```
/*-----EndDocHead---*/
{
    int i ;
    fprintf(out, " Count of units %3d\n", Kcounter );
    fprintf(out, " Equipment: "); for(i=0; i<4; i++ ) { fprintf(out, " %4d, ", UnitEquip[i] ); }
    fprintf(out, "\n Personnel: "); for(i=0; i<4; i++ ) { fprintf(out, " %4d, ", UnitPers[i] ); }
    fprintf(out, "\n" );
}
```

```
/*----- UnitEcheleon.c ----- Print_Echeleon f*/
/*----- DocHeading */
```

30 June 1999

```

extern void Print_Echeleon( FILE *out, /* UnitEcheleon.c */
                           struct Unit_List *ULp)
/*-----EndDocHead---*/
{
    struct Unit_List *cULp ;
    int i ;
    extern void Print_UnitC_comma(FILE *out,
                                struct Unit_Characteristics *luptr, char *Emark);
    //void Print_CommAssoc();
    char tempstr[64];
    struct Unit_Region_List *lUnRegLst ;

    cULp = ULp ;
    strcpy(tempstr, ".." );
    do {
        if ( cULp->UChrp->RegOfUnit != NULL ) {
            fprintf(out, "%3d: ", cULp->UChrp->RegOfUnit->xtReg->Id );
            if ( cULp->UChrp->RegOfUnit->nxtRegOfUnit != NULL ) {
                lUnRegLst = cULp->UChrp->RegOfUnit->nxtRegOfUnit;
                do { fprintf( out, "%3d: ", lUnRegLst->xtReg->Id );
                    lUnRegLst = lUnRegLst->nxtRegOfUnit ;
                } while(lUnRegLst != NULL ) ;
            }
        }
        else { fprintf( out, " : " ); }
        fprintf(out, "|%2d|", cULp->UChrp->FedNode );
        for(i=0; i<Level; i++) { fprintf(out, "~"); }
        if ( cULp->UCmdp != NULL && UELcLcounter > 0 ) {
            if ( cULp->UCmdp->UChrp->Name != NULL && UELcLcounter > 0 ) {
                sprintf( tempstr, " : Cmdr:%s:%1d: ", cULp->UCmdp->UChrp->Name, Level );
            }
        }
        else { strcpy(tempstr, " :: \n" ); }

        Print_UnitC_comma( out, cULp->UChrp, tempstr );
        //for(i=0; i<Level; i++) { fprintf(out, "~"); }
        //Print_CommAssoc(out, cULp->UChrp->CmNetp, "\n");
        UELcLcounter += 1;

        if ( cULp->Subrdp != NULL ) {
            Level += 1;
            Print_Echeleon(out, cULp->Subrdp );
            Level -= 1 ;
        }
        cULp = cULp->nrep ;
    } while (cULp != NULL ) ;

    /* for(i=0; i<Level; i++) { fprintf( stderr, " "); }
       fprintf( stderr, " return \n" ) ; */
}
/*----- UnitEcheleon.c ----- Print_Echeleon f*/
/*----- DocHeading */

```

30 June 1999

```

extern void PrintRTIEchelon( FILE *out,
                             struct Unit_List *ULp)
/*-----EndDocHead-----*/
{
    struct Unit_List *cULp ;
    int i ;
    extern void Print_UnitC_comma( FILE *out,
                                   struct Unit_Characteristics *luptr, char *Emark);
    //void Print_CommAssoc();
    char tempstr[64];
    struct Unit_Region_List *lUnRegLst ;

    cULp = ULp ;
    strcpy(tempstr, ".." );
    do {

        fprintf(out, "RTI{%5du:%4dd:%5di|%2d|",
                cULp->UChrp->Updated, cULp->UChrp->Discovered, cULp->UChrp->Interaction,
                cULp->UChrp->FedNode );
        for(i=0; i<Level; i++) { fprintf(out, "~"); }
        strcpy(tempstr, " :: \n" );

        Print_UnitC_comma( out, cULp->UChrp, tempstr );
        //for(i=0; i<Level; i++) { fprintf(out, "~"); }
        //Print_CommAssoc(out, cULp->UChrp->CmNetp, "\n");
        UELcLcounter += 1;

        if ( cULp->Subrdp != NULL ) {
            Level += 1;
            PrintRTIEchelon(out, cULp->Subrdp );
            Level -= 1 ;
        }
        cULp = cULp->nrep ;
    } while (cULp != NULL ) ;

    /* for(i=0; i<Level; i++) { fprintf( stderr, " "); }
       fprintf( stderr, " return \n" ) ; */
}
/*----- UnitEcheleon.c ----- Print_Echeleon f*/
/*----- DocHeading */

```

```

extern void PrintRTIInstanceEchelon( FILE *out,
                                   struct Unit_List *ULp)
/*-----EndDocHead---*/
{
    struct Unit_List *cULp ;
    int i ;
    extern void Print_UnitC_comma(FILE *out,
                                struct Unit_Characteristics *luptr, char *Emark);
    //void Print_CommAssoc();
    char tempstr[64];
    struct Unit_Region_List *lUnRegLst ;

    cULp = ULp ;
    strcpy(tempstr, ".." );
    do {

        fprintf(out, "RTI:" );
        for (i=0; i<ObjectsInSOM; i++) { fprintf(out, "%5d;", cULp->UChrp->ObjectInstance[i] ); }
        fprintf(out, "%5du:%4dd:%5di|%2d|",
            cULp->UChrp->Updated, cULp->UChrp->Discovered, cULp->UChrp->Interaction,
            cULp->UChrp->FedNode );
        for(i=0; i<Level; i++) { fprintf(out, "~"); }
        strcpy(tempstr, " :: \n" );

        Print_UnitC_comma( out, cULp->UChrp, tempstr );
        //for(i=0; i<Level; i++) { fprintf(out, "~"); }
        //Print_CommAssoc(out, cULp->UChrp->CmNetp, "\n");
        UELcLcounter += 1;

        if ( cULp->Subrdp != NULL ) {
            Level += 1;
            PrintRTIInstanceEchelon(out, cULp->Subrdp );
            Level -= 1 ;
        }
        cULp = cULp->nrep ;
    } while (cULp != NULL ) ;

    /* for(i=0; i<Level; i++) { fprintf( stderr, " "); }
       fprintf( stderr, " return \n" ) ; */
}
/*----- UnitEcheleon.c ----- Print_CommAssoc f*/
extern void Print_CommAssoc( FILE *out,
                            struct Comm_Net_Association *CmNetp,
                            char *Emark )
/*-----EndDocHead---*/
{
    char tempstr[64];
    //void Print_Order( FILE *out, struct Order_Packet *p, char*Emark);

    if ( CmNetp->UChrp != NULL ) {
        fprintf( out, "%s", CmNetp->UChrp->Name );
    }
    fprintf( out, "%g,%g,%g,%g,%g,%1d,To %1d,",
        CmNetp->MaxSize,
        CmNetp->MaxTime,
        CmNetp->ReplyToCommandAt,
        CmNetp->ReplyToPeerAt,
        CmNetp->SubModeChangeAt,
        CmNetp->SubModeOfActivity,
        CmNetp->SpecificUnit );
    fprintf( out, "%s", Emark );
    if ( CmNetp->CmdOrdrp != NULL ) { fprintf( out, " " );
        strcpy( tempstr, " Cmd," ); strcat( tempstr, Emark);
        fprintf(out,
            "Need to add ReadOrdr.c for Print_Order(FILE *out, Order_Packet *p, char*Emark);\n");
        // Print_Order( out, CmNetp->CmdOrdrp, tempstr );
    }
}

```

30 June 1999

```
    }
    if ( CmNetp->PeerOrdrp != NULL ) { fprintf( out, "          " );
        strcpy( tempstr, " Per," ); strcat( tempstr, Emark);
        fprintf(out,
"Need to add ReadOrdr.c for Print_Order(FILE *out, Order_Packet *p,char*Emark);\n");
//      Print_Order( out, CmNetp->PeerOrdrp, tempstr );
    }

}

/* ----- UnitEcheleon.c -----      ViewNext Xwin f*/
/*-----                               DocHeading */
```

```

extern void ViewNext()
/*-----EndDocHead---*/

{
    Vert = 8;
    Color = 140 ;

}
/* ----- UnitEcheleon.c ----- ViewNew Xwin f*/
/*----- DocMethod */
extern void ViewNew() /* UnitEcheleon.c */
/*-----EndDocHead---*/
{
    unsigned int GetXWidth();
    unsigned int GetXLenght();

    Level = 0 ;
    Vert = 10 ;
    Horiz = 10 ;
    Color = 28 ;
    maxHoriz = 0;
    XWidth = GetXWidth() - 17;
    minHoriz = XWidth ;
    XLenght = GetXLenght() - 10;
}
/* ----- UnitEcheleon.c ----- View_Echelon f*/
/*----- DocMethod */
extern void ViewEcheleonLeft( struct Unit_List *ULp)
/*-----EndDocHead---*/
{
    struct Unit_List *cULp ;
    //int i ;
    //char tempstr[64];
    extern void LalaPlace(int State, int X, int Y );

    cULp = ULp ;
    if (Horiz > maxHoriz ) maxHoriz = Horiz ;
    do {

        LalaPlace( cULp->UChrp->ViewColor, Horiz, Vert );
        cULp->UChrp->ViewHoriz = Horiz;
        cULp->UChrp->ViewVert = Vert ;

        //printf(" %5d, %5d, %5d \n", Color, Horiz, Vert);
        if ( cULp->Subrdp != NULL ) {
            Level += 1;
            Horiz += 10;
            ViewEcheleonLeft(cULp->Subrdp );
            Level -= 1 ;
            if (Vert < XLenght ) { Horiz -= 10 ; }
            if ( Vert > XLenght ) { Vert = 5 ; Horiz = maxHoriz + 10*Level ;
                                maxHoriz += 10*Level; }
        }
        Vert += VertK ;

        cULp = cULp->nrep ;

    } while (cULp != NULL ) ;

}

/* ----- UnitEcheleon.c ----- View_Echelon f*/
/*----- DocMethod */
extern void ViewEcheleonRight( struct Unit_List *ULp)
/*-----EndDocHead---*/
{

```

```
struct Unit_List *cULp ;
//int i ;
//char tempstr[64];
extern void LalaPlace( int State, int X, int Y );
```

```
cULp = ULp ;
if (Horiz == 10 ) Horiz = minHoriz - 5 ;
do {

    LalaPlace( cULp->UChrp->ViewColor, Horiz, Vert );
    cULp->UChrp->ViewHoriz = Horiz;
    cULp->UChrp->ViewVert = Vert ;

    if ( cULp->Subrdp != NULL ) {
        Level += 1;
        Horiz -= 10;
        ViewEcheleonRight(cULp->Subrdp );
        Level -= 1 ;
        if (Vert < XLength ) { Horiz += 10 ; }
        if ( Vert > XLength ) { Vert = 8;
                                Horiz = Horiz - 10*Level;
                                minHoriz -= 10*Level; }
        if ( Horiz < 10 ) { Horiz = 50 ; }
    }
    Vert += VertK ;

    cULp = cULp->nrep ;

} while (cULp != NULL ) ;

}
```

```
/* ----- UnitEcheleon.c ----- View_Refresh f*/
/*----- DocMethod */
extern void ViewRefresh( struct Unit_List *ULp)
/*-----EndDocHead---*/
{
    struct Unit_List *cULp ;
    //int i ;
    //char tempstr[64];
    extern void LalaPlace(int State, int X, int Y );
    //unsigned long C ;

    cULp = ULp ;

    do {

        LalaPlace( cULp->UChrp->ViewColor, cULp->UChrp->ViewHoriz, cULp->UChrp->ViewVert );

        /*printf("%5d %-21s %5d %4d %3d\n", cULp->UChrp->Id, cULp->UChrp->Name,
                cULp->UChrp->ViewHoriz, cULp->UChrp->ViewVert, cULp->UChrp->Echeleon );
        */
        if ( cULp->Subrdp != NULL ) {
            ViewRefresh(cULp->Subrdp );
        }
        cULp = cULp->nrep ;

    } while (cULp != NULL ) ;

}
```



```

/*****
--*  int csv( char *lstr, char *pieces[], char *delimiter )  parses up a string by some single
delimiter and allocates pieces of memory *
--*  *
*****/

#include <stdlib.h>
#include <errno.h>
#include <stdio.h>
#include <string.h>

int csv( char *lstr, char *pieces[], char *delimiter )
{
    char *bptr,*eptr, *dptr ;
    int i;

    bptr = eptr = lstr ;
    /* printf("%s %s \n", bptr, delimiter ); */

    i = 0 ;
    while( bptr != NULL && eptr != NULL ) {
        eptr = strstr( bptr, delimiter );
        if ( eptr != NULL || eptr == NULL && bptr < lstr + strlen(lstr) ) {
            if ( eptr == NULL && bptr < lstr + strlen(lstr) ) { eptr = lstr + strlen(lstr) ; }
            pieces[i] = (char *)malloc((eptr-bptr)+2);

            if ( errno == EINVAL ) {
                printf(" csv: (EINVAL) Indicates a call has requested 0 bytes. \n" );
                printf(" csv: @ %s \n", lstr );
                return(-1) ;}
            if ( errno == ENOMEM ) {
                printf(" csv: (ENOMEM) Indicates that not enough storage space was available. \n" );
                printf(" csv: @ %s \n", lstr );
                return(-1) ; }

            strncpy( pieces[i], bptr, eptr-bptr) ;
            *(pieces[i]+(eptr-bptr)) = '\0' ;
            /* printf(" %s ", pieces[i] ); */
            i += 1 ;
            bptr = eptr + 1;
        }
    }
    /* printf("\n"); */
    return(i);
} /* end of csv parse a line given a delimiter */

```

```

/* ----- DocHeading---*/

```

```
/*    file: data_distrib_mgmt_svc.c    */

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>

#include "serv_crit.h"
#include "event.h"
#include "rti.h"
#include "rtimgr.h"
#include "rti_services.h"

#define  DEBUG_DDM_MGMT  0

/* ----- DocComment---*/
/*  ddm_create_update_region:
*/
/* ----- DocHeading---*/
```

```
extern double ddm_create_update_region(int federate_nbr, int region_nbr)
/* ----- EndDocHead---*/
{
    double          svc_statistic=0.0;

    #if DEBUG_DDM_MGMT
    printf("ddm_create_update_region- region nbr:%d \n", region_nbr);
    #endif

    if (region_nbr < MAX_NBR_FEDRTN_REGIONS)
    {
        FedExdb.fedrtn_regions[region_nbr].subscribed_objects = NULL;
        FedExdb.fedrtn_regions[region_nbr].subscribed_interactions = NULL;
        FedExdb.nbr_fedrtn_regions++;
    }
    else
    {
        #if DEBUG_DDM_MGMT
        printf("region nbr exceeds max regions \n");
        #endif
        ;
    }

    #if DEBUG_DDM_MGMT
    printf("fedrtn now has: %d regions \n", FedExdb.nbr_fedrtn_regions);
    #endif

    svc_statistic =
        rtimgr_get_RTI_ambsvc_time(federate_nbr, RTI_CREATE_UPDATE_REGION,
                                   1);

    return(svc_statistic);
}

/* ----- DocComment---*/
/* ddm_associate_update_region:
*/
/* ----- DocHeading---*/
```

30 June 1999

```
extern double ddm_associate_update_region(int federate_nbr)
/* ----- EndDocHead---*/
{
    double      svc_statistic=0.0;

    svc_statistic =
        rtimgr_get_RTI_ambsvc_time(federate_nbr, RTI_ASSOC_REGION, 1);

    return(svc_statistic);
}

/* ----- DocHeading---*/
```

30 June 1999

```
/* file: declar_mgmt_svc.c */
/* This file contains the RTI Ambassador Services (or action methods) */
/* for the Declaration Mgmt services */

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>

#include "serv_crit.h"
#include "event.h"
#include "rti.h"
#include "rtimgr.h"
#include "rti_services.h"

#define    DEBUG_DECL_MGMT    0

/* ----- DocComment---*/
/* dm_publish_objclass:
 *   check FOMdb for valid obj_class
 *   if not present, error
 *   else valid class to publish via add class to Fedex_db
 *   future: validate and setup attribs
 */
/* ----- DocHeading---*/
```

30 June 1999

```

extern double dm_publish_objclass(int federate_nbr, int class_nbr)
/* ----- EndDocHead----- */
{
    double      svc_statistic=0.0;

    #if DEBUG_DECL_MGMT
    printf("dm_publish_objclass-federate:%d class:%d\n",
    federate_nbr, class_nbr);
    #endif

    /* if obj class was setup as part of FOM */
    if (class_nbr > FOMdb.nbr_fedrtn_objclasses)
    {
        printf("dm_publish: ERROR-unable to publish class since not a valid FOM class \n",
        class_nbr);
        return(0);
    }
    /* future: also test for attribute is known for this class */

    /* if class not published already */
    if (!(FedExdb.fedrtn_objclasses[class_nbr].published))
    {
        /* activate class as published */
        FedExdb.fedrtn_objclasses[class_nbr].published = TRUE;
    #if DEBUG_DECL_MGMT
        printf("dm_publish_objclass- obj class:%d newly published \n",
        class_nbr);
    #endif
        FedExdb.fedrtn_objclasses[class_nbr].owner_federate = federate_nbr;
    }
    else
    {
    #if DEBUG_DECL_MGMT
        printf("dm_publish_objclass- obj class:%d published again\n",
        class_nbr);
    #endif
    }
    ;

    svc_statistic = rtimgr_get_RTI_ambsvc_time(federate_nbr, RTI_PUBLISH_OBJCLSS, 1);

    return(svc_statistic);
}

/* ----- DocComment----- */
/* dm_publish_interact_class:
 * check FOMdb for valid interact_class
 * if not present, error
 * else valid class to publish via add class to Fedex_db
 * future: validate and setup attribs
 */
/* ----- DocHeading----- */

```

```
extern double dm_publish_interact_class(int federate_nbr,
                                       int class_nbr)
/* ----- EndDocHead---*/
{
    double      svc_statistic=0.0;

    #if DEBUG_DECL_MGMT
    printf("dm_publish_interact_class-federate:%d class:%d\n",
          federate_nbr, class_nbr);
    #endif

    /* if obj class was setup as part of FOM */
    if (class_nbr > FOMdb.nbr_fedrtn_interact_classes)
    {
    #if DEBUG_DECL_MGMT
        printf("dm_publish: ERROR-unable to publish class since not a valid FOM class \n",
              class_nbr);
    #endif
        return(0);
    }

    /* future: also test for attribute is known for this class */

    /* if class not published already */
    if (!(FedExdb.fedrtn_interact_classes[class_nbr].published))
    {
        /* activate class as published */
        FedExdb.fedrtn_interact_classes[class_nbr].published = TRUE;
        FedExdb.fedrtn_interact_classes[class_nbr].owner_federate = federate_nbr;
    }
    else
    {
    #if DEBUG_DECL_MGMT
        printf("dm_publish_interact_class- interact class:%d already published \n",
              class_nbr);
    #endif
    }
    ;

    /* nbr nodes affected */
    /* assume only this federate for now */
    svc_statistic = rtimgr_get_RTI_ambsvc_time(federate_nbr, RTI_PUBLISH_INTCLSS, 1);

    return(svc_statistic);
}
/* ----- DocComment---*/
/* create_subscribed_node:
*/
/* ----- DocHeading---*/
```

30 June 1999

```
static SUBSCRIBED_INFO_TYPE *create_subscribed_node(int class_nbr,
                                                    int federate_nbr)
/* ----- EndDocHead---*/
{
    SUBSCRIBED_INFO_TYPE    *node;

    node = (SUBSCRIBED_INFO_TYPE *) calloc(1, sizeof(SUBSCRIBED_INFO_TYPE));
    node->class_nbr = class_nbr;
    node->federate_name = federate_nbr;
    node->next = NULL;

    return(node);
}

/* ----- DocComment---*/
/* add_federate_to_regions_table:
*/
/* ----- DocHeading---*/
```


30 June 1999

```

void add_federate_to_regions_table(int class_nbr,
                                   int federate_nbr,
                                   int region_nbr,
                                   int type)
/* ----- EndDocHead---*/

{
    SUBSCRIBED_INFO_TYPE *node, *list;

    node = create_subscribed_node(class_nbr, federate_nbr);
    if (type == OBJECT_TYPE)
        list = FedExdb.fedrtn_regions[region_nbr].subscribed_objects;
    else if (type == INTERACTION_TYPE)
        list = FedExdb.fedrtn_regions[region_nbr].subscribed_interactions;
    else
        printf("add_federate_to_regions_table:error-invalid type:%d \n", type);

#ifdef DEBUG_DECL_MGMT
    printf("in add_federate_to_regions_table-type:%d node:%x list:%x \n",
          type, node, list);
#endif
    if (list != NULL)
        node->next = list;
    list = node;

#ifdef DEBUG_DECL_MGMT
    printf(" after added node- node:%x list:%x list nxt:%x\n",
          node, list, list->next);
#endif

    if (type == OBJECT_TYPE)
        FedExdb.fedrtn_regions[region_nbr].subscribed_objects = list;
    else
        FedExdb.fedrtn_regions[region_nbr].subscribed_interactions = list;
}

/* ----- DocComment---*/
/* dm_subscribe_objclass:
 * Add a class to the subscribed list
 * also for subscribe with region
 *
 * objclass => object attribute group
 * future: handle attributes too
 */
/* ----- DocHeading---*/

```

30 June 1999

```

extern double dm_subscribe_objclass(int obj_class,
                                   int federate_name,
                                   int region_nbr)
/* ----- EndDocHead---*/

{
    double      svc_statistic=0.0;

#ifdef DEBUG_DECL_MGMT
    printf("dm_subscribe_objclass- federate_name:%d class:%d region: %d\n",
          federate_name, obj_class, region_nbr);
#endif

/*
    if (obj_attrib)
        set_obj_attrib
*/

/* if obj class was setup as part of FOM */
if (obj_class < FOMdb.nbr_fedrtn_objclasses)
{
    /* if class not already subscribed */
    if (!FedExdb.fedrtn_objclasses[obj_class].nbr_subscribed_federates)
    {
        /* activate the class */
        FedExdb.federates[federate_name].receives_updates = TRUE;
        /* also send reaction event to owner federate
           * i.e., send Turn Updates On advisory
           */
    }
    FedExdb.fedrtn_objclasses[obj_class].nbr_subscribed_federates++;
    add_federate_to_regions_table(obj_class,
                                  federate_name,
                                  region_nbr,
                                  OBJECT_TYPE);
}
else
{
#ifdef DEBUG_DECL_MGMT
    printf("dm_subscribe_obj_class: ERROR- obj_class-%d is not in FOM info \n",
          obj_class);
#endif
    return(-1);
}

    svc_statistic = rtimgr_get_RTI_ambsvc_time(federate_name, RTI_SUBSCRIBE_OBJCLSS, 1);

    return(svc_statistic);
}

/* ----- DocComment---*/
/* dm_subscribe_interact_class:
*/
/* ----- DocHeading---*/

```

30 June 1999

```

extern double dm_subscribe_interact_class(int federate_name,
                                         int class_nbr,
                                         int region_nbr)

/* ----- EndDocHead----- */
{
    double   svc_statistic=0.0;

#ifdef DEBUG_DECL_MGMT
    printf("dm_subscribe_interact_class federate_name:%d class:%d region: %d\n",
           federate_name, class_nbr, region_nbr);
#endif

    /*
       if (interact_params)
           set interact_params
    */

    /* if class was setup as part of FOM */
    if (class_nbr < FOMdb.nbr_fedrtn_interact_classes)
    {
        /* if class not already subscribed */
        if (!FedExdb.fedrtn_interact_classes[class_nbr].nbr_subscribed_federates)
        {
            /* activate the class */
            FedExdb.federates[federate_name].receives_updates = TRUE;
            /* also send reaction event to owner federate
               * i.e., send Turn Interactions On advisory
            */
        }
        FedExdb.fedrtn_interact_classes[class_nbr].nbr_subscribed_federates++;
        add_federate_to_regions_table(class_nbr,
                                     federate_name,
                                     region_nbr,
                                     INTERACTION_TYPE);
    }
    else
    {
#ifdef DEBUG_DECL_MGMT
        printf("dm_subscribe_interact_class: ERROR- class:%d is not in FOM info \n",
               class_nbr);
#endif
        return(-1);
    }

    svc_statistic = rtimgr_get_RTI_ambsvc_time(federate_name, RTI_SUBSCRIBE_INTCLSS, 1);

    return(svc_statistic);
}

/* ----- DocHeading----- */

```

/* file: event_manager.c */

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

```
#include "event.h"
#include "serv_crit.h"
#include "rti.h"
#include "proto.h"
#include "rtimgr.h"
```

/* ----- DocHeading ---*/

30 June 1999

```

extern void eventmgr_process_event(EVENT_MESSAGE_TYPE *event_msg)
/* ----- EndDocHead---*/
{
/* ----- DocComment---*/
/* eventmgr_process_event:
*   get event off RTI queue, and pass to next processor
*/

    if (event_msg->WhoGetsIt.RTI == 0 && event_msg->WhoGetsIt.SIM ==0)
        /* done=0 - destroy the event msg */
        ;
    else if (event_msg->WhoGetsIt.RTI == 0 && event_msg->WhoGetsIt.SIM ==1)
        /* sim=1 - pass control to sim model */
        ;
    else if (event_msg->WhoGetsIt.RTI == 1 && event_msg->WhoGetsIt.SIM ==0)
        /* rti=2 - pss control to rti model */
        ;
    else
        printf("eventmgr_process_event- invalid value of type:%di %d\n",
            event_msg->WhoGetsIt.RTI,
            event_msg->WhoGetsIt.SIM );
}

/* ----- DocComment---*/
/* eventmgr_change_processing_mode:
*/
/* ----- DocHeading---*/

```

```
extern void eventmgr_change_processing_mode(EVENT_MESSAGE_TYPE *event_msg,
                                           int new_mode)
/* ----- EndDocHead---*/
{
    switch(new_mode)
    {
        case (DONE_PROCESSING):
            event_msg->WhoGetsIt.RTI = 0;
            event_msg->WhoGetsIt.SIM = 0;
            break;
        case (SIM_PROCESSING):
            event_msg->WhoGetsIt.RTI = 0;
            event_msg->WhoGetsIt.SIM = 1;
            break;
        case (RTI_PROCESSING):
            event_msg->WhoGetsIt.SIM = 0;
            event_msg->WhoGetsIt.RTI = 1;
            break;
        default:
            printf("eventmgr_change_processing_mode- invalid value of new_mode:%d\n",
                  new_mode);
    }
}

/* ----- DocHeading---*/
```

30 June 1999

```
static double  eventmgr_determine_events_LBTS(int federate_nbr)
/* ----- EndDocHead---*/
{
/* ----- DocComment---*/
/* eventmgr_eventmgr_determine_events_LBTS
 *   LBTS : the smallest time stamp of any unprocessed event within
 *   a federate at its cut point plus the federate's lookahead time
 */

    double  new_LBTS;

    /* check the TSO event queue for the event with a minimum time stamp */
    // new_LBTS = 130000; /* ??? hardwired for now, change to a correct value */
new_LBTS = GetLBTSfromFederate( federate_nbr );
    /* add the lookahead value to it, and return the new min LBTS value */
    new_LBTS+= FedExdb.fedrtn_time_mgmt_info.global_lookahead_value;

    return(new_LBTS);
}

/* ----- DocHeading---*/
```

30 June 1999

```

extern void    eventmgr_retrieve_LBTS_info(int  federate_nbr,
                                           int    *federate_rcvd_msgs,
                                           int    *federate_sent_msgs,
                                           double  *federate_min_LBTS,
                                           int    *all_nodes_reported)

/* ----- EndDocHead---*/
{
/* ----- DocComment---*/
/* eventmgr_retrieve_LBTS_info:
 *   return LBTS info containing:  the LBTS of unprocessed events on TSO queue
 *   and counts of current rcvd and sent msgs
 */

    FEDERATE_INFO_TYPE    *federate_info;
    int                    i;
    int    SentByFed, RcvdByFed ;

    *all_nodes_reported = FALSE;

printf("eventmgr_retrieve_LBTS_info- federate:%d \n", federate_nbr);

    federate_info = &FedExdb.federates[federate_nbr];
    *federate_min_LBTS = eventmgr_determine_events_LBTS(federate_nbr);

    ChangeFederateColorTag(federate_nbr, &SentByFed, &RcvdByFed );
    /* report values */
    *federate_rcvd_msgs = RcvdByFed;
    *federate_sent_msgs = SentByFed;

    *all_nodes_reported = TRUE;

}

/* ----- DocComment---*/
/* eventmgr_get_parent_destination
 */
/* ----- DocHeading---*/

```



```
extern int    eventmgr_get_parent_name(int federate_nbr)
/* ----- EndDocHead---*/
{
    int        parent=0;

    if (federate_nbr == FOMdb.LBTS_controller) /* This must be the controller */
        return(FOMdb.LBTS_controller);

    parent = FOMdb.nodes[federate_nbr].parent;

    return(parent);
}

/*----- DocComment---*/
/* eventmgr_get_destination
*/
/* ----- DocHeading---*/
```

30 June 1999

```

extern int    eventmgr_get_destination(int federate_nbr,
                                       EVENT_MESSAGE_TYPE  *event_msg_info_ptr)
{
    FEDERATE_DESTINS_TYPE  *dest_element;
    int    dest_federate;

    /* if the federate is a parent and the federate hasn't
       sent initial counts to parent yet, send report to same federate */
    if (rtmgr_federate_is_parent(federate_nbr) &&
        (!rtmgr_federate_processed_initial_counts(federate_nbr)))
        dest_federate = federate_nbr;
    /* otherwise, get the destination federate nbr for this report */
    else
        dest_federate = eventmgr_get_parent_name(federate_nbr);

    printf("eventmgr_get_destination- dest_federate:%d \n",
           dest_federate);

    dest_element = om_create_destinations_element(dest_federate);
    if (dest_element)
    {
        event_msg_info_ptr->destinations_list = dest_element;
    }
    else
        printf("eventmgr_get_destination: unable to create list\n");

    return(dest_federate);
}

```

```

/* ----- DocHeading ----*/

```

30 June 1999

/* file: fedrtn_mgmt_svc.c

```
* This file contains the RTI Ambassador Services (or action methods)
* for the Federation Mgmt services
*/
```

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <math.h>
```

```
#include "serv_crit.h"
#include "event.h"
#include "rti.h"
#include "rtimgr.h"
#include "rti_services.h"
#include "proto.h"
```

```
static double  fm_setup_initiate_federate_save_events(int  active_federates,
                                                         EVENT_MESSAGE_TYPE  *event_msg_info_ptr);
```

```
/* ----- DocComment ---*/
/* fm_create_fedrtn_execution:
*      add new name to fedrtn_db of names
*/
/* ----- DocHeading ---*/
```

30 June 1999

```

extern double fm_create_fedrtn_execution(int federate_nbr,
                                         int    fedrtn_name)
/* ----- EndDocHead----- */

{
    double    svc_statistic=0.0;

    printf("fm_create_fedrtn_execution-federation name:%d federate:%d\n",
           fedrtn_name,fedrtn_name);
    if (!FedExdb.fedrtn_name)
    {
        FedExdb.fedrtn_name = fedrtn_name;          /* fedrtn is active */
        FedExdb.fedex_state_status.FedEx_name_exists = TRUE;

        FedExdb.fedrtn_time_mgmt_info.LBTS_current =
            CurrentFederateTime(federate_nbr);
        printf("LBTS time set to:%d \n",
               FedExdb.fedrtn_time_mgmt_info.LBTS_current);
    }
    else
        printf("fm_create_fedrtn_execution: fedrtn_name-%d is already active \n",
               fedrtn_name);

    /* only 1 federate is affected by this action */
    svc_statistic =
        rtimgr_get_RTI_ambsvc_time(federate_nbr, RTI_CREATE_FEDEX,
                                   1);

    return(svc_statistic);
}
/* ----- DocComment----- */
/* fm_initialize_federate:
*/
/* ----- DocHeading----- */

```

30 June 1999

```
static void fm_initialize_federate(FEDERATE_INFO_TYPE *federate,
                                   int federate_name,
                                   int fedrtn_name)
/* ----- EndDocHead---*/
{
    federate->assoc_fedrtn_name = fedrtn_name;
    federate->federate_state = ACTIVE;
    federate->federate_rti_version = 2.0;
    federate->receives_updates = FALSE;
    federate->saved_status = FALSE;
    federate->active_interactions = NULL;
    federate->federate_LBTS = 0;
    federate->federate_marker_mode = 0; /* start mode at white=0 */
}

/* ----- DocComment---*/
/* fm_join_fedrtn_execution
*/
/* ----- DocHeading---*/
```

30 June 1999

```

extern double fm_join_fedrtn_execution(int federate_name,
                                       int fedrtn_name)
/* ----- EndDocHead-----*/

{
    double    svc_statistic=0.0;

    printf("fm_join_fedrtn_execution- federate:%d to federation:%d\n",
           federate_name, fedrtn_name);

    if (federate_name > FedExdb.nbr_member_federates)
        printf("federate:%d joining is out of sequence\n", federate_name);

    if (federate_name <= MAX_NBR_FEDERATES)
    {
        FedExdb.nbr_member_federates++;

        fm_initialize_federate(&FedExdb.federates[federate_name],
                               federate_name, fedrtn_name);

        /* query fedrtn for nbr federates currently active */
        printf("new nbr member federates: %d \n", FedExdb.nbr_member_federates);

        /* return only 1 federate so far
           for initiating join
        */
        svc_statistic =
            rtimgr_get_RTI_ambsvc_time(federate_name, RTI_JOIN_FEDEX,
                                       1);
    }
    else
        printf("fm_join_fedrtn_execution- unable to join/create new federate:%d in fedex \n",
               federate_name);

    return(svc_statistic);
}

/* ----- DocComment-----*/
/* fm_request_fedrtn_save
*/
/* ----- DocHeading-----*/

```

30 June 1999

```

extern double fm_request_fedrtn_save(int federate_nbr,
                                     int *active_federates,
                                     EVENT_MESSAGE_TYPE *event_msg_info_ptr)
/* ----- EndDocHead---*/

{
    double      svc_statistic=0.0;

    printf("fm_request_fedrtn_save\n");

    FedExdb.fedex_state_status.save_in_process = TRUE;

    /* each federate needs to perform a save */
    /* compute nbr federates currently active */
    *active_federates = FedExdb.nbr_member_federates;

    /* use fctn to acheive each federate performing a save */
    svc_statistic = fm_setup_initiate_federate_save_events(*active_federates,
                                                           event_msg_info_ptr);

    svc_statistic +=
        rtimgr_get_RTI_ambsvc_time(federate_nbr, RTI_RQST_FEDRTN_SAVE,
                                   *active_federates);

    return(svc_statistic);

}
/* ----- DocComment---*/
/* fm_initiate_federate_save:   Fed Amb Svc
*/
/* ----- DocHeading---*/

```

```
extern double  fm_initiate_federate_save()
{
```

```
    double      svc_statistic=0.0;
```

```
    svc_statistic =
        rtimgr_get_Fed_ambsvc_time(RTI_INITIATE_FED_SAVE,
                                    1);
```

```
    return(svc_statistic);
```

```
}
```

```
/* ----- DocComment---*/
/* fm_setup_initiate_federate_save_events
*/
/* ----- DocHeading---*/
```


30 June 1999

```

static double fm_setup_initiate_federate_save_events(int active_federates,
                                                    EVENT_MESSAGE_TYPE *event_msg_info_ptr)

/* ----- EndDocHead----- */
{
    double      svc_statistic=0.0;
    FEDERATE_INFO_TYPE *fed_info;
    int         i;
    FEDERATE_DESTINS_TYPE *destinations_list, *dest_element;

    printf("fm_setup_initiate_federate_save_events\n");

    destinations_list = event_msg_info_ptr->destinations_list;
    /* each federate needs to perform a save */
    /* create a save event for all federates currently active */
    for (i=0; i<active_federates; i++)
    {
        dest_element = om_create_destinations_element(i);
        if (dest_element)
        {
            printf("federate:%d adding new element:%x onto list:%x\n",
                  i, dest_element, destinations_list);
            if (i==0)
                dest_element->next =destinations_list;
            destinations_list = dest_element;
        }
    }
    if (active_federates)
    {
        event_msg_info_ptr->Rti.rti_svc_nbr = RTI_INITIATE_FED_SAVE;
        FedExdb.nbr_federates_saving = active_federates;
    }

    svc_statistic =
        rtingr_get_Fed_ambsvc_time(RTI_INITIATE_FED_SAVES_SETUP,
                                   active_federates);

    return(svc_statistic);
}
/* ----- DocComment----- */
/* add_federate_to_feds_saving_list:
*/
/* ----- DocHeading----- */

```

30 June 1999

```
static void add_federate_to_feds_saving_list(int federate_nbr)
/* ----- EndDocHead---*/
{

}

/* ----- DocComment---*/
/* fm_federate_save_begun
*/
/* ----- DocHeading---*/
```

30 June 1999

```
extern double fm_federate_save_begun(int federate_nbr,
                                     int *nbr_federates)
/* ----- EndDocHead----- */
{
    double    svc_statistic = 0.0;

    /* this federate has begun saving, flag it */
    FedExdb.federates[federate_nbr].saved_status= SAVING;

    add_federate_to_feds_saving_list(federate_nbr);

    *nbr_federates = FedExdb.nbr_member_federates;
    svc_statistic =
        rtimgr_get_RTI_ambsvc_time(federate_nbr, RTI_FED_SAVE_BEGUN,
                                   *nbr_federates);

    return(svc_statistic);
}

/* ----- DocComment----- */
/*    fm_federate_save_achieved
*/
/* ----- DocHeading----- */
```

30 June 1999

```
extern double fm_federate_save_achieved(int federate_nbr)
/* ----- EndDocHead---*/
{
    double    svc_statistic=0.0;

    /* set federate's save status to complete */
    FedExdb.federates[federate_nbr].saved_status = SAVE_COMPLETE;

    svc_statistic =
        rtimgr_get_RTI_ambsvc_time(federate_nbr, RTI_FED_SAVE_COMPLETE, 1);

    return(svc_statistic);
}
/* ----- DocComment---*/
/* is_federate_in_fed_saved_list
*/
/* ----- DocHeading---*/
```

30 June 1999

```
static int is_federate_in_fed_saved_list(int federate_nbr)
{
    return(TRUE);
}
```

```
/* ----- DocComment---*/
/*  fm_is_fedrtn_saved
*/
/* ----- DocHeading---*/
```

```
extern int fm_is_fedrtn_saved()
/* ----- EndDocHead---*/
{
    int completed= FALSE;
    int nbr_federates_saving =0;
    int i;

    /* loop thru all federates */
    for (i=0; i<FedExdb.nbr_member_federates; i++)
    {
        /* test if federate is part of federates saving list */
        if (is_federate_in_fed_saved_list(i))
        {
            /* test for completed with save */
            if (FedExdb.federates[i].saved_status == SAVE_COMPLETE)
                nbr_federates_saving++;
        }
    }
    /* test for all completed */
    if (nbr_federates_saving == FedExdb.nbr_federates_saving)
        completed = TRUE;
    else
        completed = FALSE;

    return(completed);
}

/* ----- DocComment---*/
/* fm_fedrtn_save_achieved
*/
/* ----- DocHeading---*/
```

30 June 1999

```
extern double  fm_fedrtn_save_achieved()
/* ----- EndDocHead---*/

{
    double      svc_statistic=0.0;

    svc_statistic =
        rtimgr_get_Fed_ambsvc_time(RTI_FEDRTN_SAVED,
                                   1);

    return(svc_statistic);
}

/* ----- DocComment---*/
/*  fm_setup_fedrtn_complete_events
*/
/* ----- DocHeading---*/
```

30 June 1999

```

extern double  fm_setup_fedrtn_complete_events(int federate_nbr,
                                              int *saving_feds,
                                              EVENT_MESSAGE_TYPE *event_msg_info_ptr)
/* ----- EndDocHead-----*/
{
    FEDERATE_DESTINS_TYPE *dest_element, *destinations_list;
    double    svc_statistic=0.0;
    int      i;

    printf("fm_setup_fedrtn_complete_events\n ");
    destinations_list = event_msg_info_ptr->destinations_list;
    /* loop thru all federates */
    for (i=0; i<FedExdb.nbr_member_federates; i++)
    {
        /* test if federate is part of federates saving list */
        if (is_federate_in_fed_saved_list(i))
        {
            /* create an event to the queue */
            dest_element = om_create_destinations_element(i);
            if (dest_element)
            {
                printf("federate:%d adding new element:%x onto list:%x\n",
                       i, dest_element, destinations_list);
                if (*saving_feds==0)
                    dest_element->next = destinations_list;
                destinations_list = dest_element;
                (*saving_feds)++;
            }
        }
    }

    if (*saving_feds)
        event_msg_info_ptr->Rti.rti_svc_nbr = RTI_FEDRTN_SAVED;

    svc_statistic =
        rtimgr_get_RTI_ambsvc_time(federate_nbr, RTI_FED_SAVE_COMPLETE,
                                   1);

    return(svc_statistic);
}
/* ----- DocHeading -----*/

```



```
/* file: init_rti.c */
```

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <strings.h>
#include <errno.h>
#include <math.h>
#include "rti_services.h"
#include "serv_crit.h"
#include "event.h"
#include "rti.h"
#include "rtimgr.h"
```

```
/* globally available variables */
RID_INFO_TYPE      RIDdb;
FOM_INFO_TYPE      FOMdb;
FEDEX_INFO_TYPE    FedExdb;
```

```
extern int      service_criteria_lines;
```

```
/* ----- DocComment ---*/
/*  RIDdb_init:
*/
/* ----- DocHeading ---*/
```

```
void RIDdb_init()
/* ----- EndDocHead---*/

{

/* initialize info on the I/O, hardware info
 * and other configuration related items
 */

/* hardwire values for now
 * later, read in values from rdr file
 */
RIDdb.distrib_fedex = DISTRIBUTED;
RIDdb.network_type = ETHERNET;

}

/* ----- DocComment---*/
/* program cpm */
/* ----- DocHeading---*/
```

```
extern void Initialize_RTI()
/* ----- EndDocHead---*/
{
    int            i ;
    FILE           *outfile;
    RTI_SERVICE_TBL_ENTRY_TYPE  *rtisvc_tbl_ptr=NULL;

    printf("beginning cpm_main\n");

    /* initialize tables */
    rtimgr_init(rtisvc_tbl_ptr);
    printf("initialized rtisvc_tbl_ptr address:%x\n",rtisvc_tbl_ptr);

    outfile = fopen("hla_cpm_criteria_tbl.out","w");
    if (outfile == NULL)
    {
        printf("CriteriaCreate unable to open Service Criteria table out file \n");
        exit(-1);
    }

    /* print out created tables: */
    /* - criteria table */
    fprintf(outfile,"\n CPM Criteria Table:  \n");
    fprintf(outfile,"Svc Nbr:   Svc Type:           Action Name:           Criteria:
\n");
    for (i=0; i< service_criteria_lines; i++)
    {
        if (!(service_criteria[i].service_nbr))
            continue;

#ifdef 0
        printf("\n\nncriteria table line[%d]: \n",
            i);
        printf("svc nbr:%d service type: %s\n",
            service_criteria[i].service_nbr,
            service_criteria[i].service_type);
#endif
        fprintf(outfile,"%6d      %4s      %34s %20s \n",
            service_criteria[i].service_nbr,
            service_criteria[i].service_type,
            service_criteria[i].rtiamb_action_name,
            service_criteria[i].criteria);
    }
    fclose (outfile);

    /* load and setup the RIDdb info */
    RIDdb_init();

} /* end */
/* ----- DocHeading ---*/
```

```
/* file:   io_manager.c */
/* This manager implements the DES (Discrete Event Simulation)
 * element of the system */

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>

#include "serv_crit.h"
#include "event.h"
#include "rti.h"
#include "proto.h"

#define    DEBUG_IOMGR    0

/* ----- DocComment --*/
/* iomgr_determine_iochannel:
*/
/* ----- DocHeading ---*/
```

```
static int  iomgr_determine_iochannel()
/* ----- EndDocHead---*/

{
    return(RIDdb.network_type);
}

/* ----- DocComment ---*/
/* iomgr_send_ioevent:
 * function to simulate sending an event over the network.
 * The event is added to the output queue via the AddEvent
 * and AddPriorityEvent routines.
 */
/* ----- DocHeading ---*/
```

30 June 1999

```

extern void iomgr_send_ioevent(EVENT_MESSAGE_TYPE *event_msg_info_ptr,
                               double service,
                               int priority)
/* ----- EndDocHead ----*/
{
    int channel_type;
    FEDERATE_DESTINS_TYPE *destinations_list, *element;
    char string[2];

    #if DEBUG_IOMGR
        printf("iomgr_send_ioevent\n");
    #endif

    channel_type = iomgr_determine_iochannel();
    /* update real time value => arrival time + svc complete time */

    /* if ethernet channel send 1 msg for all services */
    if (channel_type == ETHERNET)
    {
        /* place the Fed Amb svc event onto the io event queue */
    #if DEBUG_IOMGR
        printf("adding io_event to ioq on ethernet \n");
    #endif

    // Cristl the time in the event message should be updated Will
    // relative the the current physical time on each federate */
    // :)

    event_msg_info_ptr->Time.PhysicalTime = service +
        CurrentFederateTime( event_msg_info_ptr->Rti.federate_name );

    /* update the color for for these new events */
    event_msg_info_ptr->Color.ColorTag =
        GetColorTag(event_msg_info_ptr->Rti.federate_name);

    if (!priority)
        AddEvent(stdout, "Out", event_msg_info_ptr);
    else /* must be admin event w/priority */
        /* ie, add to front of queue */
        {
            printf(" iomgr_send_ioevent Schedule for %8.5f \n", event_msg_info_ptr-
>Time.PhysicalTime);
            AddPriorityEvent(stdout, "Out", event_msg_info_ptr );
        }

    /* setup complete time for statsmgr reporting */
    event_msg_info_ptr->Time.RTIComplete = event_msg_info_ptr->Time.PhysicalTime;
    event_msg_info_ptr->Time.RTIService =
        event_msg_info_ptr->Time.RTIComplete -
        event_msg_info_ptr->Time.RTIEnter;

    // NOTE      QueuesPrint(stdout) ;
    }
    /* else if ATM send separte messages for services */
    else if (channel_type == ATM)
    {
    #if DEBUG_IOMGR
        printf("adding io_event to ioq on ATM \n");
    #endif
    /* for number of federates */
    destinations_list = event_msg_info_ptr->destinations_list;
    if (destinations_list)
    {
        for (element=destinations_list; element != NULL;

```

30 June 1999

```
                                element = element->next)
{
    /* place a Fed Amb svc event onto the io queues
     * for each recving federate
     */

    /* update real time value =>
     * arrival time + svc complete time
     */
    // iomgr_add_ioevent_to_ioq(event_msg_info_ptr);
    AddEvent(stdout, "Out", event_msg_info_ptr);
}
}

}

/*-----                                DocHeading    */
```

```

/* file: mood.c Xwindow utilities */
/* Modification of basicwin program from Xlib programming manual */
/*-----EndDocHead-----*/
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <X11/Xos.h>

/* gcc -c basicwin.c -I$OPENWINHOME/include */
/* now need to fix the library reference to resolve sysbols */

#include <stdio.h>

#include "bitmaps/basic_bitmap"
#define LINEWIDTH 5
#define XA_BITMAPDEPTH 1 /* Bitmaps have depth 1 */
#define AvailableColors 109
/* Global variables colors 109 from 0 to 109 */
char *colors[] = {
"CornflowerBlue", "cyan", "deeppink2", "green", "LightPink2",
"LightCoral", "tomato", "OrangeRed", "HotPink", "red",

"lightgray", "tan", "DeepPink", "pink", "LightPink",
"cyan", "RosyBrown", "tomato", "CadetBlue", "aquamarine",

"SlateBlue2", "Blue", "SeaGreen", "DodgerBlue2", "magenta2",
"sienna1", "tan", "crimson", "PaleGreen", "DarkOrange",

"ivory2", "RosyBrown3", "LavenderBlush2", "MistyRose2", "NavajoWhite2",
"firebrick", "cyan", "IndianRed", "DarkOrchid", "firebrick",

"cyan", "IndianRed1", "honeydew2", "IndianRed2", "IndianRed3",
"DeepSkyBlue", "SkyBlue", "LightSkyBlue", "LightBlue", "PowderBlue",

"NavyBlue", "CornflowerBlue", "SlateBlue", "MediumBlue", "DodgerBlue",
"DarkKhaki", "PaleGoldenrod", "LightYellow", "gold", "DarkGoldenrod",

"DarkSalmon", "salmon", "orange", "PeachPuff2", "coral",
"goldenrod", "RosyBrown", "IndianRed", "SaddleBrown", "peru",
"DarkSalmon", "salmon", "orange", "PeachPuff2", "coral", "crimson", "maroon",
"DarkGreen", "DarkOliveGreen", "SeaGreen", "ForestGreen", "SpringGreen",
"OliveDrab", "green", "LimeGreen", "PaleGreen", "LawnGreen",
"NavyBlue", "CornflowerBlue", "SlateBlue", "MediumBlue", "DodgerBlue",
"DeepSkyBlue", "SkyBlue", "LightSkyBlue", "LightBlue", "PowderBlue",
"LightSalmon",
"PaleVioletRed", "MediumVioletRed", "azure", "AliceBlue", "MistyRose",
"VioletRed", "magenta", "violet", "plum", "orchid", "LemonChiffon2",
"MediumOrchid", "DarkOrchid", "DarkViolet", "BlueViolet", "MediumPurple",
"thistle", "snow2", "seashell2", "AntiqueWhite2", "bisque2",
"RoyalBlue2", "blue2", "DodgerBlue2", "RosyBrown1", "RosyBrown2", "RosyBrown4",
"IndianRed4", "sienna1", "sienna2", "sienna3", "sienna4",
"burlywood", "beige", "wheat", "SandyBrown", "tan", "gold",
"chocolate", "firebrick", "red", "gray", "HotPink",
"cornsilk2", "azure2"
};

Display *display; /* Physical device information */
int screen; /* Which screen on the device */

Window winM;
Pixmap icon_pixmapM;
XSizeHints size_hintsM; /* For the window manager */
XEvent reportM; /* Elts. in event queue */
GC gcM; /* Graphics context */
XFontStruct *font_infoM;
int acM;

```



```
char *avM[64];
int  Total_Nodes, Total_Objects ;
unsigned int w1140, h867;
unsigned int CurLeft, CurRight; /* Scales to adjust display */
                                int    WidthScale  = 1 ;
                                double HeightScale = 2.0 ; /* 1.5 ; */

int Gline_style = LineSolid;
int Gcap_style  = CapButt; /* CapRound;*/
int Gjoin_style = JoinBevel; /* JoinRound */
int Gdash_offset = LineSolid;
unsigned int Gline_width = LINEWIDTH ;
GC  Ggc;
/* Used to determine if window is too small to be used */

#define SMALL 1
#define OK 0

/* Get the graphics context */

//void get_GC(win, gc, font_info)
// Window win;
// GC *gc;
// XFontStruct *font_info;

/*----- DocHeading */
```

```

extern void get_GC( Window win, GC *gc, XFontStruct *font_info )
/*-----EndDocHead---*/
{
    unsigned long valuemask = 0;
    XGCValues values;
    unsigned int line_width = LINEWIDTH ;
    int line_style = LineSolid; /* LineSolid, LineOnOffDash, LineDoubleDash*/
    int cap_style = CapButt; /* CapNotLast, CapRound; CapProjecting */
    int join_style = JoinBevel; /* JoinRound; JoinMiter*/
    // int dash_offset = LineSolid;
    // static char dash_list[] = { 12, 24 };
    // int list_length = 2;
    Status XGetRGBColormaps();
    // XStandardColormap **std_colormap_return, *C ;
    // int *count_return;
    // Atom property;

    /* Create default graphics context */

    *gc = XCreateGC( display, win, valuemask, &values);
    Ggc = *gc ;

    /* Specify font */
    // property = RGB_DEFAULT_MAP;
    // XGetRGBColormaps(display, win, std_colormap_return, count_return, property);
    // C = *std_colormap_return ;
    // printf("red      %4d  %4d \n", C->red_max,
    //                C->red_mult );
    XSetFont(display, *gc, font_info->fid);

    /* Specify black foreground since default may be white on white */
    // XSetForeground(display, *gc, DefaultColormap(display, screen ));

    XSetForeground(display, *gc, BlackPixel(display, screen));

    /* Set line attributes */

    XSetLineAttributes(display, *gc, line_width, line_style, cap_style,
                        join_style);

    /* Set dashes to be line_width in length */

    /* XSetDashes(display, *gc, dash_offset, dash_list, list_length);*/
}

/* Load in a font  XFontStruct **font_info; */

/*-----DocHeading */

```

```
void load_font(XFontStruct **font_info)
/*-----EndDocHead---*/
{
    char *fontname = "9x15";

    /* Check out that font */

    if ((*font_info = XLoadQueryFont(display, fontname)) == NULL)
    {
        (void) fprintf(stderr, "Cannot open %s font\n", fontname);
        exit (-1);
    }
}

/* Place the text in the window - nicely centered, of course */

/*-----DocHeading */
```

```

extern void draw_text(
    Window win,
    GC gc,
    XFontStruct *font_info,
    unsigned int win_width, unsigned int win_height )
/*-----EndDocHead-----*/
{
    int y = 20; /* Offset from corner of the window */
    char *string[4];
    int i, len[4];
    int width[3];
    char cd[3][50]; /* Window height, width, depth */
    int font_height;
    int initial_y_offset, x_offset;
    char stuff[4][64] ;

/* Init. string values */

    //string[0] = strsave("Radical Window Action!");
    // string[1] = strsave("To terminate program: Press any key");
    // string[2] = strsave("or button while in this window");
    // string[3] = strsave("Screen Dimensions:");

    string[0] = strcpy(stuff[0], "Radical Window Action!");
    string[1] = strcpy(stuff[1], "To terminate program: Press any key");
    string[2] = strcpy(stuff[2], "or button while in this window");
    string[3] = strcpy(stuff[3], "Screen Dimensions:");

/* Calculate the lengths of the strings - XTextWidth and XDrawString need
 * them.
 */

    for (i=0; i<4; i++)
        len[i] = strlen(string[i]);

/* Get string widths for centering */

    for (i=0; i<3; i++)
        width[i] = XTextWidth(font_info, string[i], len[i]);

/* Output text, centered on each line */

    XDrawString(display, win, gc, (win_width-width[0])/2, y, string[0], len[0]);
    XDrawString(display, win, gc, (win_width-width[1])/2,
        (int)(win_height - 35), string[1], len[1]);
    XDrawString(display, win, gc, (win_width-width[2])/2,
        (int)(win_height - 15), string[2], len[2]);

/* Copy numbers into string variables */

    (void) sprintf(cd[0], " Height: %d pixels",
        DisplayHeight(display, screen));
    (void) sprintf(cd[1], " Width: %d pixels",
        DisplayWidth(display, screen));
    (void) sprintf(cd[2], " Depth: %d plane(s)",
        DefaultDepth(display, screen));

/* Reuse the length variables */

    for (i=0; i<3; i++)
        len[i] = strlen(cd[i]);

    font_height = font_info->max_bounds.ascent +
        font_info->max_bounds.descent;

```

30 June 1999

```
initial_y_offset = win_height/2 - font_height -
                    font_info->max_bounds.descent;
x_offset = (int) win_width/4;

/* Draw the strings */

XDrawString(display, win, gc, x_offset, (int)initial_y_offset, string[3],
            len[3]);
for (i=1; i<=3; i++)
    XDrawString(display, win, gc, x_offset, (int)initial_y_offset +
                (i * font_height), cd[i-1], len[i-1]);
}

/*----- DocHeading */
```

30 June 1999

```
extern void draw_graphics(
    Window win,
    GC gc,
    unsigned int window_width, unsigned int window_height)
/*-----EndDocHead---*/
{
    int x, y;
    unsigned int width, height;

    height = window_height/2;
    width = 3 * window_width/4;
    x = window_width/2 - width/2;    /* Center */
    y = window_height/2 - height/2;
    XDrawRectangle(display, win, gc, x, y, width, height);
}

/*----- DocHeading */
```

```
extern void TooSmall(
    Window win,
    GC gc,
    XFontStruct *font_info)
/*-----EndDocHead---*/
{
    char *string1 = "Too Small";
    int x_offset, y_offset;

    y_offset = font_info->max_bounds.ascent + 2;
    x_offset = 2;

    /* Output text, centered on each line */

    XDrawString(display, win, gc, x_offset, y_offset, string1, strlen(string1));
}

/*----- DocHeading */
```

```

extern void Lalalnit(int TotNodes, int TotObjects)
/*-----EndDocHead---*/
{
/* Main routine - initialization and event loop */

    unsigned int width, height;          /* Window size */
    int x = 0, y = 0;                    /* Position within root window */
    // int i;
    // int jj, kk, ll;
    int kk;
    unsigned int border_width = 4;       /* border 4 pixels wide */
    unsigned int display_width,
        display_height;
    char *window_name = "The Window";
    char *icon_name = "The Icon";
    char *display_name = NULL;           /* Use the display on this machine */
    // int window_size = 0;               /* OK or SMALL */

/* Check the command line for a display name */
    if ( TotNodes <= 0 ) { Total_Nodes = 1; }
    else { Total_Nodes = TotNodes; }
    if ( TotObjects <= 0 ) { Total_Objects = 1; }
    else { Total_Objects = TotObjects; }

/*xx for (i=1; i<ac; i++)
    {
        if (strcmp(av[i], "-display") == 0)
            display_name = strsave(av[++i]);
    }
*/
/* Connect to X server */

    if ( (display=XOpenDisplay(display_name)) == NULL)
    {
        (void) fprintf(stderr,
            "%s cannot connect to X server %s\n", avM[0],
            XDisplayName(display_name));
        exit(-1);
    }

/* Determine the size of the screen */

    screen = DefaultScreen(display);     /* Get the screen number */

    display_width = DisplayWidth(display, screen);
    display_height = DisplayHeight(display, screen);
    printf("Xwidth %4d, Xheight %4d \n", display_width, display_height);

/* Size the window so that it has enough room for text */

    width = display_width/WidthScale;
    height = display_height/HeightScale;
    w1140 = width;
    CurRight = width;
    h867 = height;

/* Create an opaque window */

    winM = XCreateSimpleWindow(display, RootWindow(display, screen),
        x, y, width, height, border_width,
        DefaultColormap(display, screen),
        WhitePixel(display, screen));

/* Create a pixmap of depth 1 (bitmap) for the icon */

    icon_pixmapM = XCreateBitmapFromData(display, winM, icon_bitmap_bits,

```



```

        icon_bitmap_width, icon_bitmap_height);

/* Initialize size hint property for window manager */

    size_hintsM.flags = PPosition | PSize | PMinSize;
    size_hintsM.x = x;
    size_hintsM.y = y;
    size_hintsM.width = width;
    size_hintsM.height = height;
    size_hintsM.min_width = 350;
    size_hintsM.min_height = 250;

/* Set the standard properties for the window manager (always do this before
 * mapping the window).
 */

    XSetStandardProperties(display, winM, window_name, icon_name, icon_pixmapM,
                          avM, acM, &size_hintsM);
    printf("XSetStandardProperties ac %3d, av %s \n", acM, avM );
/* Inform the server which events this window is interested in */

    XSelectInput(display, winM, ExposureMask | KeyPressMask | ButtonPressMask |
                 StructureNotifyMask);

/* Set the font */

    load_font(&font_infoM);

/* Create a graphics context for text and drawing */

    get_GC(winM, &gcM, font_infoM); /* this does return gc */

/* Send the window to the display */

    XMapWindow(display, winM);

/* Event loop would need to be implemented somewhere */
    kk=0;
        kk+=1;
        XNextEvent(display, &reportM); /* Get next event */
        XSetForeground(display, gcM, 0xd81384);

/*return(1); */
}
/*----- DocHeading */

```

30 June 1999

```
extern void LalaClear() {  
char str[12];  
    XClearWindow(display, winM);  
    XClearArea(display, winM, 0,0,0,0, False);  
// printf("Cleared Window? press Enter\n");  
// gets(str);  
}
```

/*-----

DocHeading */

30 June 1999

```

extern void LalaUpdate(int Node, int ObjId, int State)
/*-----EndDocHead---*/
{
    //int jj,
    int group, row, col, x1,y1,x2,y2 ;
    XColor      srcColor, dummyColor;
    int Acolor ;

    group = (w1140/ Total_Nodes) * Node ;
    row   = ObjId/ (( w1140/Total_Nodes)/10) ;
    col   = ObjId - row * (( w1140/Total_Nodes)/10) ;
    x1 = group + col * 10;
    y1 = row * 5 + 1 ;
    x2 = x1 + 10 ;
    y2 = y1 ;
    Acolor = State;
    if ( State > AvailableColors ) Acolor = State % AvailableColors;
    XAllocNamedColor(display, DefaultColormap(display, 0), colors[Acolor], &srcColor, &dummyColor);

    XSetForeground( display, Ggc, srcColor.pixel);

    XDrawLine(display, winM, gcM, x1, y1, x2, y2 );

    XMapWindow(display, winM);
    /* XNextEvent(display, &reportM); */ /* Get next event */
    /* while (XCheckTypedEvent(display, Expose, &reportM)); */
    /* for (jj=1+kk; jj<1140; jj += 5) {
        XDrawLine(display, winM, gcM, jj, 0, jj, 867 );
        XSetForeground(display,gcM, State );
    } */
}
/*-----DocMethod */

extern void LalaColor( int State )
/*-----EndDocHead---*/
{
    printf("    %10s %1d \n", colors[State], State );
}

/*-----DocMethod */

extern void LalaPlace( int State, int X, int Y )
/*-----EndDocHead---*/
{
    //int jj,kk;
    //int group, row, col, x1,y1,x2,y2 ;
    int x1,y1,x2,y2 ;

    XColor      srcColor, dummyColor;
    int Acolor ;

    x1 = X;
    y1 = Y;
    x2 = x1 + 5 ;
    y2 = y1 ;
    Acolor = State;
    if ( State > AvailableColors ) Acolor = State % AvailableColors;

    XAllocNamedColor(display, DefaultColormap(display, 0), colors[Acolor], &srcColor, &dummyColor);

    XSetForeground( display, Ggc, srcColor.pixel);

    XDrawLine( display, winM, Ggc, x1, y1, x2, y2 );

    XMapWindow(display, winM);
    /* XNextEvent(display, &reportM); */ /* Get next event */
    /* while (XCheckTypedEvent(display, Expose, &reportM)); */
    /* for (jj=1+kk; jj<1140; jj += 5) {

```

```

        XDrawLine(display, winM, gcM, jj, 0, jj, 867 );
        XSetForeground(display,gcM, State );
    }*/
}

/*----- DocMethod */
extern void LalaDraw(int Node, int ObjId, int State,
                    double x, double y, double xbase, double xrange,
                    double ybase, double yrange )
/*-----EndDocHead---*/
{
    //int jj, kk;
    int x1, y1, x2, y2 ;
    double xdelta, ydelta, ydraw, xdraw ;
    XColor      srcColor, dummyColor;
    int Acolor ;

    xdelta = w1140/xrange;
    ydelta = h867/yrange;

    xdraw = xdelta * (x-xbase);
    ydraw = ydelta * (y-ybase);

    x1 = (int)xdraw ;
    y1 = (int)ydraw;

    x2 = x1 +5;
    y2 = y1 ;
    /*printf(
    "Draw %8.8x %6.2f, %6.2f, %6.2f, %6.2f, %6.2f, %6.2f, %4d, %4d, %4d, %4d \n",
    State, x, y, xbase, xrange, ybase, yrange, x1, y1, x2, y2 );*/
    x1 = x1 % w1140 ;
    y1 = y1 % h867 ;
    x2 = x2 % w1140 ;
    y2 = y2 % h867 ;
    Acolor = State;
    if ( State > AvailableColors ) Acolor = State % AvailableColors;
    XAllocNamedColor(display, DefaultColormap(display, 0), colors[Acolor], &srcColor, &dummyColor);

    XSetForeground( display, Ggc, srcColor.pixel);

    XDrawLine(display, winM, gcM, x1, y1, x2, y2 );
    XMapWindow(display, winM);
}
/*----- DocHeading */

```

```

extern void LalaTimeQueue(int Node, int State,
    double xS, double xE, double xbase, double xrange, int Tag, int Id )
/*-----EndDocHead---*/
{
    //int jj, kk;
    int x1, x2, y1 ;
    double xdelta, x2draw, xdraw ;
    XColor      srcColor, dummyColor;
    int Acolor ;

    xdelta = w1140/xrange;

    y1 = Node * ((h867 - 40)/(Total_Nodes+1)) + 20 ;

    xdraw = xdelta * (xS-xbase);

    x1 = (int)(xdraw) ;

    x2draw = xdelta * (xE-xbase) ;
    x2 = (int)(x2draw) ;
    if (x1 == x2) { x2 = x1 + 1; }
    /*printf(
    "Draw %3d, %6.2f, %6.2f, %6.2f, %6.2f, %4d, %4d, %4d \n",
    State, xS, xE, xbase, xrange, x1, y1, x2 );*/
    x1 = x1 % w1140 ;
    x2 = x2 % w1140 ;
    if ( xdraw > 0.0 && x1 <= x2 ){
        Acolor = State ;
        if ( Acolor > AvailableColors ) Acolor = Acolor % AvailableColors;
        XAllocNamedColor(display, DefaultColormap(display, 0), colors[Acolor], &srcColor,
        &dummyColor);

        XSetForeground( display, Ggc, srcColor.pixel);

        XDrawLine(display, winM, gcM, x1, y1, x2, y1 );
        XMapWindow(display, winM);
        if ( Tag > 0 ) {
            y1 = y1 + Id ;
            Acolor = Tag + 20 ;
            XAllocNamedColor(display, DefaultColormap(display, 0),
            colors[Acolor], &srcColor, &dummyColor);

            XSetForeground( display, Ggc, srcColor.pixel);

            XDrawLine(display, winM, gcM, x1, y1, x2, y1 );
        }
        XMapWindow(display, winM);
    }
} /* end of LalaTimeQueue */

/*-----DocMethod */
extern void LalaDrawLink( int State,
    int x1, int y1, int x2, int y2 )
/*-----EndDocHead---*/
{
    //int jj, kk;
    //int group, row, col;
    unsigned int Lcl_line_width ;
    //double xdelta, ydelta, ydraw, xdraw ;
    XColor      srcColor, dummyColor;
    int Acolor ;
    /* Set line attributes */
    Lcl_line_width = 1;
    XSetLineAttributes(display, Ggc, Lcl_line_width, Gline_style, Gcap_style,
    Gjoin_style);

```

30 June 1999

```

Acolor = State;
if ( State > AvailableColors ) Acolor = State % AvailableColors;
XAllocNamedColor(display, DefaultColormap(display, 0), colors[Acolor], &srcColor, &dummyColor);
XSetForeground( display, Ggc, srcColor.pixel);

        XDrawLine(display, winM, gcM, x1, y1, x2, y2 );
        XMapWindow(display, winM);

XSetLineAttributes(display, Ggc, Gline_width, Gline_style, Gcap_style,
        Gjoin_style);

}

unsigned int GetXWidth()    { return( w1140 );    }
unsigned int GetXLenght()  { return( h867  );    }

/*-----                                DocMethod    */
extern void LalaFinished()
/*-----EndDocHead---*/
{
/* Exit gracefully */

        XUnloadFont(display, font_infoM->fid);
        XFreeGC(display, gcM);
        XCloseDisplay(display);
}
/* ----- DocHeading---*/

```

/* file: object_mgmt_svc.c */

/* This file contains the RTI Ambassador Services (or action methods)
 * for the Object Mgmt services
 */

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <strings.h>
#include <errno.h>

#include "serv_crit.h"
#include "event.h"
#include "rti.h"
#include "rtimgr.h"
#include "rti_services.h"

#define DEBUG_OBJ_MGMT 0

/* ----- DocComment---*/
/* create_regions_node
*/
/* ----- DocHeading---*/

30 June 1999

```
static REGIONS_LIST_TYPE *create_regions_node(int region_nbr)
/* ----- EndDocHead---*/
{
    REGIONS_LIST_TYPE *node;

    node = (REGIONS_LIST_TYPE *) calloc(1, sizeof(REGIONS_LIST_TYPE));
    if (!node)
    {
        printf("create_regions_node: ERROR-unable to create node storage\n");
        return(NULL);
    }
    node->region_nbr = region_nbr;
    node->next = NULL;

    return(node);
}

/* ----- DocComment---*/
/* create_obj_instance:
*/
/* ----- DocHeading---*/
```


30 June 1999

```

static OBJECT_INSTANCE_TYPE *create_obj_instance(REGIONS_LIST_TYPE *regions_ptr,
                                                int      federate_nbr,
                                                int      class_nbr)
/* ----- EndDocHead----- */
{
    OBJECT_INSTANCE_TYPE *obj_instance;

    obj_instance = (OBJECT_INSTANCE_TYPE *) calloc(1, sizeof(OBJECT_INSTANCE_TYPE));

    if (!obj_instance)
    {
        printf("create_obj_instance: ERROR-unable to create obj_instance storage\n");
        return(NULL);
    }
    obj_instance->owner_federate = federate_nbr;
    obj_instance->objclass_nbr = class_nbr;
    obj_instance->associated_regions = regions_ptr;
/*    obj_instance->next = NULL;    */

    return(obj_instance);
}

/* ----- DocComment----- */
/* add_obj_to_federates_table:
*/
/* ----- DocHeading----- */

```

30 June 1999

```
static void add_obj_to_federates_table(OBJECT_INSTANCE_TYPE *node,
                                       int federate_nbr)
/* ----- EndDocHead---*/
{
/*
   if (FedExdb.federates[federate_nbr].registered_obj_instances == NULL)
       FedExdb.federates[federate_nbr].registered_obj_instances = node;
*/
/*
   else
       FedExdb.federates[federate_nbr].registered_obj_instances->next = node;
*/

}

/* ----- DocComment---*/
/* om_is_class_published:
*/
/* ----- DocHeading---*/
```

```
extern int om_is_class_published(int class_nbr,
                                int class_type)
/* ----- EndDocHead---*/
{
    if (class_type == OBJECT_TYPE)
    {
        if (!(FedExdb.fedrtn_objclasses[class_nbr].published))
            return(FALSE);
        else
            return(TRUE);
    }
    else if (class_type == INTERACTION_TYPE)
    {
        if (!(FedExdb.fedrtn_interact_classes[class_nbr].published))
            return(FALSE);
        else
            return(TRUE);
    }
    else
    {
        printf("om_is_class_published: ERROR\n");
        return(FALSE);
    }
}
/*----- DocComment---*/
/*  om_instance_exists:
*/
/* ----- DocHeading---*/
```

```
extern int om_instance_exists(int instance_nbr)
{
    if ((instance_nbr <= FedExdb.nbr_fedrtn_obj_instances) &&
        (FedExdb.fedrtn_obj_instances[instance_nbr].owner_federate))
        return(TRUE);
    else
        return(FALSE);
}

/* ----- DocComment---*/
/* om_register_instance:
*/
/* ----- DocHeading---*/
```

30 June 1999

```

extern double om_register_instance(int class_nbr,
                                   int instance_nbr,
                                   int region_nbr,
                                   int federate_nbr,
                                   EVENT_MESSAGE_TYPE *event_msg_info_ptr,
                                   int *rtiamb_nbr_federates)
/* ----- EndDocHead----- */
{
    REGIONS_LIST_TYPE *regions_ptr, *region_list;
    double svc_statistic=0.0;

#ifdef DEBUG_OBJ_MGMT
    printf("om_register_instance- class:%d instance:%d region:%d federate:%d\n",
           class_nbr, instance_nbr, region_nbr, federate_nbr);
#endif
    /* Don't process when the instance to be created is either:
       - for an invalid class or
       - from an unpublished class */
    if ((class_nbr > FOMdb.nbr_fedrtn_objclasses) ||
        (!om_is_class_published(class_nbr, OBJECT_TYPE)))
    {
        printf("om_register_instance-ERROR:invalid class:%d for this instance\n",
               class_nbr);
        return(0.0);
    }

    /* if instance already exists */
    if (om_instance_exists(instance_nbr))
    {
        region_list
            = FedExdb.fedrtn_obj_instances[instance_nbr].associated_regions;
        /* only add region when it is not already associated */
        for (region_list; region_list != NULL; region_list=region_list->next)
        {
            if (region_list->region_nbr == region_nbr)
                break;
        }
        if (region_list == NULL)
        {
#ifdef DEBUG_OBJ_MGMT
            printf("adding regions node:%x to instance:%d regions list:%x \n",
                   regions_ptr, instance_nbr, region_list);
#endif
            /* create a regions node for this instance */
            if (!(regions_ptr = create_regions_node(region_nbr)))
            {
                printf("om_register_instance-ERROR: null regions_ptr \n");
                return(0.0);
            }
            FedExdb.fedrtn_obj_instances[instance_nbr].associated_regions
                = regions_ptr;
#ifdef DEBUG_OBJ_MGMT
            printf("instance:%d now assoc also with region:%d\n",
                   instance_nbr, region_nbr);
#endif
        }
    }
    /* else new instance */
    else if (FedExdb.nbr_fedrtn_obj_instances < MAX_NBR_OBJ_INSTANCES)
    {
        /* if instance is out of sequence */
        if (!(FedExdb.nbr_fedrtn_obj_instances+1 == instance_nbr))
        {
#ifdef DEBUG_OBJ_MGMT
            printf("om_register_instance: warning-instance nbr:%d out of sequence\n",

```

30 June 1999

```

        instance_nbr);
#endif
    /* instance within max nbr instances */
    if (instance_nbr < MAX_NBR_OBJ_INSTANCES)
        FedExdb.nbr_fedrtn_obj_instances = instance_nbr;
    else
    {
        printf("om_register_instance: ERROR: instance nbr:%d exceeds
MAX_NBR_OBJ_INSTANCES\n", instance_nbr);
        return(0.0);
    }
    /* instance in sequence */
    else
        FedExdb.nbr_fedrtn_obj_instances++;

    /* create a regions node for this instance */
    if (!(regions_ptr = create_regions_node(region_nbr)))
    {
        printf("om_register_instance-ERROR: null regions_ptr \n");
        return(0.0);
    }

    FedExdb.fedrtn_obj_instances[FedExdb.nbr_fedrtn_obj_instances].owner_federate
        = federate_nbr;
    FedExdb.fedrtn_obj_instances[FedExdb.nbr_fedrtn_obj_instances].objclass_nbr
        = class_nbr;
    FedExdb.fedrtn_obj_instances[FedExdb.nbr_fedrtn_obj_instances].associated_regions
        = regions_ptr;

    #if DEBUG_OBJ_MGMT
        printf("created new instance:%d associated with region:%d\n",
            instance_nbr, region_nbr);
        printf("added regions node:%x to list:%x \n",
            regions_ptr, regions_ptr->next);
    #endif
    }
    else
    {
        #if DEBUG_OBJ_MGMT
            printf("om_register_instance:ERROR- unable to make another instance, reached
MAX_NBR_OBJ_INSTANCES\n");
        #endif
        return(0.0);
    }

    svc_statistic =
        rtimgr_get_RTI_ambsvc_time(federate_nbr,
            RTI_REGISTER_INST,
            1);

    svc_statistic +=
        om_setup_discover_events(federate_nbr,
            class_nbr,
            region_nbr,
            event_msg_info_ptr,
            rtiamb_nbr_federates);

    return(svc_statistic);
}

/* ----- DocComment---*/
/* om_is_object_registered:
*/
/* ----- DocHeading---*/

```

30 June 1999

```
extern int om_is_object_registered(int instance_nbr)
/* ----- EndDocHead---*/
{
    if (FedExdb.fedrtn_obj_instances[instance_nbr].owner_federate)
        return(TRUE);
    else
        return(FALSE);
}

/* ----- DocComment---*/
/*  om_update_attrib_values:
*/
/* ----- DocHeading---*/
```

30 June 1999

```

extern double om_update_attrib_values(int federate_nbr,
                                     int instance_nbr,
                                     int tag_nbr,
                                     int fedrtn_time,
                                     EVENT_MESSAGE_TYPE *event_msg_info_ptr,
                                     int *rtiamb_nbr_federates)
/* ----- EndDocHead---*/
{
    double svc_statistic=0.0;

    /* future: handle tag nbr and fedrtn_time */
    if (!om_is_object_registered(instance_nbr))
    {
        printf("om_update_attrib_values: ERROR- unable to update unregistered instance:%d\n",
               instance_nbr);
        return(0.0);
    }

    svc_statistic =
        rtimgr_get_RTI_ambsvc_time(federate_nbr, RTI_UPDATE_ATTRIB,1);

    svc_statistic +=
        om_setup_reflect_events(federate_nbr,
                                instance_nbr,
                                tag_nbr,
                                fedrtn_time,
                                event_msg_info_ptr,
                                rtiamb_nbr_federates);

    return(svc_statistic);
}

/* ----- DocComment---*/
/* om_send_interaction
*/
/* ----- DocHeading---*/

```



```

extern double om_send_interaction(int federate_nbr,
                                int class_nbr,
                                int interact_instance_nbr,
                                int region_nbr,
                                int tag_nbr,
                                int fedrtn_time,
                                EVENT_MESSAGE_TYPE *event_msg_info_ptr,
                                int *rtiamb_nbr_federates)
/* ----- EndDocHead-----*/

{
    double svc_statistic=0.0;

#ifdef DEBUG_OBJ_MGMT
    printf("processing om_send_interaction- federate:%d class:%d instance:%d region:%d\n",
           federate_nbr, class_nbr, interact_instance_nbr, region_nbr);
#endif

    /* future: handle tag nbr and fedrtn_time */
    if (!om_is_class_published(class_nbr, INTERACTION_TYPE))
    {
        printf("om_send_interaction: ERROR- federate:%d class:%d instance:%d region:%d\n",
               federate_nbr, class_nbr, interact_instance_nbr, region_nbr);
        return(0.0);
    }
    FedExdb.nbr_fedrtn_interact_instances++;

#ifdef DEBUG_OBJ_MGMT
    printf("fedrtn_interact_instances:%d\n",
           FedExdb.nbr_fedrtn_interact_instances);
#endif

    svc_statistic =
        rtimgr_get_RTI_ambsvc_time(federate_nbr, RTI_SEND_INT, 1);

    svc_statistic +=
        om_setup_receive_interaction_events(federate_nbr,
                                           class_nbr,
                                           interact_instance_nbr,
                                           region_nbr,
                                           tag_nbr,
                                           fedrtn_time,
                                           event_msg_info_ptr,
                                           rtiamb_nbr_federates);

    return(svc_statistic);
}

/* ----- DocComment-----*/
/* om_request_attrib_value_update
*/
/* ----- DocHeading-----*/

```

```
extern double om_request_attrib_value_update()
/* ----- EndDocHead---*/

{

/* this svc will cause a query svc to all active federates
 * for info on these attrib/instance values
 */

/* only compute time for initiation of this svc, at this point */
return(1);

/* pair - provide attrib values */

}

/* Fed Amb Services */

/* ----- DocComment---*/
/* om_create_destinations_element
*/
/* ----- DocHeading---*/
```

30 June 1999

```

extern FEDERATE_DESTINS_TYPE *om_create_destinations_element(int federate_nbr)
/* ----- EndDocHead---*/
{
    FEDERATE_DESTINS_TYPE *element;
    char str[12];
    element = (FEDERATE_DESTINS_TYPE *) calloc(1, sizeof(FEDERATE_DESTINS_TYPE));
    if (element != NULL)
    {
        element->federate= federate_nbr;
        element->next = NULL;
    }
    else
        printf("error: calloc of FEDERATE_DESTINS_TYPE\n");

    if ( federate_nbr < 1 ) {
        fprintf(stdout,
/* CRISTL */ "om_create_destinations_element CATCH the real culprit Fed %2d %s\n",
        federate_nbr, " when being put on the list, without relent");
        gets(str);
    }

    return(element);

}

/* ----- DocComment---*/
/*  om_discover_object
*/
/* ----- DocHeading---*/

```

30 June 1999

```
extern double om_discover_object(int federate_nbr,
                                int class_nbr,
                                int region_nbr)
/* ----- EndDocHead---*/

{
    double svc_statistic=0.0;

#ifdef DEBUG_OBJ_MGMT
    printf("om_discover_object- federate:%d processing class nbr:%d for region:%d \n",
           federate_nbr, class_nbr, region_nbr);
#endif

    svc_statistic =
        rtimgr_get_Fed_ambsvc_time(RTI_DISCVR_OBJ,
                                   1);

    return(svc_statistic);
}

/* Setup routine */
/* ----- DocComment---*/
/*   om_setup_discover_events
*/
/* ----- DocHeading---*/
```

30 June 1999

```

extern double om_setup_discover_events(int federate_nbr,
                                       int class_nbr,
                                       int region_nbr,
                                       EVENT_MESSAGE_TYPE *event_msg_info_ptr,
                                       int *subscribed_nbr)

{
    SUBSCRIBED_INFO_TYPE *subscribed_list;
    int tmp_federate_nbr;
    FEDERATE_DESTINS_TYPE *dest_element, *destinations_list;
    double svc_statistic=0.0;

    #if DEBUG_OBJ_MGMT
    printf("om_setup_discover_events- federate:%d processing class nbr:%d for region:%d feds:%d\n",
          federate_nbr, class_nbr, region_nbr, *subscribed_nbr);
    #endif

    destinations_list = event_msg_info_ptr->destinations_list;
    subscribed_list = FedExdb.fedrtn_regions[region_nbr].subscribed_objects;
    if (subscribed_list == NULL)
    {
        #if DEBUG_OBJ_MGMT
        printf("om_setup_discover_events- note:no federates subscribed to region:%d\n",
              region_nbr);
        #endif
    }
    /* for each node in the subscribed list, setup an event */
    for (subscribed_list; subscribed_list != NULL;
        subscribed_list=subscribed_list->next)
    {
        #if DEBUG_OBJ_MGMT
        printf("processing class:%d \n",
              subscribed_list->class_nbr);
        #endif
        if (class_nbr == subscribed_list->class_nbr)
        {
            tmp_federate_nbr = subscribed_list->federate_name;
            #if DEBUG_OBJ_MGMT
            printf("class w/ region subscribed to federate:%d \n", tmp_federate_nbr);
            #endif
            dest_element = om_create_destinations_element(tmp_federate_nbr);
            if (dest_element)
            {
                #if DEBUG_OBJ_MGMT
                printf("adding new element:%x onto list:%x\n",
                      dest_element, destinations_list);
                #endif
                if (*subscribed_nbr)
                    dest_element->next = destinations_list;
                destinations_list = dest_element;
                (*subscribed_nbr)++;
            }
            else
            {
                #if DEBUG_OBJ_MGMT
                printf("error- unable to create new dest element\n");
                #endif
            }
        }
    }
    /* when subscribers exist for this registered instance,
    * setup/change the event's svc nbr to
    * the fedamb pair event name:discover for the further events

```

30 June 1999

```
    */
    if (*subscribed_nbr)
    {
        event_msg_info_ptr->Rti.rti_svc_nbr = RTI_DISCVR_OBJ;
        event_msg_info_ptr->destinations_list = destinations_list;
        eventmgr_change_processing_mode(event_msg_info_ptr, RTI_PROCESSING);
    }
    svc_statistic =
        rtimgr_get_RTI_ambsvc_time(federate_nbr, RTI_DISCVR_SETUP,
                                   *subscribed_nbr);

    return(svc_statistic);
}

/* Fed Amb service */
/* ----- DocComment---*/
/* om_reflect_attrib_values:
*/
/* ----- DocHeading---*/
```

30 June 1999

```
extern double om_reflect_attrib_values(int instance_nbr,
                                       int class_nbr,
                                       int tag_nbr,
                                       int fedrtn_time)
/* ----- EndDocHead---*/

{
    double svc_statistic=0.0;

    #if DEBUG_OBJ_MGMT
    printf("processing reflect for instance:%d with class:%d \n",
          instance_nbr,
          class_nbr);
    #endif

    svc_statistic =
        rtimgr_get_Fed_ambsvc_time(RTI_REFLECT_ATTRIB,
                                   1);

    return(svc_statistic);
}

/* ----- DocComment---*/
/* om_setup_reflect_events:
*/
/* ----- DocHeading---*/
```

30 June 1999

```

extern double om_setup_reflect_events(int federate_nbr,
                                     int instance_nbr,
                                     int tag_nbr,
                                     int fedrtn_time,
                                     EVENT_MESSAGE_TYPE *event_msg_info_ptr,
                                     int *subscribed_federates)

/* ----- EndDocHead----- */
{
    int class_nbr;
    REGIONS_LIST_TYPE *obj_assoc_regions;
    SUBSCRIBED_INFO_TYPE *subscribed_list;
    int tmp_federate_nbr;
    FEDERATE_DESTINS_TYPE *dest_element, *destinations_list;
    double svc_statistic=0.0;

    #if DEBUG_OBJ_MGMT
    printf("processing om_setup_reflect_events for instance:%d \n",
           instance_nbr);
    #endif

    destinations_list = event_msg_info_ptr->destinations_list;
    class_nbr = FedExdb.fedrtn_obj_instances[instance_nbr].objclass_nbr;

    #if DEBUG_OBJ_MGMT
    printf(" class: %d \n", class_nbr);
    #endif
    obj_assoc_regions = FedExdb.fedrtn_obj_instances[instance_nbr].associated_regions;

    for (obj_assoc_regions; obj_assoc_regions!=NULL;
         obj_assoc_regions=obj_assoc_regions->next)
    {
        subscribed_list =FedExdb.fedrtn_regions[obj_assoc_regions-
        >region_nbr].subscribed_objects;

        #if DEBUG_OBJ_MGMT
        printf(" processing region:%d \n",
               obj_assoc_regions->region_nbr);
        #endif

        for (subscribed_list; subscribed_list != NULL;
             subscribed_list= subscribed_list->next)
        {
            #if DEBUG_OBJ_MGMT
            printf(" processing class:%d \n",
                   subscribed_list->class_nbr);
            #endif

            if (subscribed_list->class_nbr == class_nbr)
            {
                tmp_federate_nbr = subscribed_list->federate_name;
                dest_element = om_create_destinations_element(tmp_federate_nbr);
                if (*subscribed_federates)
                    dest_element->next = destinations_list;
                destinations_list = dest_element;
                (*subscribed_federates)++;
            }
            #if DEBUG_OBJ_MGMT
            printf("added federate:%d to destinations list\n", tmp_federate_nbr);
            #endif
        }
    }

    /* when subscribers exist for this update instance,
     * setup/change the event's svc nbr to
     * the fedamb pair event name:reflect for the further events

```


30 June 1999

```
*/
if (*subscribed_federates)
{
    event_msg_info_ptr->Rti.rti_svc_nbr = RTI_REFLECT_ATTRIB;
    event_msg_info_ptr->destinations_list = destinations_list;
    eventmgr_change_processing_mode(event_msg_info_ptr, RTI_PROCESSING);
}

/* future: handle the tag nbr, fedrtn time and event nbr */

    svc_statistic =
        rtimgr_get_RTI_ambsvc_time(federate_nbr, RTI_REFLECT_SETUP,
                                   *subscribed_federates);
return(svc_statistic);

}

/* ----- DocComment---*/
/*  om_receive_interaction
*/
/* ----- DocHeading---*/
```

30 June 1999

```
extern double om_receive_interaction(int class_nbr,
                                     int instance_nbr)
/* ----- EndDocHead---*/

{
    double svc_statistic=0.0;

    #if DEBUG_OBJ_MGMT
    printf("processing om_receive_interaction for instance:%d with class:%d \n",
           instance_nbr, class_nbr);
    #endif

    svc_statistic =
        rtimgr_get_Fed_ambsvc_time(RTI_RECEIVE_INT,
                                   1);

    return(svc_statistic);

}
/* ----- DocComment---*/
/* om_setup_receive_interaction_events
*/
/* ----- DocHeading---*/
```

30 June 1999

```

extern double om_setup_receive_interaction_events(int federate_nbr,
                                                int    class_nbr,
                                                int    instance_nbr,
                                                int    region_nbr,
                                                int    tag_nbr,
                                                int    fedrtn_time,
                                                EVENT_MESSAGE_TYPE *event_msg_info_ptr,
                                                int    *subscribed_federates)
/* ----- EndDocHead----- */
{
    SUBSCRIBED_INFO_TYPE *subscribed_list;
    int tmp_federate_nbr;
    FEDERATE_DESTINS_TYPE *dest_element, *destinations_list;
    double svc_statistic=0.0;

    #if DEBUG_OBJ_MGMT
    printf("processing om_setup_receive_interaction_events for class:%d region:%d\n",
          class_nbr, region_nbr);
    #endif

    destinations_list= event_msg_info_ptr->destinations_list;

    /* compute nbr nodes subscribed for this interaction class */
    /* loop through list of subscribed federates and
     * send an event to all applicable
     */

    subscribed_list =
        FedExdb.fedrtn_regions[region_nbr].subscribed_interactions;
    #if DEBUG_OBJ_MGMT
    printf("processing region:%d \n",
          region_nbr);
    #endif
    for (subscribed_list; subscribed_list != NULL;
        subscribed_list= subscribed_list->next)
    {
        if (subscribed_list->class_nbr == class_nbr)
        {
            tmp_federate_nbr = subscribed_list->federate_name;
            dest_element = om_create_destinations_element(tmp_federate_nbr);
            if (*subscribed_federates)
                dest_element->next = destinations_list;
            destinations_list = dest_element;
            (*subscribed_federates)++;
            #if DEBUG_OBJ_MGMT
            printf("added federate:%d to destinations list\n",
                  tmp_federate_nbr);
            #endif
        }
    }

    /* when subscribers exist for this interaction class,
     * setup/change the event's svc nbr to
     * the fedamb pair event name:receive for the further events
     */
    if (*subscribed_federates)
    {
        event_msg_info_ptr->Rti.rti_svc_nbr = RTI_RECEIVE_INT;
        event_msg_info_ptr->destinations_list = destinations_list;
        eventmgr_change_processing_mode(event_msg_info_ptr, RTI_PROCESSING);
    }

    svc_statistic =
        rtimgr_get_RTI_ambsvc_time(federate_nbr, RTI_RECEIVE_INT_SETUP,

```

30 June 1999

```
                                *subscribed_federates);  
    return(svc_statistic);  
}  
  
/* ----- DocComment---*/  
/*  om_provide_attrib_value_update  
*/  
/* ----- DocHeading---*/
```

30 June 1999

```
extern double om_provide_attrb_value_update()
/* ----- EndDocHead---*/

{
    int  nbr_feds_in_fedrtn=0;
    double  svc_statistic=0.0;

    /* compute nbr current active federates */

    svc_statistic =
        rtimgr_get_Fed_ambsvc_time(RTI_PRVD_ATTRIB_VALS,
                                   nbr_feds_in_fedrtn);
    return(svc_statistic);
}

/* ----- DocHeading ---*/
```

/* file: rti_utils.c */

```

/*****
--*  get_string_peices was previously csv() i.e., renamed
--*  int get_string_peices( char *lstr, char *pieces[], char *delimiter )
--*      parses up a string by some single delimiter and allocates pieces of memory *
--*  */
*****/

#include <stdlib.h>
#include <errno.h>
#include <stdio.h>
#include <string.h>
#include "serv_crit.h"

#define EOLN    '\n'
#define RETURN '\r'
#define CRITERIA_START_OFFSET    3
#define CRITERIA_END_OFFSET      30
/* ----- DocComment ---*/
/*  get_string_peices:
*/
/* ----- DocHeading ---*/
```

30 June 1999

```

extern int get_string_peices( char *lstr, char *pieces[], char *delimiter)
/* ----- EndDocHead----- */
{
    char *bptr,*eptr;
    int i;
    /* begin ptr, end ptr, list string */
    bptr = eptr = lstr ;
    #if 0
        printf("\nget_string_peices(csv): bptr-%s \n delimiter:%s \n", bptr, delimiter );
    #endif
    i = 0 ;
    while( (bptr != NULL) && (eptr != NULL) &&
        (*bptr != EOLN) && (*bptr != RETURN))
    {
        /* find the first delimiter in the string, and extract substring */
        eptr = strstr( bptr, delimiter );
        /* if endptr indicates substring retrieved or
           out of delimiters, and not at end of line, extract last substring
        */
        if ( eptr != NULL || eptr == NULL && bptr < lstr + (strlen(lstr)-1) )
        {
            /* if last substring to extract, set the eptr correctly */
            if ( eptr == NULL && bptr < lstr + (strlen(lstr)-1) )
            {
                eptr = lstr + (strlen(lstr)-1);
            }
            pieces[i] = (char *) malloc((eptr-bptr)+2);

            if ( errno == EINVAL )
            {
                printf(" get_string_peices: (EINVAL) Indicates a call has requested 0 bytes. \n" );
                printf(" get_string_peices: @ %s \n", lstr );
                return(-1) ;
            }
            if ( errno == ENOMEM )
            {
                printf(" get_string_peices: (ENOMEM) Indicates that not enough storage space was
available. \n" );
                printf(" get_string_peices: @ %s \n", lstr );
                return(-1) ;
            }

            strncpy( pieces[i], bptr, eptr-bptr ) ;
            *(pieces[i]+(eptr-bptr) ) = '\0' ;
        }
        #if 0
            printf("peice[%d]: %s \n", i, pieces[i] );
        #endif
        i += 1 ;
        bptr = eptr + 1;
    } /* end of if - not end of delimiters in string */
    } /* end of while - not end of line string */
    #if 0
        printf("\n");
    #endif
    return(i);
} /* end of get_string_peices - parse a line given a delimiter */

/* ----- DocComment ----- */
/* change_strgpeices_to_onestrng:
 *   convert list of peices into one criteria string for later compares
 *   against the Federation status string.
 *   Also, validate the table input for valid entries.
 */
/* ----- DocHeading ----- */

```

30 June 1999

```

extern int change_strgpeices_to_onestrng(int      num_peices,
                                           char      *peices[128],
                                           char      criteria[80])
/* ----- EndDocHead---*/
{
    int i, j=0;

    #if 0
    printf("in change_strgpeices_to_onestrng\n");
    #endif

    /* start passed the service nbr and type peices */
    for (i=CRITERIA_START_OFFSET; i < num_peices && i < CRITERIA_END_OFFSET; i++)
    {
        if (!strcmp(peices[i], "T", 1))
            criteria[j] = '1';
        else if (!strcmp(peices[i], "F", 1))
            criteria[j] = '0';
        else if (!strcmp(peices[i], "o", 1)) /* optional- don't care */
            criteria[j] = 'x';
        else if (!strcmp(peices[i], "n", 1)) /* not applicable */
            criteria[j] = '*';
        else
        {
            criteria[j] = '*'; /* use n/a */
        }
        #if 0
        printf("invalid entry for criteria[%d]\n", i);
        #endif
        j++;
    }
    criteria[i] = '\0';
    #if 0
    printf("criteria string:%s \n", criteria);
    criteria = "test";
    printf("test criteria: %s\n", criteria);
    #endif
    return(1);
}

/* ----- DocHeading ---*/

```



```

/* file: rti_manager.c */

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <strings.h>
#include <errno.h>
#include <math.h>

#include "serv_crit.h"
#include "event.h"
#include "rti.h"
#include "rtmgr.h"
#include "rti_services.h"
#include "statsmgr.h"
#include "proto.h"

/* turn on/off printf stmts in the rti_manager.c file */
#define  DEBUG_RTI_MGR  0

/* permit creation of federate processing files -
 * which keep track:
 * - of all RTI service processing at a federate
 * - are useful for verification of federate
 *   pair services exchanges:
 *       register/discover
 *       update/reflect instances
 *       send/receive interactions
 */
#define  REPORT_FEDERATE_PROCESSING 0

extern double  rtmgr_get_RTI_ambsvc_time(int  federate_nbr,
                                         int   RTIambsvc,
                                         int   nbr_federates);

extern double  rtmgr_get_Fed_ambsvc_time(int   fed_ambsvc,
                                         int   nbr_federates);

static void    rtmgr_retrieve_svctblinfo(RTI_EVENT_MSG_TYPE    *svc_msg_info,
                                         RTI_SERVICE_TBL_ENTRY_TYPE  *current_rti_tblsvc);

static int     rtmgr_criteria_compare(FEDEX_STATE_INFO  fedex_state_status,
                                       char               *criteria_strg);

static double  rtmgr_compute_elapsed_time_statistic(int           federate_nbr,
                                                      int  service_time,
                                                      EVENT_MESSAGE_TYPE    *event_msg_info_ptr);

/* globally available variables */
RTI_SERVICE_TBL_ENTRY_TYPE  service_criteria[SERVICE_CRITERIA_MAX_DATA_LINES];
int                          service_criteria_lines=0;
RTI_FEDAMB_SERVICE_TYPE    fedamb_svcs[SERVICE_CRITERIA_MAX_DATA_LINES];
int                          nbr_fedamb_svcs=0;
FILE                       *rti_svcstats_outfiles[MAX_NBR_FEDERATES+1];

#define LINE_LENGTH  1000

/* ----- DocHeading -----*/

```

```

static int rtimgr_criteria_create(RTI_SERVICE_TBL_ENTRY_TYPE *rtisvc_tbl_ptr)
/* ----- EndDocHead----- */

{

/* ----- DocComment ----- */
/* criteria_create: */
/* Function which reads in a Criteria table for RTI Services */

FILE *infile;
int i,j, linestatus;
int m=0;
RTI_SERVICE_TBL_ENTRY_TYPE *crit_table_entry;
char line[LINE_LENGTH], *list[128];
char criteria[CRITERIA_MAX];

#if 0
    printf("criteria_create rtn\n");
    printf("rtisvc_tbl_ptr address:%x\n",rtisvc_tbl_ptr);
#endif

    infile = fopen("hla_cpm_criteria_params.txt", "r" );

    if (infile == NULL)
    { printf("CriteriaCreate unable to open Service Criteria table file \n");
      return(0);
    }
    crit_table_entry = rtisvc_tbl_ptr; /* init start and prev table holders */
#if 0
    printf("starting table entry address:%x\n", rtisvc_tbl_ptr);
#endif
    linestatus = (int)fgets( line, LINE_LENGTH, infile );
    line[ (strlen(line) -1) ] = 0 ;

    #if 0
        printf("\n First retrieved title line:\n %s \n", line);
        printf("\n CPM Criteria Table:  \n %s \n", line);
    #endif

    for (i=0; i < SERVICE_CRITERIA_MAX_DATA_LINES; i++)
    {
        bzero(line, LINE_LENGTH);
        bzero(criteria, CRITERIA_MAX);
        bzero(crit_table_entry, sizeof(RTI_SERVICE_TBL_ENTRY_TYPE));

        linestatus = (int)fgets( line, LINE_LENGTH, infile );
        line[ (strlen(line) -1) ] = 0 ;
        if ( linestatus != 0 )
        {
            #if 0
                printf("\n next retrieved line[%d]:\n%s\n", i, line );
            #endif
            j = get_string_peices(&line[0], list, "," );
            #if 0
                printf("j: %d peices of line\n", j);
            #endif
            if ( !j )
            {
                #if 0
                    printf("unable to parse line for segments\n");
                #endif
            }
            if (!strcmp(list[0], "")) /* skip lines of Mgmt types; no service data */
            {
                #if 0
                    printf("list[0]:%s == NULL\n",list[0]);
                #endif
            }
        }
    }
}

```

```

#endif
    continue;
}
else /* if had peices in line then process */
{
    if (j <= 3)
    {
#if 0
        printf("Too few elements to Criteria file entry... not saving in table \n");
#endif
        continue;
    }
    change_strgpeices_to_onestrng(j, list, criteria);
    strcpy(crit_table_entry->criteria, criteria);
#if 0
    printf("crit_table_entry address:%x\n", crit_table_entry);
    printf("put criteria:%s in table \n",
           crit_table_entry->criteria);
#endif
    crit_table_entry->service_nbr = atoi(list[0]);
#if 0
    printf("put service_nbr:%d in table \n",
           crit_table_entry->service_nbr);
#endif
    strcpy(crit_table_entry->service_type, list[1]);
#if 0
    printf("put service_type:%s in table \n",
           crit_table_entry->service_type);
#endif
    crit_table_entry->rtiamb_action = atoi(list[33]);
#if 0
    printf("put rtiamb_action:%d in table \n",
           crit_table_entry->rtiamb_action);
#endif
    strcpy(crit_table_entry->rtiamb_action_name, list[34]);
#if 0
    printf("put rtiamb_action_name:%s in table \n",
           crit_table_entry->rtiamb_action_name);
#endif
    crit_table_entry->fedamb_reaction = atoi(list[36]);
#if 0
    printf("put fedamb_reaction:%d in table \n",
           crit_table_entry->fedamb_reaction);
#endif
    if ( strlen(list[37]) < MAX_SVC_NAME_LEN)
    {
        strncpy(crit_table_entry->fedamb_reaction_name,
                list[37], strlen(list[37]) );
    }
    else
    {
        strncpy(crit_table_entry->fedamb_reaction_name,
                list[37], MAX_SVC_NAME_LEN);
    }
#if 0
    printf("wills info-list[37]:%s len:%d \n",
           list[37], strlen(list[37]));
    printf("put fedamb_reaction_name:%s in table \n",
           crit_table_entry->fedamb_reaction_name);
#endif
    if (crit_table_entry->fedamb_reaction)
    {
        fedamb_svcs[m].fedamb_reaction =
            crit_table_entry->fedamb_reaction;
        strcpy(fedamb_svcs[m].fedamb_reaction_name,
               crit_table_entry->fedamb_reaction_name);
    }
#if 0

```

```
                printf("put fedamb svc in fedamb_svcs table\n");
#endif
                m++;
            }
            crit_table_entry++;    /* bump to next table entry */

        } /* end of else- setup of criteria peices */
    } /* end of valid line to process */
} /* end of for loop of lines */
printf("saved %d lines into services table\n", i);

service_criteria_lines = i;
nbr_fedamb_svcs = m;
printf("saved %d fedamb svcs into fedamb table\n", m);

fclose (infile);
return(1);
}

/* ----- DocComment---*/
/* rtimgr_FOMdb_init:
*/
/* ----- DocHeading---*/
```

```

static void rtimgr_FOMdb_init()
/* ----- EndDocHead----- */

{
    int i;

    /* init -hardwire for now
       future: read in FOM vals from rdr file
    */
    FOMdb.fedrtn_name = 1;
    FOMdb.fedrtn_rti_version= 1.32;
    FOMdb.fedrtn_rti_version= 1.32;
    FOMdb.fedrtn_global_lookahead_value = 2.0;
    FOMdb.nbr_routing_spaces = 100;
    FOMdb.nbr_fedrtn_objclasses = 10;
    /* fill for now with random values */
    for (i=0; i < FOMdb.nbr_fedrtn_objclasses; i++)
    {
        FOMdb.object_classes[i].associated_routing_space = i+2;
        FOMdb.object_classes[i].associated_nbr_attrib_parms = i;
    }
    FOMdb.nbr_fedrtn_interact_classes = 10;
    for (i=0; i < FOMdb.nbr_fedrtn_interact_classes; i++)
    {
        FOMdb.interact_classes[i].associated_routing_space = i+1;
        FOMdb.interact_classes[i].associated_nbr_attrib_parms = i;
    }
    /* setup reduction network configuration for LBTS reporting */

    /* test LBTS configuration - MAX_NBR_FEDERATES = 10 */
    #if 1
    printf("configured for 10 federates \n");
    /* LBTS configuration - MAX_NBR_FEDERATES = 10 */
    FOMdb.LBTS_controller = 1;
    FOMdb.nodes[1].parent = -1;
    FOMdb.nodes[1].children = TRUE;
    FOMdb.nodes[2].parent = 1;
    FOMdb.nodes[2].children = TRUE;
    FOMdb.nodes[3].parent = 1;
    FOMdb.nodes[3].children = TRUE;
    FOMdb.nodes[4].parent = 2;
    FOMdb.nodes[4].children = FALSE;
    FOMdb.nodes[5].parent = 2;
    FOMdb.nodes[5].children = FALSE;
    FOMdb.nodes[6].parent = 2;
    FOMdb.nodes[6].children = FALSE;
    FOMdb.nodes[7].parent = 3;
    FOMdb.nodes[7].children = FALSE;
    FOMdb.nodes[8].parent = 3;
    FOMdb.nodes[8].children = FALSE;
    FOMdb.nodes[9].parent = 3;
    FOMdb.nodes[9].children = FALSE;
    FOMdb.nodes[10].parent = 3;
    FOMdb.nodes[10].children = FALSE;

    #else
    /* test LBTS configuration - MAX_NBR_FEDERATES = 5 */
    printf("configured for 5 federates \n");

    /* test LBTS configuration - MAX_NBR_FEDERATES = 5 */
    FOMdb.LBTS_controller = 1;
    FOMdb.nodes[1].parent = -1;
    FOMdb.nodes[1].children = TRUE;
    FOMdb.nodes[2].parent = 1;
    FOMdb.nodes[2].children = TRUE;
    FOMdb.nodes[3].parent = 2;

```

```
FOMdb.nodes[3].children = FALSE;
FOMdb.nodes[4].parent = 2;
FOMdb.nodes[4].children = FALSE;
FOMdb.nodes[5].parent = 2;
FOMdb.nodes[5].children = FALSE;

/*
#else
    printf("ERROR: unable to configure reduction network \n");
*/

#endif
}

/* ----- DocComment---*/
/* rtimgr_clear_reduction_network_info:
*/
/* ----- DocHeading---*/
```

30 June 1999

```

extern void rtimgr_clear_reduction_network_info()
/* ----- EndDocHead---*/
{
    int i,j;
    int parent;
    LBTS_REDCTN_NETWK_INFO *federate_info;

    /* clear all values for initialization on federate time mgmt */

    for (i=1; i<= MAX_NBR_FEDERATES; i++)
    {
        if (parent = FOMdb.nodes[i].parent)
        {
            federate_info = &FedExdb.federates[parent].reduction_network_info;
            for (j=0; j < MAX_NBR_CHILDREN; j++)
            {
                federate_info->children_reported[j] = FALSE;
            }
            FedExdb.federates[i].reduction_network_info.i_reported = FALSE;
            FedExdb.federates[i].reduction_network_info.sent_initial_counts = FALSE;
            FedExdb.federates[i].reduction_network_info.total_rcvd_msgs =
                FedExdb.federates[i].reduction_network_info.total_sent_msgs = 0;
            FedExdb.federates[i].reduction_network_info.best_LBTS = 0.0;
        }
    }
}

/* ----- DocHeading---*/

```

30 June 1999

```

static void rtimgr_FedExdb_init()
/* ----- EndDocHead-----*/

{
/* ----- DocComment-----*/
/* rtimgr_FedExdb_init:
*   Initialize all values at the Fed Execution before any runs.
*/
    int i,j;
    int parent;
    int offset;
    LBTS_REDUCTN_NETWK_INFO *federate_info;

    /* init */
    FedExdb.fedrtn_name = 0;
    for (i=0; i < FOMdb.nbr_fedrtn_objclasses; i++)
    {
        FedExdb.fedrtn_objclasses[i].associated_routing_space
            = FOMdb.object_classes[i].associated_routing_space;
        FedExdb.fedrtn_objclasses[i].nbr_attribs
            = FOMdb.object_classes[i].associated_nbr_attrib_parms;
        FedExdb.fedrtn_objclasses[i].owner_federate = 0;
        FedExdb.fedrtn_objclasses[i].published = FALSE;
        FedExdb.fedrtn_objclasses[i].nbr_subscribed_federates = 0;
    }
    FedExdb.nbr_member_federates = 0;
    FedExdb.nbr_fedrtn_obj_instances = 0;
    for (i=0; i < FOMdb.nbr_fedrtn_interact_classes; i++)
    {
        FedExdb.fedrtn_interact_classes[i].associated_routing_space
            = FOMdb.interact_classes[i].associated_routing_space;
        FedExdb.fedrtn_interact_classes[i].nbr_parms
            = FOMdb.interact_classes[i].associated_nbr_attrib_parms;
        FedExdb.fedrtn_interact_classes[i].owner_federate = 0;
        FedExdb.fedrtn_interact_classes[i].published = FALSE;
        FedExdb.fedrtn_interact_classes[i].nbr_subscribed_federates = 0;
    }
    FedExdb.nbr_fedrtn_interact_instances = 0;
    bzero(&FedExdb.fedex_state_status, sizeof(FEDEX_STATE_INFO));
    FedExdb.nbr_fedrtn_regions = 0;
    FedExdb.nbr_save_names = 0;

    /* time mgmt */
    FedExdb.fedrtn_time_mgmt_info.global_lookahead_value =
        FOMdb.fedrtn_global_lookahead_value;
    FedExdb.fedrtn_time_mgmt_info.LBTS_current = 0.0;
    FedExdb.fedrtn_time_mgmt_info.LBTS_proposed = 0.0;
    FedExdb.fedrtn_time_mgmt_info.current_state = INACTIVE; /* INACTIVE LBTS mode */

    rtimgr_clear_reduction_network_info();

    /* setup all federates reduction network info for federate time mgmt */

    for (i=1; i<= MAX_NBR_FEDERATES; i++)
    {
        FedExdb.federates[i].reduction_network_info.nbr_children = 0;
        if (parent = FOMdb.nodes[i].parent)
        {
            federate_info =
                &FedExdb.federates[parent].reduction_network_info;
            federate_info->children_names[federate_info->nbr_children] = i;
            federate_info->nbr_children += 1;
        }
    }
}

```


}

/* ----- DocHeading---*/

30 June 1999

```

extern void rtimgr_init(RTI_SERVICE_TBL_ENTRY_TYPE *rtisvc_tbl_ptr)
/* ----- EndDocHead----- */

{
/* ----- DocComment----- */
/* rtimgr_init:
 * initialize rti model tables
 */

    int            i;
    char            prefix_strg[30]="hla_cpm_svcstatstbl_out.fed";
    char            file_strgname[32];

    printf("beginning rti_init\n");

#ifdef 0
    printf("service_criteria address:%x\n", service_criteria);
    printf("service_criteria[0].service_type contents:%d\n",
        service_criteria[0].service_type);
#endif

    rtisvc_tbl_ptr = &service_criteria[0];
#ifdef 0
    printf("initial rtisvc_tbl_ptr address:%x\n",rtisvc_tbl_ptr);
#endif

    /* load and setup the criteria table */
    if (!(rtimgr_criteria_create(rtisvc_tbl_ptr)))
        exit(-1);

    /* load and setup the FOMdb info */
    rtimgr_FOMdb_init();
    /* initialized the FedExdb info */
    rtimgr_FedExdb_init();

#ifdef REPORT_FEDERATE_PROCESSING
    /* initialize Federate specific files */
    for (i=1; i <= MAX_NBR_FEDERATES; i++)
    {
        sprintf(file_strgname,"%s%d",prefix_strg,i);
        rti_svcstats_outfiles[i] = fopen(&file_strgname[0],"w");
        if (rti_svcstats_outfiles[i] == NULL)
            printf("Error-Unable to open output file:%s \n", file_strgname);
        else
        {
            printf("creating outfile:%s for verification stats on fed:%d\n",
                file_strgname, i);
            fprintf(rti_svcstats_outfiles[i],
                "\nRTI Services Table for federate:%d      date: \n",
                    i);
            fprintf(rti_svcstats_outfiles[i],
                "\nService      Service      Time      Originating  Nbr nodes  Unique\n");
            fprintf(rti_svcstats_outfiles[i],
                "Number:      Name:      elapsed:      node:      affected:  Msg Id:\n");

        } /* files opened successfully */
    } /* loop of federates */
#endif

    /* create and open a statistics output file */
    statsmgr_init_statruns_file();

}
/* ----- DocComment----- */
/* rtimgr_federate_processed_initial_counts
 */

```

30 June 1999

/* ----- DocHeading---*/

30 June 1999

```
extern int rtimgr_federate_processed_initial_counts(int federate_nbr)
/* ----- EndDocHead---*/
{
    if (FedExdb.federates[federate_nbr].reduction_network_info.sent_initial_counts)
        return(TRUE);

    else
        return(FALSE);

}

/* ----- DocComment---*/
/* rtimgr_federate_is_parent
*/
/* ----- DocHeading---*/
```

30 June 1999

```
extern int rtimgr_federate_is_parent(int federate_nbr)
{
    if (FOMdb.nodes[federate_nbr].children)
        return(TRUE);
    else
        return(FALSE);
}
```

```
/* ----- DocComment---*/
/* rtimgr_update_fedrtn_state_status:
*/
/* ----- DocHeading---*/
```

30 June 1999

```
static void rtmgr_update_fedrtm_state_status(FEDEX_STATE_INFO fedex_state)
/* ----- EndDocHead---*/
{
    /* retrieve state information from fedex and set status accordingly */
    /* note: most of these values will be kept up with from
       * other functions */

    #if DEBUG_RTI_MGR
        printf("\ncurrent values of fedex_state:\n");
        printf("FedExNameExists:%d save_in_process:%d\n",
            fedex_state.FedEx_name_exists,
            fedex_state.save_in_process);
    #endif
        ;
    }

/* ----- DocHeading---*/
```

30 June 1999

```

static void    rtimgr_printsvc_stat(int  federate_nbr,
                                     int    action,
                                     char    *action_name,
                                     double  statistic,
                                     int    originating_federate,
                                     int    nbr_federates,
                                     int    msg_id)
/* ----- EndDocHead---*/
{
/* ----- DocComment---*/
/*  rtimgr_printsvc_stat:
 *   Output the RTI and Fed Ambassador service times to individual
 *   Federate files for verification and debugging of the RTI services.
 *   Note: only call this function when REPORT_FEDERATE_PROCESSING is on.
 */
#ifdef DEBUG_RTI_MGR
    printf("rtimgr_printsvc_stat-\n");
    printf("action nbr:    action:            statistic: \n");
    printf(" %6d %20s    %6.4f \n",
           action, action_name, statistic);
#endif

#ifdef REPORT_FEDERATE_PROCESSING
    /* print statistic to the outfile table */
    fprintf(rti_svcstats_outfiles[federate_nbr], "%5d %27s  %6.4f    %3d    %3d\n",
           action, action_name, statistic,
           originating_federate, nbr_federates, msg_id);
#else
    printf("REPORT_FEDERATE_PROCESSING disabled\n");
#endif
}

/* ----- DocComment---*/
/*  rtimgr_compute_elapsed_time_statistic:
 */
/* ----- DocHeading---*/

```

```
static double rtimgr_compute_elapsed_time_statistic(int    federate_nbr,
                                                    int    service_time,
                                                    EVENT_MESSAGE_TYPE
                                                    *event_msg_info_ptr)

/* ----- EndDocHead-----*/
{
    event_msg_info_ptr->Time.PhysicalTime = service_time +
        CurrentFederateTime(federate_nbr);
    event_msg_info_ptr->Time.RTIComplete =
        event_msg_info_ptr->Time.PhysicalTime;
    event_msg_info_ptr->Time.RTIService =
        event_msg_info_ptr->Time.RTIComplete -
        event_msg_info_ptr->Time.RTIEnter;
    if (event_msg_info_ptr->Time.RTIService < 0)
    {
        printf("rtimgr_compute_elapsed_time_statistic-ERROR: invalid elapsed time. \n");
        event_msg_info_ptr->Time.RTIService = 0.0;
    }

    return(event_msg_info_ptr->Time.RTIService);
}

/* ----- DocComment-----*/
/* rtimgr_process_rtiamb_svc()
*/
/* ----- DocHeading-----*/
```


30 June 1999

```

static double rtimgr_process_rtiamb_svc(int rtiamb_action,
                                         char *action_name,
                                         RTI_EVENT_MSG_TYPE *svc_msg_info,
                                         EVENT_MESSAGE_TYPE *event_msg_info_ptr)
/* ----- EndDocHead----- */
{
    double rtiamb_svc_statistic = 0.0;
    double service_elapsed_time_statistic = 0.0;
    /* nbr nodes/federate affected by the action */
    int rtiamb_nbr_federates = 0;
    int index;
    int change_processing_mode = TRUE;

#ifdef DEBUG_RTI_MGR
    printf("\nrtimgr_process_rtiamb_svc- action:%d \n",
          rtiamb_action);
#endif

    /* process rti ambassador service */
    switch (rtiamb_action) {
        case RTI_CREATE_FEDEX :
            printf("current phys time:%d \n",
                  event_msg_info_ptr->Time.PhysicalTime);
            rtiamb_svc_statistic =
                fm_create_fedrtn_execution(svc_msg_info->federate_name,
                                           svc_msg_info->fedrtn_exname);
            service_elapsed_time_statistic = rtiamb_svc_statistic;
            rtiamb_nbr_federates = 1;
            index =
                statsmgr_get_statsarray_index(RTI_CREATE_FEDEX,
                                              svc_msg_info->federate_name);
            statsmgr_collect_statistic(index, service_elapsed_time_statistic);

            break;
        case RTI_JOIN_FEDEX :
            rtiamb_svc_statistic =
                fm_join_fedrtn_execution(svc_msg_info->federate_name,
                                         svc_msg_info->fedrtn_exname);
            service_elapsed_time_statistic = rtiamb_svc_statistic;
            index =
                statsmgr_get_statsarray_index(RTI_JOIN_FEDEX,
                                              svc_msg_info->federate_name);
            statsmgr_collect_statistic(index, service_elapsed_time_statistic);

            break;
        case RTI_RQST_FEDRTN_SAVE :
            rtiamb_svc_statistic =
                fm_request_fedrtn_save(svc_msg_info->federate_name,
                                       &rtiamb_nbr_federates,
                                       event_msg_info_ptr);

            service_elapsed_time_statistic = rtiamb_svc_statistic;
            if (rtiamb_nbr_federates)
            {
                iomgr_send_ioevent(event_msg_info_ptr, rtiamb_svc_statistic, FALSE);
                service_elapsed_time_statistic = event_msg_info_ptr->Time.RTIService;
            }
            index =
                statsmgr_get_statsarray_index(RTI_RQST_FEDRTN_SAVE,
                                              svc_msg_info->federate_name);
            statsmgr_collect_statistic(index, service_elapsed_time_statistic);

            break;
        case RTI_FED_SAVE_BEGUN :
            rtiamb_svc_statistic =
                fm_federate_save_began(svc_msg_info->federate_name,

```

30 June 1999

```

                                &rtiamb_nbr_federates);
service_elapsed_time_statistic = rtiamb_svc_statistic;
index =
    statsmgr_get_statsarray_index(RTI_FED_SAVE_BEGUN,
                                svc_msg_info->federate_name);
statsmgr_collect_statistic(index, service_elapsed_time_statistic);

    break;
case RTI_FED_SAVE_COMPLETE:
    rtiamb_svc_statistic =
        fm_federate_save_achieved(svc_msg_info->federate_name);
    service_elapsed_time_statistic = rtiamb_svc_statistic;
    /* test the remaining federates saving status
       for possible federation save complete
    */
    if (fm_is_fedrtn_saved())
    {
        FedExdb.fedex_state_status.save_in_process = FALSE;
        rtiamb_svc_statistic +=
            fm_setup_fedrtn_complete_events(svc_msg_info->federate_name,
                                            &rtiamb_nbr_federates,
                                            event_msg_info_ptr);
        service_elapsed_time_statistic = rtiamb_svc_statistic;
        if (rtiamb_nbr_federates)
        {
            iomgr_send_ioevent(event_msg_info_ptr, rtiamb_svc_statistic, FALSE);
            service_elapsed_time_statistic =
                event_msg_info_ptr->Time.RTIService;
        }
    }
    index =
        statsmgr_get_statsarray_index(RTI_FED_SAVE_COMPLETE,
                                    svc_msg_info->federate_name);
    statsmgr_collect_statistic(index, service_elapsed_time_statistic);

    break;
case RTI_PUBLISH_OBJCLSS :
    rtiamb_svc_statistic =
        dm_publish_objclass(svc_msg_info->federate_name,
                            svc_msg_info->obj_class_nbr);
    service_elapsed_time_statistic = rtiamb_svc_statistic;
    index =
        statsmgr_get_statsarray_index(RTI_PUBLISH_OBJCLSS,
                                    svc_msg_info->federate_name);
    statsmgr_collect_statistic(index, service_elapsed_time_statistic);

    break;
case RTI_PUBLISH_INTCLSS :
    rtiamb_svc_statistic =
        dm_publish_interact_class(svc_msg_info->federate_name,
                                svc_msg_info->interact_class_nbr);
    service_elapsed_time_statistic = rtiamb_svc_statistic;
    index =
        statsmgr_get_statsarray_index(RTI_PUBLISH_INTCLSS,
                                    svc_msg_info->federate_name);
    statsmgr_collect_statistic(index, service_elapsed_time_statistic);

    break;
case RTI_SUBSCRIBE_INTCLSS :
    rtiamb_svc_statistic =
        dm_subscribe_interact_class(svc_msg_info->federate_name,
                                svc_msg_info->interact_class_nbr,
                                svc_msg_info->region_nbr);
    service_elapsed_time_statistic = rtiamb_svc_statistic;
    index =
        statsmgr_get_statsarray_index(RTI_SUBSCRIBE_INTCLSS,

```

30 June 1999

```

                                svc_msg_info->federate_name);
    statsmgr_collect_statistic(index, service_elapsed_time_statistic);

    break;
case RTI_SUBSCRIBE_OBJCLSS :
    rtiamb_svc_statistic =
        dm_subscribe_objclass(svc_msg_info->obj_class_nbr,
                                svc_msg_info->federate_name,
                                svc_msg_info->region_nbr);
    service_elapsed_time_statistic = rtiamb_svc_statistic;
    index =
        statsmgr_get_statsarray_index(RTI_SUBSCRIBE_OBJCLSS,
                                        svc_msg_info->federate_name);
    service_elapsed_time_statistic = rtiamb_svc_statistic;
    statsmgr_collect_statistic(index, service_elapsed_time_statistic);

    break;
case RTI_REGISTER_INST :

    /* compute how long it took to determine rtiamb svc
       destinations with db complexity formula
    */
    rtiamb_svc_statistic =
        om_register_instance(svc_msg_info->obj_class_nbr,
                                svc_msg_info->obj_instance_nbr,
                                svc_msg_info->region_nbr,
                                svc_msg_info->federate_name,
                                event_msg_info_ptr,
                                &rtiamb_nbr_federates);
    service_elapsed_time_statistic = rtiamb_svc_statistic;

    if (rtiamb_nbr_federates)
    {
        iomgr_send_ioevent(event_msg_info_ptr, rtiamb_svc_statistic, FALSE );
        change_processing_mode = FALSE;
        /* report elapsed service time */
        service_elapsed_time_statistic=
            event_msg_info_ptr->Time.RTIService;
    }
    index =
        statsmgr_get_statsarray_index(RTI_REGISTER_INST,
                                        svc_msg_info->federate_name);
    statsmgr_collect_statistic(index, service_elapsed_time_statistic);

    break;

case RTI_UPDATE_ATTRIB :
    rtiamb_svc_statistic =
        om_update_attrib_values(svc_msg_info->federate_name,
                                svc_msg_info->obj_instance_nbr,
                                svc_msg_info->tag_name,
                                svc_msg_info->fedrtn_time,
                                event_msg_info_ptr,
                                &rtiamb_nbr_federates);

    service_elapsed_time_statistic = rtiamb_svc_statistic;
    if (rtiamb_nbr_federates)
    {
        iomgr_send_ioevent(event_msg_info_ptr, rtiamb_svc_statistic, FALSE );
        change_processing_mode = FALSE;
        /* report elapsed service time */
        service_elapsed_time_statistic= event_msg_info_ptr->Time.RTIService;
    }
    index =
        statsmgr_get_statsarray_index(RTI_UPDATE_ATTRIB,

```

```

                                svc_msg_info->federate_name);
    statsmgr_collect_statistic(index, service_elapsed_time_statistic);

    break;

case RTI_SEND_INT :
    rtiamb_svc_statistic =
        om_send_interaction(svc_msg_info->federate_name,
                            svc_msg_info->interact_class_nbr,
                            svc_msg_info->interact_instance_nbr,
                            svc_msg_info->region_nbr,
                            svc_msg_info->tag_name,
                            svc_msg_info->fedrtn_time,
                            event_msg_info_ptr,
                            &rtiamb_nbr_federates);

    service_elapsed_time_statistic = rtiamb_svc_statistic;
    if (rtiamb_nbr_federates)
    {
        iomgr_send_ioevent(event_msg_info_ptr, rtiamb_svc_statistic, FALSE);
        change_processing_mode = FALSE;
        /* report elapsed service time */
        service_elapsed_time_statistic= event_msg_info_ptr->Time.RTIService;
    }

    index = statsmgr_get_statsarray_index(RTI_SEND_INT,
                                           svc_msg_info->federate_name);
    statsmgr_collect_statistic(index, service_elapsed_time_statistic);

    break;

case RTI_RQST_ATTRIB_VALS :
    rtiamb_svc_statistic =
        om_request_attrib_value_update();
    service_elapsed_time_statistic = rtiamb_svc_statistic;
    index =
        statsmgr_get_statsarray_index(RTI_RQST_ATTRIB_VALS,
                                       svc_msg_info->federate_name);
    statsmgr_collect_statistic(index, service_elapsed_time_statistic);

    break;

case RTI_CREATE_UPDATE_REGION:
    rtiamb_svc_statistic =
        ddm_create_update_region(svc_msg_info->federate_name,
                                 svc_msg_info->region_nbr);
    service_elapsed_time_statistic = rtiamb_svc_statistic;
    index = statsmgr_get_statsarray_index(RTI_CREATE_UPDATE_REGION,
                                           svc_msg_info->federate_name);
    statsmgr_collect_statistic(index, service_elapsed_time_statistic);
    break;

case RTI_ASSOC_REGION :
    rtiamb_svc_statistic =
        ddm_associate_update_region(svc_msg_info->federate_name);
    service_elapsed_time_statistic = rtiamb_svc_statistic;
    index = statsmgr_get_statsarray_index(RTI_ASSOC_REGION,
                                           svc_msg_info->federate_name);
    statsmgr_collect_statistic(index, service_elapsed_time_statistic);
    break;

case RTI_TIME_ADV_RQST:
    rtiamb_svc_statistic =
        tm_time_adv_request(svc_msg_info->federate_name,
                            svc_msg_info->fedrtn_time,
                            &rtiamb_nbr_federates,
                            event_msg_info_ptr);

```

30 June 1999

```

        service_elapsed_time_statistic = rtiamb_svc_statistic;
        index = statsmgr_get_statsarray_index(RTI_TIME_ADV_RQST,
                                              svc_msg_info->federate_name);

        change_processing_mode = FALSE;
        statsmgr_collect_statistic(index, service_elapsed_time_statistic);
        break;
    case RTI_RPTNG_FED_LBTS:
    case RTI_RPTNG_RCV_LBTS:
    case RTI_RPTNG_SND_LBTS:
        rtiamb_svc_statistic =
            tm_controller_LBTS_compute(svc_msg_info->federate_name,
                                      &rtiamb_nbr_federates,
                                      event_msg_info_ptr);

        service_elapsed_time_statistic=
            rtimgr_compute_elapsed_time_statistic(
                svc_msg_info->federate_name,
                rtiamb_svc_statistic,
                event_msg_info_ptr);
        index = statsmgr_get_statsarray_index(RTI_RPTNG_FED_LBTS,
                                              svc_msg_info->federate_name);
        /* preserve processing mode from compute rtn */
        change_processing_mode = FALSE;
        statsmgr_collect_statistic(index, service_elapsed_time_statistic);
        break;
    default: printf("ERROR:no process service for svc:%d\n", rtiamb_action);
}
#endif
#ifdef DEBUG_RTI_MGR
    printf("rtiamb_svc_statistic:%f\n", rtiamb_svc_statistic);
#endif

#ifdef REPORT_FEDERATE_PROCESSING
    rtimgr_printsvcs_stat(svc_msg_info->federate_name,
                        rtiamb_action,
                        action_name,
                        rtiamb_svc_statistic,
                        svc_msg_info->origin_fed_name,
                        rtiamb_nbr_federates,
                        event_msg_info_ptr->Time.UniqueMsgId);
#endif

    if (change_processing_mode)
        eventmgr_change_processing_mode(event_msg_info_ptr, DONE_PROCESSING);

    return(rtiamb_svc_statistic);
}
/* ----- DocComment-----*/
/* rtimgr_process_fedamb_svc
*/
/* ----- DocHeading-----*/

```

30 June 1999

```

extern double rtimgr_process_fedamb_svc(int          fedamb_reaction,
                                           char      *reaction_name,
                                           RTI_EVENT_MSG_TYPE *current_rti_svc_msg,
                                           EVENT_MESSAGE_TYPE *event_msg_info_ptr)
/* ----- EndDocHead----- */
{
    double fedamb_svc_statistic = 0.0;
    double service_elapsed_time_statistic = 0.0;
    int    nbr_federates;
    int    index;
    int    change_processing_mode ;

    change_processing_mode = TRUE ;
#ifdef DEBUG_RTI_MGR
    printf("\nrtimgr_process_fedamb_svc- reaction:%d and name:%s\n",
          fedamb_reaction, reaction_name);
#endif

    /* process pair rti service => federate ambassador service */
    switch (fedamb_reaction) {
        case RTI_INITIATE_FED_SAVE :
            fedamb_svc_statistic = fm_initiate_federate_save();
            nbr_federates = 1;
            service_elapsed_time_statistic =
                rtimgr_compute_elapsed_time_statistic(
                    current_rti_svc_msg->federate_name,
                    fedamb_svc_statistic,
                    event_msg_info_ptr);
            index = statsmgr_get_statsarray_index(RTI_INITIATE_FED_SAVE,
                                                  current_rti_svc_msg->federate_name);
            statsmgr_collect_statistic(index, service_elapsed_time_statistic);
            break;
        case RTI_FEDRTN_SAVED :
            fedamb_svc_statistic = fm_fedrtn_save_achieved();
            nbr_federates = 1;
            service_elapsed_time_statistic =
                rtimgr_compute_elapsed_time_statistic(
                    current_rti_svc_msg->federate_name,
                    fedamb_svc_statistic,
                    event_msg_info_ptr);
            index = statsmgr_get_statsarray_index(RTI_FEDRTN_SAVED,
                                                  current_rti_svc_msg->federate_name);
            statsmgr_collect_statistic(index, service_elapsed_time_statistic);
            break;
        case RTI_DISCVR_OBJ :
            fedamb_svc_statistic =
                om_discover_object(current_rti_svc_msg->federate_name,
                                  current_rti_svc_msg->obj_class_nbr,
                                  current_rti_svc_msg->region_nbr);
            nbr_federates = 1;
            service_elapsed_time_statistic =
                rtimgr_compute_elapsed_time_statistic(
                    current_rti_svc_msg->federate_name,
                    fedamb_svc_statistic,
                    event_msg_info_ptr);
            index = statsmgr_get_statsarray_index(RTI_DISCVR_OBJ,
                                                  current_rti_svc_msg->federate_name);
            statsmgr_collect_statistic(index, service_elapsed_time_statistic);
            break;
        case RTI_REFLECT_ATTRIB :
            fedamb_svc_statistic =
                om_reflect_attrib_values(current_rti_svc_msg->obj_instance_nbr,
                                          current_rti_svc_msg->obj_class_nbr,

```

30 June 1999

```

                                current_rti_svc_msg->tag_name,
                                current_rti_svc_msg->fedrtn_time);

    nbr_federates =1;
    service_elapsed_time_statistic =
        rtimgr_compute_elapsed_time_statistic(
            current_rti_svc_msg->federate_name,
            fedamb_svc_statistic,
            event_msg_info_ptr);

    index = statsmgr_get_statsarray_index(RTI_REFLECT_ATTRIB,
                                           current_rti_svc_msg->federate_name);
    statsmgr_collect_statistic(index, service_elapsed_time_statistic);
    break;
case RTI_RECEIVE_INT :
    fedamb_svc_statistic =
        om_receive_interaction(current_rti_svc_msg->interact_class_nbr,
                               current_rti_svc_msg->interact_instance_nbr);

    nbr_federates =1;
    service_elapsed_time_statistic =
        rtimgr_compute_elapsed_time_statistic(
            current_rti_svc_msg->federate_name,
            fedamb_svc_statistic,
            event_msg_info_ptr);

    index = statsmgr_get_statsarray_index(RTI_RECEIVE_INT,
                                           current_rti_svc_msg->federate_name);
    statsmgr_collect_statistic(index, service_elapsed_time_statistic);
    break;
case RTI_PRVD_ATTRIB_VALS :
    fedamb_svc_statistic = om_provide_attrib_value_update();
    nbr_federates =1;
    service_elapsed_time_statistic = fedamb_svc_statistic;
    index = statsmgr_get_statsarray_index(RTI_PRVD_ATTRIB_VALS,
                                           current_rti_svc_msg->federate_name);

    statsmgr_collect_statistic(index, service_elapsed_time_statistic);

    break;
case RTI_QUERY_FED_LBTS:
    fedamb_svc_statistic = tm_query_fed_LBTS(current_rti_svc_msg->federate_name,
                                              event_msg_info_ptr,
                                              &nbr_federates);
    index = statsmgr_get_statsarray_index(RTI_QUERY_FED_LBTS,
                                           current_rti_svc_msg->federate_name);
    service_elapsed_time_statistic=
        rtimgr_compute_elapsed_time_statistic(
            current_rti_svc_msg->federate_name,
            fedamb_svc_statistic,
            event_msg_info_ptr);

    change_processing_mode = FALSE;
    statsmgr_collect_statistic(index, service_elapsed_time_statistic);
    break;
case RTI_TIME_ADV_GRANT:
    fedamb_svc_statistic = tm_time_adv_grant(current_rti_svc_msg->federate_name,
                                              current_rti_svc_msg->fedrtn_time);

    service_elapsed_time_statistic =
        rtimgr_compute_elapsed_time_statistic(
            current_rti_svc_msg->federate_name,
            fedamb_svc_statistic,
            event_msg_info_ptr);

    index
        = statsmgr_get_statsarray_index(RTI_TIME_ADV_GRANT,
                                           current_rti_svc_msg->federate_name);
    statsmgr_collect_statistic(index, service_elapsed_time_statistic);
    break;

```

```
        default: printf("invalid fedamb_action for svc:%d\n", fedamb_reaction);
                nbr_federates =1;
    }
    #if DEBUG_RTI_MGR
        printf("fedamb_svc_statistic:%6.4f \n", fedamb_svc_statistic);
    #endif

    #if REPORT_FEDERATE_PROCESSING
        /* compute fedamb_svc_statistic above,
         * ie, how long it took to determine fedamb svc destinations
         * using complexity formula
         */
        rtimgr_printsvc_stat(current_rti_svc_msg->federate_name,
                            fedamb_reaction,
                            reaction_name,
                            fedamb_svc_statistic,
                            current_rti_svc_msg->origin_fed_name,
                            nbr_federates,
                            event_msg_info_ptr->Time.UniqueMsgId);
    #endif

    if (change_processing_mode)
        eventmgr_change_processing_mode(event_msg_info_ptr, SIM_PROCESSING);

    return(fedamb_svc_statistic);
}

/* ----- DocComment---*/
/* rtimgr_is_fedamb_svc:
*/
/* ----- DocHeading---*/
```


30 June 1999

```
static int rtimgr_is_fedamb_svc(int fedamb_svcnbr,
                                char *fedamb_reaction_name)
{
    int i;

    /* valid rti fedamb services */
    for (i=0; i < nbr_fedamb_svcs; i++)
    {
        if (fedamb_svcs[i].fedamb_reaction == fedamb_svcnbr)
        {
            strncpy(fedamb_reaction_name,
                    fedamb_svcs[i].fedamb_reaction_name,
                    MAX_SVC_NAME_LEN);
            return(TRUE);
        }
    }
    return(FALSE);
}

/* ----- DocComment---*/
/* rtimgr_RTEvent:
*/
/* ----- DocHeading---*/
```

30 June 1999

```

extern double rtimgr_RTLevent(EVENT_MESSAGE_TYPE *event_msg_info_ptr)

/* ----- EndDocHead----- */
{
    RTI_EVENT_MSG_TYPE          *svc_msg_info;
    RTI_SERVICE_TBL_ENTRY_TYPE  rtisvc_tblentry;
    double                      svc_statistic;
    char                        fedamb_svc_name[MAX_SVC_NAME_LEN];

    #if DEBUG_RTI_MGR
    printf("rtimgr_RTLevent-\n");
    #endif

    /* point at rti svc portion of event msg */
    svc_msg_info = &event_msg_info_ptr->Rti;

    /* when not fedamb svc, must be rtiamb svc */
    if (!rtimgr_is_fedamb_svc(svc_msg_info->rti_svc_nbr,
                            &fedamb_svc_name[0]))
    {
        /* lookup eventsvc in criteria_table, retrieve criteria and other items */
        rtimgr_retrieve_svctblinfo(svc_msg_info, &rtisvc_tblentry);
    #if 0
        /* if (eventsvc) not found */
        if (error)
            printf("eventsvc nbr:%d not found  \n",
                  service_nbr);
    #endif

    #if DEBUG_RTI_MGR
        printf("rti service:%d found and matched entry in criteria table\n",
              svc_msg_info->rti_svc_nbr);
        printf("printing retrieved values-\n");
        printf("service nbr:  service_type: rtiamb_actionname:  action:  \n");
        printf("%d      %s      %s      %d      \n",
              rtisvc_tblentry.service_nbr,
              rtisvc_tblentry.service_type,
              rtisvc_tblentry.rtiamb_action_name,
              rtisvc_tblentry.rtiamb_action);
        printf("criteria: %s\n", rtisvc_tblentry.criteria);
    #endif

        /* get status of Fedrtn Execution */
        rtimgr_update_fedrtn_state_status(FedExdb.fedex_state_status);

        /* compare fedex status with criteria
         if no match prohibit service execution
         */
        if (!rtimgr_criteria_compare(FedExdb.fedex_state_status,
                                     rtisvc_tblentry.criteria))
        {
            printf("failed criteria compare\n");
            return(0);
        }

        /* otherwise, svc met criteria; process service */
        /* process rti ambassador service */
        svc_statistic=
            rtimgr_process_rtiamb_svc(rtisvc_tblentry.rtiamb_action,
                                     rtisvc_tblentry.rtiamb_action_name,
                                     svc_msg_info,
                                     event_msg_info_ptr);
    }
    else /* must be fed ambassador service */
    {
    #if DEBUG_RTI_MGR
        printf("processing fed amb svc\n");
    #endif
    }
}

```

```
#endif
    svc_statistic=
        rtimgr_process_fedamb_svc(svc_msg_info->rti_svc_nbr,
                                fedamb_svc_name,
                                svc_msg_info,
                                event_msg_info_ptr);
    }
    return((double) svc_statistic);
}

/* ----- DocHeading---*/
```

30 June 1999

```

static void rtimgr_retrieve_svctblinfo(RTI_EVENT_MSG_TYPE      *svc_msg_info,
                                       RTI_SERVICE_TBL_ENTRY_TYPE *current_rti_tblsvc)
/* ----- EndDocHead----- */
{
/* ----- DocComment----- */
/* rtimgr_retrieve_svctblinfo:
* lookup eventsvc in criteria_table, retrieve criteria and other items
*/

    RTI_SERVICE_TBL_ENTRY_TYPE  *rtisvc_tbl_ptr;
    int      i;

    #if 0
    printf("rtimgr_retrieve_svctblinfo-\n");
    #endif

    rtisvc_tbl_ptr = &service_criteria[0];

/*    printf("rtisvc_tbl_ptr start address:%x\n",rtisvc_tbl_ptr); */

/* lookup service_nbr in table and retrieve other values */
for (i=0; i < service_criteria_lines; i++)
{
    if (rtisvc_tbl_ptr->service_nbr == svc_msg_info->rti_svc_nbr)
    {
        current_rti_tblsvc->service_nbr = svc_msg_info->rti_svc_nbr;
        strcpy(current_rti_tblsvc->service_type, rtisvc_tbl_ptr->service_type);
        strcpy(current_rti_tblsvc->criteria , rtisvc_tbl_ptr->criteria);
        current_rti_tblsvc->rtiamb_action =rtisvc_tbl_ptr->rtiamb_action;
        strcpy(current_rti_tblsvc->rtiamb_action_name,
            rtisvc_tbl_ptr->rtiamb_action_name);
        current_rti_tblsvc->rtiamb_cpu_svc_time =rtisvc_tbl_ptr->rtiamb_cpu_svc_time;
        current_rti_tblsvc->fedamb_reaction =rtisvc_tbl_ptr->fedamb_reaction;
        current_rti_tblsvc->fedamb_cpu_svc_time =rtisvc_tbl_ptr->fedamb_cpu_svc_time;
    #if DEBUG_RTI_MGR
        printf("found svc nbr:%d in services table\n",
            svc_msg_info->rti_svc_nbr);
    #endif
        return;
    }
    rtisvc_tbl_ptr++;    /* bump to next table entry */
}

printf("ERROR: unable to retrieve a svc nbr:%d from services table \n",
    svc_msg_info->rti_svc_nbr);

/* hardwire values for now */
/*
    strcpy(current_rti_tblsvc->service_type,"RTI");
    strcpy(current_rti_tblsvc->criteria , "110000");
    current_rti_tblsvc->rtiamb_action = 2001;
    strcpy(current_rti_tblsvc->rtiamb_action_name, "Publish");
    current_rti_tblsvc->rtiamb_cpu_svc_time = 0.0001;
    current_rti_tblsvc->fedamb_reaction = 0;
    strcpy(current_rti_tblsvc->fedamb_reaction_name,"none");
    current_rti_tblsvc->fedamb_cpu_svc_time = 0.0001;
*/
}

/* ----- DocComment----- */
/* rtimgr_criteria_compare:
*/
/* ----- DocHeading----- */

```

30 June 1999

```
static int  rtimgr_criteria_compare(FEDEX_STATE_INFO fedex_state_status,
                                   char      *criteria_strg)
/* ----- EndDocHead---*/
{

    /* need compare here of the two info items */

    /* for now, always return true= they match */
    return(TRUE);

}

/* ----- DocComment---*/
/*  rtimgr_get_RTI_ambsvc_time:
*/
/* ----- DocHeading---*/
```

```

extern double rtmgr_get_RTI_ambsvc_time(int federate_nbr,
                                       int    RTIambsvc,
                                       int    nbr_federates)
/* ----- EndDocHead----- */
{
    double          svc_time=0.0;

/* ----- DocComment----- */
/* Complexity Factors and Formulae:
 * The following formulae are used in determining the time for a service to
 * complete
 * complete execution. Computational times result from accumulation of the
 * factors specific to the execution of each service. The factors most variable
 * are those that entail the setup of data and retrieval of FOM and FedEx
 * database information since the cost is directly dependent on the existing database
 * size and retrieval method. Moreover, the more items (e.g., nbr of federates,
 * regions, and object instance subscriptions) in the database, the longer the
 * the computation. Likewise, the type of search algorithm employed (e.g.,
 * hash table, linear search, or binary search) will also affect the cost of
 * the computation, since hash table searches into databases are quicker to
 * find an item than using a linear search. Our experiment assumes hash
 * table setup and searches.
 *
 * note: To makeup the total time for RTI actions/reactions, two additional
 * time values are also collected outside of this function, as follows:
 * - svc_criteria verification = constant time prior to service action.
 *   Time to verify conditions that must be met before the service can execute.
 *   E.g. Verify that certain Federation Execution modes are active, such as
 *   Federation exists, and federate is a member. Likewise, that other modes
 *   are inactive, such as Save In Process, or Restore In Process.
 * - network_IO = time for I/O operations prior to service action.
 *   Time for I/O from a service (output information such as an federate's
 *   assigned name or a response service) to be transmitted to another node.
 */

/* Complexity Factors:
 * A) constraints_verification = time to verify that the service will not exceed
 * the database limits that will be affected by this service.
 *   E.g. Verify that joining the federate, or adding a region will not exceed
 * the maximum federates or regions permitted in the federation according to the i
 * predefined FOM constraints.
 * B) lookup_in_fomdb = time to verify objclass is valid FOM class
 * C) lookup_in_fedexdb = time to verify objclass is currently published.
 * D) setup_fed_in_fedexdb = time to add federate to the FedExDB federates table.
 * E) setup_region_in_fedexdb = time to add region to the FedExDB regions table and
 * initialize the region entry values
 * F) setup_class_in_fedexdb = time to setup class in FedExDB and initialize values.
 * G) setup_instance_in_fedexdb = time to setup an instance in the FedExDB and
 * initialize the values.
 * H) compute_nbr_nodes = time to compute which nodes have subscribed to the class
 * and/or region of this instance. This is a factor of which search algorithm is
 * employed in the database searches.
 * I) setup_events = time to setup an event for a node and send it to the node
 * J) forward_to_sim_model = time to pass on the service info to the SIM Model,
 * i.e., time to wait in TSO queue before serviced.
 * K) start_LBTS = time to toggle color mode for Federation's LBTS controller to
 * start LBTS calculation
 * L) retrieve_local_LBTS_info = time to retrieve the LBTS local info from
 * the federate database and time to change its marker mode state and variable
 * info to the new white/red mode.
 * M) forward_LBTS_info = time to setup and send the LBTS info to the parent
 * federate.
 * N) accumulate_LBTS_info = time for LBTS controller to compute new LBTS totals

```

30 June 1999

```

* whenever it receives a federate's reported LBTS info.
* O) advance_LBTS = time to advance the local federate's LBTS to the newly granted
* LBTS value and release the TSO events accordingly to the SIM model.
*/

```

```

#ifdef DEBUG_RTI_MGR
    printf("rtimgr_get_RTI_ambsvc_time-\n");
#endif

switch(RTIambsvc) {
    case (RTI_CREATE_FEDEX):
        /* this svc only affects the FedExDB
           to check if FedEx Name already exists
           svc_time is constant each time
        */
        svc_time = statsmgr_constraints_verification;
        break;
    case (RTI_JOIN_FEDEX):
        /* this svc takes time to:
           * verify that fed can be added to fedex (< maxfeds)
           * add federate to the FedExDB federates table
           * and initialize the federate values
        */
        svc_time = (statsmgr_constraints_verification +
                    statsmgr_setup_fed_in_fedexdb(nbr_federates));
        break;
    case (RTI_CREATE_UPDATE_REGION):
        /* this svc takes time to:
           * verify region can be added to fedex (< max regions)
           * add region to the FedExDB regions table
           * and initialize the region entry values
        */
        svc_time = (statsmgr_constraints_verification +
                    statsmgr_setup_region_in_fedexdb(nbr_federates));
        break;
    case (RTI_PUBLISH_OBJCLSS):
        /* this svc takes time to:
           * verify objclass is valid FOM class,
           * then setup class in FedExdb and initialize values
        */
        svc_time = (statsmgr_constraints_verification +
                    statsmgr_lookup_in_fomdb(nbr_federates) +
                    statsmgr_setup_class_in_fedexdb(nbr_federates));
        break;
    case (RTI_SUBSCRIBE_OBJCLSS):
        /* this svc takes time to:
           * verify objclass is valid FOM class,
           * then setup class in FedExdb and initialize values
        */
        svc_time = (statsmgr_constraints_verification +
                    statsmgr_lookup_in_fomdb(nbr_federates) +
                    statsmgr_setup_class_in_fedexdb(nbr_federates));
        break;
    case (RTI_REGISTER_INST):
        /* this svc takes time to:
           * verify objclass is valid FOM class,
           * verify objclass is valid published class,
           * then setup instance in FedExdb and initialize values
        */
        svc_time = (statsmgr_constraints_verification +
                    statsmgr_lookup_in_fomdb(nbr_federates) +
                    statsmgr_lookup_in_fedexdb(nbr_federates) +
                    statsmgr_setup_instance_in_fedexdb(nbr_federates));
        break;
    case (RTI_DISCVR_SETUP):
        /* time to:compute all nodes to receive a discover */
        svc_time = (statsmgr_compute_nbr_nodes(nbr_federates) +

```

30 June 1999

```

        statsmgr_setup_events);
    break;
case (RTI_PUBLISH_INTCLSS):
    /* this svc takes time to:
     *   verify interact class is valid FOM class,
     *   then setup class in FedEXdb and initialize values
     */
    svc_time = (statsmgr_constraints_verification +
                statsmgr_lookup_in_fomdb(nbr_federates) +
                statsmgr_setup_class_in_fedexdb(nbr_federates));
    break;
case (RTI_SUBSCRIBE_INTCLSS):
    /* this svc takes time to:
     *   verify interact class is valid FOM class,
     *   then setup class in FedEXdb and initialize values
     */
    svc_time = (statsmgr_constraints_verification +
                statsmgr_lookup_in_fomdb(nbr_federates) +
                statsmgr_setup_class_in_fedexdb(nbr_federates));
    break;
case (RTI_SEND_INT):
    /* this svc takes time to:
     *   verify instance is valid FOM class
     *   verify instance is published class
     */
    svc_time = (statsmgr_constraints_verification +
                (statsmgr_lookup_in_fomdb(nbr_federates)) +
                (statsmgr_lookup_in_fedexdb(nbr_federates)));
    break;
case (RTI_RECEIVE_INT_SETUP):
    /* time to: compute all nodes to receive the interaction */
    svc_time = statsmgr_compute_nbr_nodes(nbr_federates) +
                statsmgr_setup_events;
    break;
case (RTI_UPDATE_ATTRIB):
    /* this svc takes time to:
     *   verify instance is valid published class,
     */
    svc_time = (statsmgr_constraints_verification +
                statsmgr_lookup_in_fedexdb(nbr_federates) );
    break;
case (RTI_REFLECT_SETUP):
    /* time to: compute all nodes to receive a reflect */
    svc_time = statsmgr_constraints_verification +
                (statsmgr_compute_nbr_nodes(nbr_federates) +
                statsmgr_setup_events);
    break;
case (RTI_TIME_ADV_RQST):
    /* Only perform advance when the Controller recvs the advance request */
    if (federate_nbr != FOMdb.LBTS_controller)
        svc_time = statsmgr_forward_LBTS_info;
    else /* time to: start the LBTS computation at the controller
         */
        svc_time = statsmgr_start_LBTS;
    break;
case (RTI_LBTS_QUERY_SETUP):
    /* time to: compute all nodes to receive a LBTS request/query */
    /* and create an event for each */
    svc_time = statsmgr_compute_nbr_nodes(nbr_federates) +
                statsmgr_setup_events;
    break;
case (RTI_RPTNG_FED_LBTS):
    /* time to: when acting as the controller,
     *   accumulate(compute) received values into totals and
     *   and when grant criteria met, send grant to federates
     *   OR when not acting as the controller (acting as intermediate node),
     *   forward (pass on) the LBTS values

```


30 June 1999

```

*/
if (federate_nbr == FOMdb.LBTS_controller)
    svc_time= statsmgr_accumulate_LBTS_info;
else
    svc_time= statsmgr_forward_LBTS_info;
break;
case (RTI_TIME_ADV_GRANT_SETUP):
    /* when grant is met-
    /* time to: compute all nodes to receive the time adv grant */
    /*           and create an event for each                */
    svc_time= statsmgr_compute_nbr_nodes(nbr_federates) +
               statsmgr_setup_events;
    break;
default:
    printf("\nrtingr_get_RTI_ambsvc_time-unable to find RTI amb service match \n",
           RTIambsvc);
    svc_time= 0.0001;
    svc_time = svc_time * nbr_federates;
}
#endif
return(svc_time);

}
/* ----- DocComment---*/
/* rtingr_get_Fed_ambsvc_time:
*   same as above,i.e. complexity description
*/
/* ----- DocHeading---*/

```

30 June 1999

```

extern double    rtimgr_get_Fed_ambsvc_time(int fed_ambsvc,
                                             int    nbr_federates)
/* ----- EndDocHead----- */
{
    double        svc_time = 0.0;

    #if DEBUG_RTI_MGR
    printf("rtimgr_get_Fed_ambsvc_time-\n");
    #endif

    /* ----- DocComment----- */
    /* db complexity formula:
    * time for this service to compute and setup data into databases
    * need function to compute time with the following
    * factors considered:
    *     1- cost of lookup is dependent on the existing database
    *        items like:
    *            #federates, #regions, #objects, subscriptions
    *            i.e., as more items in database, takes longer to compute.
    *     2- cost of search algorithm like:
    *            hash table, linear search, or binary search
    *            i.e., hashtable is quicker to find an item than using a linear search
    */

    switch(fed_ambsvc) {
        case (RTI_DISCVR_OBJ):
            svc_time= statsmgr_forward_to_sim_model(nbr_federates);
            break;
        case (RTI_REFLECT_ATTRIB):
            svc_time= statsmgr_forward_to_sim_model(nbr_federates); break;
        case (RTI_RECEIVE_INT):
            svc_time= statsmgr_forward_to_sim_model(nbr_federates); break;
        case (RTI_QUERY_FED_LBTS):
            svc_time= statsmgr_retrieve_local_LBTS_info;
            break;
        case (RTI_TIME_ADV_GRANT):
            svc_time= statsmgr_advance_LBTS +
                statsmgr_forward_to_sim_model(nbr_federates);
            break;
        case (RTI_INITIATE_FED_SAVE):
            svc_time= statsmgr_forward_to_sim_model(nbr_federates); break;
        case (RTI_PRVD_ATTRIB_VALS):
            svc_time= statsmgr_forward_to_sim_model(nbr_federates); break;
        default:    printf("unable to find FED amb service match \n",
                        fed_ambsvc);
    }

    #if DEBUG_RTI_MGR
    printf("service: %d nbr_federates:%d svc_time:%f \n",
        fed_ambsvc, nbr_federates, svc_time);
    #endif

    return(svc_time);
}

/* ----- DocComment----- */
/* rtimgr_clear:
* Clear all values for an interval run, in preperation for the next run.
*/
/* ----- DocHeading----- */

```

30 June 1999

```
extern void rtimgr_clear()
/* ----- EndDocHead---*/
{

printf("rtimgr_clear- clearing out totals for this run \n");
    statmgr_clear_accum_totals();

printf("clearing out LBTS information \n");
    tm_clear_LBTS_info();

printf("clearing out reduction network info \n");
    rtimgr_clear_reduction_network_info();


}

/* ----- DocComment---*/
/* rtimgr_final_cleanup:
 * Print out the status of the system prior to cleanup.
 * Then delete all allocated objects created during execution.
 */
/* ----- DocHeading---*/
```

30 June 1999

```

extern void rtimgr_final_cleanup()
/* ----- EndDocHead----- */
{
    int i;
    REGIONS_LIST_TYPE      *regions_ptr, *last_region;
    SUBSCRIBED_INFO_TYPE    *region_subscribers, *last=NULL;
    SUBSCRIBED_INFO_TYPE    *region_interact_subscribers;
    FILE                    *outfile;

    outfile = fopen("hla_cpm_final_profile.out", "w" );

    if (outfile== NULL)
    { printf("rtimgr_cleanup:unable to open hla_cpm_final_profile file \n");
      exit(-1);
    }

    /* print out fed amb svcs */
    fprintf(outfile, "\n\n Fed Amb Svcs:%d \n", nbr_fedamb_svcs);
    fprintf(outfile, "Svc Nbr:   Svc Action: \n");
    for (i=0; i<nbr_fedamb_svcs; i++)
    {
        fprintf(outfile, "%d          %s \n",
            fedamb_svcs[i].fedamb_reaction,
            fedamb_svcs[i].fedamb_reaction_name);
    }

    /* print out the Fed Ex Db */
    fprintf(outfile, "\nFED EX Database \n");
    fprintf(outfile, "fedrtn_name:%d\n", FedExdb.fedrtn_name);
    fprintf(outfile, "nbr_member_federates:%d\n", FedExdb.nbr_member_federates);
    fprintf(outfile, "\nfedrtn_objclasses-%d\n", FOMdb.nbr_fedrtn_objclasses);
    fprintf(outfile, "class nbr      owner_federate   published      nbr_subscribed_federates\n");
    for (i=0; i<FOMdb.nbr_fedrtn_objclasses; i++)
    {
        fprintf(outfile, "%d          %d          %d          %d \n",
            i,
            FedExdb.fedrtn_objclasses[i].owner_federate,
            FedExdb.fedrtn_objclasses[i].published,
            FedExdb.fedrtn_objclasses[i].nbr_subscribed_federates);
    }
    fprintf(outfile, "\nfedrtn_interact_classes-%d\n", FOMdb.nbr_fedrtn_interact_classes);
    fprintf(outfile, "class nbr      owner_federate   published      nbr_subscribed_federates\n");
    for (i=0; i<FOMdb.nbr_fedrtn_interact_classes; i++)
    {
        fprintf(outfile, "%d          %d          %d          %d \n",
            i,
            FedExdb.fedrtn_interact_classes[i].owner_federate,
            FedExdb.fedrtn_interact_classes[i].published,
            FedExdb.fedrtn_interact_classes[i].nbr_subscribed_federates);
    }

    fprintf(outfile, "\nfederates:%d\n\n", FedExdb.nbr_member_federates);

    fprintf(outfile, "\nnbr_fedrtn_obj_instances-%d\n", FedExdb.nbr_fedrtn_obj_instances);
    fprintf(outfile, "obj_instance:  owner_federate: objclass_nbr:   \n");
    for (i=0; i <= FedExdb.nbr_fedrtn_obj_instances; i++)
    {
        fprintf(outfile, "%d          %d          %d \n", i,
            FedExdb.fedrtn_obj_instances[i].owner_federate,
            FedExdb.fedrtn_obj_instances[i].objclass_nbr);
        fprintf(outfile, "          assoc_regions: addr: nxt: \n");
        for (regions_ptr=FedExdb.fedrtn_obj_instances[i].associated_regions;
            regions_ptr!=NULL;
            )
        {
            fprintf(outfile, "          %d          %x          %x\n",

```

```

        regions_ptr->region_nbr, regions_ptr, regions_ptr->next);
    last_region = regions_ptr;
    regions_ptr=regions_ptr->next;
    free(last_region);
}
}
fprintf(outfile, "\nnbr fedrtn_interact_instances-%d\n",
        FedExdb.nbr_fedrtn_interact_instances);

fprintf(outfile, "\nnbr_fedrtn_regions-%d\n", FedExdb.nbr_fedrtn_regions);

fprintf(outfile, "region:          \n");
for (i=0; i <= FedExdb.nbr_fedrtn_regions; i++)
{
    fprintf(outfile, "%d \n", i);
    for (region_subscribers = FedExdb.fedrtn_regions[i].subscribed_objects;
        region_subscribers != NULL; )
    {
        fprintf(outfile, "    object node:%x\n", region_subscribers);
        fprintf(outfile, "    subscriber info- federate: %d objclass: %d nxt:%x\n",
            region_subscribers->federate_name,
            region_subscribers->class_nbr,
            region_subscribers->next);
        last = region_subscribers;
        region_subscribers=region_subscribers->next;
        free(last);
    }
    fprintf(outfile, "\n");
    for (region_interact_subscribers = FedExdb.fedrtn_regions[i].subscribed_interactions;
        region_interact_subscribers != NULL; )
    {
        fprintf(outfile, "    interact node:%x\n", region_interact_subscribers);
        fprintf(outfile, "    subscriber info- federate: %d int_class: %d nxt:%x\n",
            region_interact_subscribers->federate_name,
            region_interact_subscribers->class_nbr,
            region_interact_subscribers->next);
        last = region_interact_subscribers;
        region_interact_subscribers = region_interact_subscribers->next;
        free(last);
    }
}
fprintf(outfile, "\nnbr_save_names-%d\n", FedExdb.nbr_save_names);

#if REPORT_FEDERATE_PROCESSING
    for (i=1; i <= MAX_NBR_FEDERATES; i++)
        fclose(rti_svcstats_outfiles[i]);

#endif

    statsmgr_cleanup();

    fclose(outfile);
}

```

```

/* file: stats_manager.c */

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <math.h>

#include "rti_services.h"
#include "statsmgr.h"
#include "rti.h"

extern double statsmgr_norm_distrib(int nbr_federates);

#define MAX_TBL_ENTRIES 1000
#define DEBUG_STATS 0

/* global variables */
/* statistics_table- all buckets for accumulating svcs totals */
STATS_DATA_TYPE statistics_table[MAX_TBL_ENTRIES];
static int statsmgr_nbr_table_entries = 0;
static int stats_interval_run_nbr = 1;
FILE *stats_outfile;

/* reftables - references to the specific federate's statistic bucket by svc type */
STATISTIC_CPM_TYPE create_reftable[MAX_NBR_FEDERATES+1];
STATISTIC_CPM_TYPE join_reftable[MAX_NBR_FEDERATES+1];
STATISTIC_CPM_TYPE pubobj_reftable[MAX_NBR_FEDERATES+1];
STATISTIC_CPM_TYPE subobj_reftable[MAX_NBR_FEDERATES+1];
STATISTIC_CPM_TYPE register_reftable[MAX_NBR_FEDERATES+1];
STATISTIC_CPM_TYPE discover_reftable[MAX_NBR_FEDERATES+1];
STATISTIC_CPM_TYPE update_reftable[MAX_NBR_FEDERATES+1];
STATISTIC_CPM_TYPE reflect_reftable[MAX_NBR_FEDERATES+1];
STATISTIC_CPM_TYPE savedone_reftable[MAX_NBR_FEDERATES+1];
STATISTIC_CPM_TYPE pubint_reftable[MAX_NBR_FEDERATES+1];
STATISTIC_CPM_TYPE subint_reftable[MAX_NBR_FEDERATES+1];
STATISTIC_CPM_TYPE sendint_reftable[MAX_NBR_FEDERATES+1];
STATISTIC_CPM_TYPE region_create_reftable[MAX_NBR_FEDERATES+1];
STATISTIC_CPM_TYPE reqst_timeadv_reftable[MAX_NBR_FEDERATES+1];
STATISTIC_CPM_TYPE rpt_lbts_reftable[MAX_NBR_FEDERATES+1];
STATISTIC_CPM_TYPE start_fedsave_reftable[MAX_NBR_FEDERATES+1];
STATISTIC_CPM_TYPE recvint_reftable[MAX_NBR_FEDERATES+1];
STATISTIC_CPM_TYPE query_fedlbts_reftable[MAX_NBR_FEDERATES+1];
STATISTIC_CPM_TYPE grant_timeadv_reftable[MAX_NBR_FEDERATES+1];
STATISTIC_CPM_TYPE misc_reftable[MAX_NBR_FEDERATES+1];

/* ----- DocHeading -----*/

```

```
extern void statsmgr_init_statruns_file()
/* ----- EndDocHead---*/
{
/* ----- DocComment ---*/
/* statsmgr_init_statruns_file()
*   Create and open the Statistics File which reports the
*   time durations of services for the RTI.
*/

    int                i,j;
    STATS_DATA_TYPE    *entry;

#ifdef DEBUG_STATS
    printf("statsmgr_init_statruns_file-\n");
#endif

    stats_outfile= fopen("hla_rti_statistics.out","a");

    if (stats_outfile == NULL)
    {
        printf("Error- unable to open statistics output file \n");
        exit;
    }
    printf("creating stats_outfile:hla_rti_statistics.out \n");
}

/* ----- DocHeading---*/
```

30 June 1999

```

extern int initialize_statistic(STATISTIC_CPM_TYPE *stat_entry,
                                char *svcname,
                                int federate_nbr )

/* ----- EndDocHead-----*/
{
/* ----- DocComment-----*/
/* initialize_statistic:
* assign a tag index into array for this name
*/
    char          strg2[6];
    int           stats_tbl_index;

    stats_tbl_index = statsmgr_nbr_table_entries ;

    strcpy(stat_entry->stat_label_name, svcname) ; /* , MAX_SVC_NAME_LEN-2); */
    sprintf(strg2,"%02d",federate_nbr);
    strcat(stat_entry->stat_label_name, strg2 );

#ifdef DEBUG_STATS
    printf("new stat_label_name:%s \n",
           stat_entry->stat_label_name);
#endif

    stat_entry->statsarray_index = stats_tbl_index;
#ifdef DEBUG_STATS
    printf(" with stats_tbl_index:%d \n", stats_tbl_index);
#endif
    strncpy(statistics_table[stats_tbl_index].stat_label_name,
           stat_entry->stat_label_name, MAX_SVC_NAME_LEN);

    /* initialize value to zero */
    statistics_table[stats_tbl_index].samples_count =0;
    statistics_table[stats_tbl_index].samples_totals[MIN_STAT]=
        statistics_table[stats_tbl_index].samples_totals[MAX_STAT] =
        statistics_table[stats_tbl_index].samples_totals[MEAN_STAT] =0.0;

    statistics_table[stats_tbl_index].replicates_count = 0;
    statistics_table[stats_tbl_index].replicates_totals[MIN_STAT]=
        statistics_table[stats_tbl_index].replicates_totals[MAX_STAT]=
        statistics_table[stats_tbl_index].replicates_totals[MEAN_STAT]=
        statistics_table[stats_tbl_index].replicates_totals[VARIANCE_STAT]=
        statistics_table[stats_tbl_index].replicates_totals[SMPL_CUM_TOTL]=0.0;

    statsmgr_nbr_table_entries += 1;
    return(stats_tbl_index); /* so the calling routine can remember */
}

/* ----- DocComment-----*/
/* statsmgr_setup_stats_tables
*/
/* ----- DocHeading-----*/

```


30 June 1999

```

extern void statsmgr_setup_stats_tables()
/* ----- EndDocHead----- */
{
    int    i;        /* federate nbr */
    int    j=0;       /* statistics table reference index */

    #if DEBUG_STATS
    printf("statsmgr_setup_stats_tables-\n");
    #endif

    /* setup the statistic output table info */
    initialize_statistic(&misc_reftable[0], "MISC", 0 );
    initialize_statistic(&create_reftable[1], "CREATE_FED", 1 );

    for (i=1; i <= MAX_NBR_FEDERATES && (j+18) < MAX_TBL_ENTRIES; i++)
    {
        /* setup the statistic output table info */
        /* j is just used as a dummy */
        j = initialize_statistic(&join_reftable[i],      "JOIN_FED", i );
        j = initialize_statistic(&pubobj_reftable[i],     "PUBOBJ_FED", i );
        j = initialize_statistic(&subobj_reftable[i],     "SUBOBJ_FED", i );
        j = initialize_statistic(&register_reftable[i],   "REGSTR_FED", i );
        j = initialize_statistic(&discover_reftable[i],   "DISCVR_FED", i );
        j = initialize_statistic(&update_reftable[i],     "UPDATE_FED", i );
        j = initialize_statistic(&reflect_reftable[i],    "REFLCT_FED", i );
        j = initialize_statistic(&savdone_reftable[i],    "SAVEDONE_FED", i );
        j = initialize_statistic(&pubint_reftable[i],     "PUBINT_FED", i );
        j = initialize_statistic(&subint_reftable[i],     "SUBINT_FED", i );
        j = initialize_statistic(&sendint_reftable[i],    "SENTINT_FED", i );
        j = initialize_statistic(&region_create_reftable[i], "REGCR_FED", i );
        j = initialize_statistic(&reqst_timeadv_reftable[i], "RQSTADV_FED", i );
        j = initialize_statistic(&rpt_lbts_reftable[i],    "RPTLBTS_FED", i );
        j = initialize_statistic(&start_fedsave_reftable[i], "STRTSAVE_FED", i );
        j = initialize_statistic(&rcvint_reftable[i],     "RCVINT_FED", i );
        j = initialize_statistic(&query_fedlbts_reftable[i], "QRYLBTS_FED", i );
        j = initialize_statistic(&grant_timeadv_reftable[i], "GRNTADV_FED", i );

    }

    /* print out the reftables, for reference later in verifications only */
}

/* ----- DocComment----- */
/*    lookup_in_fomdb
*/
/* ----- DocHeading----- */

```

30 June 1999

```
extern double statsmgr_lookup_in_fomdb(int nbr_federates)
/* ----- EndDocHead---*/
{
    double tmp;

    tmp =statsmgr_norm_distrib(nbr_federates);

    return(tmp);
}

/* ----- DocComment---*/
/*    lookup_in_fedexdb
*/
/* ----- DocHeading---*/
```

30 June 1999

```
extern double statsmgr_lookup_in_fedexdb(int nbr_federates)
/* ----- EndDocHead---*/
{
    return(statsmgr_norm_distrib(nbr_federates));
}

/* ----- DocComment---*/
/*      setup_fed_in_fedexdb
*/
/* ----- DocHeading---*/
```

```
extern double statsmgr_setup_fed_in_fedexdb(int nbr_federates)
/* ----- EndDocHead---*/
{
    return(statsmgr_norm_distrib(nbr_federates));
}

/* ----- DocComment---*/
/*      setup_region_in_fedexdb
*/
/* ----- DocHeading---*/
```

30 June 1999

```
extern double statsmgr_setup_region_in_fedexdb(int nbr_federates)
/* ----- EndDocHead---*/
{
    return(statsmgr_norm_distrib(nbr_federates));
}

/* ----- DocComment---*/
/*      setup_class_in_fedexdb
*/
/* ----- DocHeading---*/
```

30 June 1999

```
extern double statsmgr_setup_class_in_fedexdb(int nbr_federates)
/* ----- EndDocHead---*/
{
    double tmp;

    tmp = statsmgr_norm_distrib(nbr_federates);

    return(tmp);
}

/* ----- DocComment---*/
/*      setup_instance_in_fedexdb
*/
/* ----- DocHeading---*/
```

30 June 1999

```
extern double statsmgr_setup_instance_in_fedexdb(int nbr_federates)
/* ----- EndDocHead---*/
{
    return(statsmgr_norm_distrib(nbr_federates));
}

/* ----- DocComment---*/
/* compute_nbr_nodes:
*/
/* ----- DocHeading---*/
```

30 June 1999

```
extern double statsmgr_compute_nbr_nodes(int nbr_federates)
/* ----- EndDocHead---*/
{
    return(statsmgr_norm_distrib(nbr_federates));
}

/* ----- DocComment---*/
/* statsmgr_forward_to_sim_model:
*/
/* ----- DocHeading---*/
```



```
extern double statsmgr_forward_to_sim_model(int  nbr_federates)
/* ----- EndDocHead---*/
{
    return(statsmgr_norm_distrib(nbr_federates));
}

/* ----- DocComment---*/
/* statsmgr_norm_distrib:
 * - normal distribution as nbr_federates increases???
 */
/* ----- DocHeading---*/
```

30 June 1999

```

extern double statsmgr_norm_distrib(int nbr_federates)
/* ----- EndDocHead---*/
{
    double stat;

    /* possible types of searches */
    /*
    bubble sort = slow      = n squared
    binary tree search = middle = n log n
    hash table = fastest    = usually only 1 search
    */

    /* hardwire for now */
    stat = nbr_federates * 0.0001;

    #if 0
        switch (nbr_federates)
        {
            case(1):
                stat = (0.0001);
                break;
            case(2):
                stat = (0.0002);
                break;
            case(3):
                stat = (0.0003);
                break;
            default:
                printf("statsmgr_norm_distrib- nbr federates not setup\n");
                stat = (0.000001);
        }
    #endif

    return(stat);
}

/* ----- DocComment---*/
/* statsmgr_get_statsarray_index:
*/
/* ----- DocHeading---*/

```

30 June 1999

```

extern int statsmgr_get_statsarray_index(int    action,
                                         int      federate_nbr)
/* ----- EndDocHead----- */
{
    #if DEBUG_STATS
        printf("\nstatsmgr_get_statsarray_index- action:%d federate:%d\n",
              action, federate_nbr);
    #endif

    switch(action) {
        case(RTI_CREATE_FEDEX):
            return(create_reftable[federate_nbr].statsarray_index);
        case(RTI_JOIN_FEDEX):
            return(join_reftable[federate_nbr].statsarray_index);
        case(RTI_FED_SAVE_COMPLETE):
            return(savedone_reftable[federate_nbr].statsarray_index);
        case(RTI_PUBLISH_OBJCLSS):
            return(pubobj_reftable[federate_nbr].statsarray_index);
        case(RTI_PUBLISH_INTCLSS):
            return(pubint_reftable[federate_nbr].statsarray_index);
        case(RTI_SUBSCRIBE_INTCLSS):
            return(subint_reftable[federate_nbr].statsarray_index);
        case(RTI_SUBSCRIBE_OBJCLSS):
            return(subobj_reftable[federate_nbr].statsarray_index);
        case(RTI_REGISTER_INST):
            return(register_reftable[federate_nbr].statsarray_index);
        case(RTI_UPDATE_ATTRIB):
            return(update_reftable[federate_nbr].statsarray_index);
        case(RTI_SEND_INT):
            return(sendint_reftable[federate_nbr].statsarray_index);
        case(RTI_RQST_ATTRIB_VALS):
            return(0);
        case(RTI_CREATE_UPDATE_REGION):
            return(region_create_reftable[federate_nbr].statsarray_index);
        case(RTI_ASSOC_REGION):
            return(0);
        case(RTI_TIME_ADV_RQST):
            return(reqst_timeadv_reftable[federate_nbr].statsarray_index);
        case(RTI_RPTNG_FED_LBTS):
            return(rpt_lbts_reftable[federate_nbr].statsarray_index);
        case(RTI_INITIATE_FED_SAVE):
            return(start_fedsave_reftable[federate_nbr].statsarray_index);
        case(RTI_DISCVR_OBJ):
            return(discover_reftable[federate_nbr].statsarray_index);
        case(RTI_REFLECT_ATTRIB):
            return(reflect_reftable[federate_nbr].statsarray_index);
        case(RTI_RECEIVE_INT):
            return(recvint_reftable[federate_nbr].statsarray_index);
        case(RTI_QUERY_FED_LBTS):
            return(query_fedlbts_reftable[federate_nbr].statsarray_index);
        case(RTI_TIME_ADV_GRANT):
            return(grant_timeadv_reftable[federate_nbr].statsarray_index);
        default: printf("no match found\n");
                return(0);
    }
}

/* ----- DocHeading----- */

```

```

extern void statsmgr_accum_totals(double      replicate_time,
                                   STATS_DATA_TYPE *statstbl_entry)
/* ----- EndDocHead----- */
{
    double      second_term= 0.0;
/* ----- DocComment----- */
/* statsmgr_accum_totals:
 *   Accumulate the sample values into a replicate.
 *   Whenever enough samples are accumulated to form a replicate,
 *   perform the statistical formulas for mean, variance and std deviation,
 *   and preserve the results.
 *   Only report the totals when complete.
 */

#if DEBUG_STATS
printf("statsmgr_accum_totals for replicate_time:%8.6f\n",
      replicate_time);
#endif

#if DEBUG_STATS
/*
printf("current tbl values-\n");
printf("replicates counted:%d total time:%f mean time:%f\n",
      statstbl_entry->replicates_count,
      statstbl_entry->replicate_totals[SMPL_CUM_TOTL],
      statstbl_entry->replicate_totals[MEAN_STAT]);
*/

printf("repl sample:   nbr samples: totaltime:   mean time:   min:   max:   var sec
term:   cum sum:   variance:\n");
#endif

    /* mean = 1/count * sumof(sample_times) */
    (statstbl_entry->replicates_count)++;
    statstbl_entry->replicate_totals[SMPL_CUM_TOTL] +=
        replicate_time;

    statstbl_entry->replicate_totals[MEAN_STAT]=
        statstbl_entry->replicate_totals[SMPL_CUM_TOTL] / statstbl_entry-
>replicates_count;

#if DEBUG_STATS
    /* sample_time, nbr samples, cum total, mean */
    printf(" %8.6f %3d %8.6f %8.6f",
        replicate_time,
        statstbl_entry->replicates_count,
        statstbl_entry->replicate_totals[SMPL_CUM_TOTL],
        statstbl_entry->replicate_totals[MEAN_STAT]);
#endif

    if (statstbl_entry->replicate_totals[MIN_STAT] > replicate_time)
        statstbl_entry->replicate_totals[MIN_STAT] = replicate_time;
    if (statstbl_entry->replicate_totals[MAX_STAT] < replicate_time)
        statstbl_entry->replicate_totals[MAX_STAT] = replicate_time;

    /* variance = 1/(count-1) * (sumof( squared(this_sample - mean))) */
    if (statstbl_entry->replicates_count < 2)
    {
        statstbl_entry->replicate_totals[MIN_STAT] = replicate_time;
        statstbl_entry->replicate_totals[MAX_STAT] = replicate_time;
        statstbl_entry->replicate_totals[VARIANCE_STAT] = 0.0;
    }
    else
    {
        second_term = replicate_time - statstbl_entry->replicate_totals[MEAN_STAT];
    }
}

```

30 June 1999

```
second_term *= second_term;
statstbl_entry->replicates_totals[VAR_CUM_TOTL_SEC_TERM] += second_term;
statstbl_entry->replicates_totals[VARIANCE_STAT] =
    statstbl_entry->replicates_totals[VAR_CUM_TOTL_SEC_TERM] /
    (statstbl_entry->replicates_count - 1);
}
#endif
/* min and max */
printf("    %6.4f    %6.4f ",
    statstbl_entry->replicates_totals[MIN_STAT],
    statstbl_entry->replicates_totals[MAX_STAT]);

/* second_term, cum total, and variance */
printf("    %8.6f    %8.6f    %8.6f \n",
    second_term,
    statstbl_entry->replicates_totals[VAR_CUM_TOTL_SEC_TERM],
    statstbl_entry->replicates_totals[VARIANCE_STAT]);
#endif
}

/* ----- DocComment---*/
/* statsmgr_collect_statistic()
 * Add this sample's values to the running totals accumulated so far for a replicate.
 * Every specified nbr of samples (NBR_SAMPLES_IN_REPLICATE) = 1 replicate.
 */
/* ----- DocHeading---*/
```

```

extern void statsmgr_collect_statistic(int    stat_entry_index,
                                       double   sample_time)
/* ----- EndDocHead----- */
{
    STATS_DATA_TYPE    *statstbl_entry;
    int                i;

    #if DEBUG_STATS
    printf("statsmgr_collect_statistic for stat_entry_index:%d sample_time:%8.6f\n",
          stat_entry_index, sample_time);
    #endif

    statstbl_entry = &statistics_table[stat_entry_index];

    statstbl_entry->samples[statstbl_entry->samples_count] = sample_time;
    statstbl_entry->samples_count++;
    #if DEBUG_STATS
    printf("samples_count:%d\n", statstbl_entry->samples_count);
    #endif

    if (statstbl_entry->samples_count < NBR_SAMPLES_IN_REPLICATE)
    {
        return;
    }
    else
    {
        statstbl_entry->samples_count = 0;

        /* compute cumulative total for sample times */
        statstbl_entry->samples_totals[SMPL_CUM_TOTL]=0.0;
        for (i=0; i<NBR_SAMPLES_IN_REPLICATE; i++)
        {
            statstbl_entry->samples_totals[SMPL_CUM_TOTL] +=
                statstbl_entry->samples[i];
        }
        /* compute mean for samples */
        statstbl_entry->samples_totals[MEAN_STAT] =
            statstbl_entry->samples_totals[SMPL_CUM_TOTL] / NBR_SAMPLES_IN_REPLICATE;

        statsmgr_accum_totals(statstbl_entry->samples_totals[MEAN_STAT],
                              statstbl_entry);
    }

    /* ----- DocComment----- */
    /* statsmgr_print_accum_totals()
    */
    /* ----- DocHeading----- */

```

```

extern void statsmgr_print_accum_totals()
/* ----- EndDocHead----- */
{
    int                i,j;
    STATS_DATA_TYPE    *entry;

#ifdef DEBUG_STATS
    printf("statsmgr_print_accum_totals-\n");
#endif

    if (stats_outfile == NULL)
    {
        printf("Error- unable to add to statistics output file \n");
        exit;
    }

    printf("adding to stats_outfile \n");
    fprintf(stats_outfile,
        "CPM RTI STATISTICS TABLES:          Interval Run#: %d\n", stats_interval_run_nbr);
    fprintf(stats_outfile,
        "  INDX:,      STAT_TAG:,rep, #SAMPLES:  MIN:      MAX:      MEAN:  VAR:      STD:
CONVERGED: \n");
    for (i=0; i< statsmgr_nbr_table_entries; i++)
    {
        entry = &statistics_table[i];
        /* if incomplete sample replicate, make one now */
        if (entry->samples_count > 0)
        {
            for (j=0; j<entry->samples_count; j++)
            {
                entry->samples_totals[SMPL_CUM_TOTL] +=
                    entry->samples[j];
            }
            entry->samples_totals[MEAN_STAT] =
                entry->samples_totals[SMPL_CUM_TOTL] / entry->samples_count;
            statsmgr_accum_totals(entry->samples_totals[MEAN_STAT],
                                entry);
            entry->samples_count=0;
        }
        fprintf(stats_outfile,
            " %3d,%16s,Rep%02d,%5d,  %8.6f,%8.6f,%8.6f,%8.6f,%8.6f, %3d\n",
            /* "%15s %5d %g %g %g %g %g %2d\n", */
            i,
            entry->stat_label_name, stats_interval_run_nbr,
            entry->replicates_count,
            entry->replicates_totals[MIN_STAT],
            entry->replicates_totals[MAX_STAT],
            entry->replicates_totals[MEAN_STAT],
            entry->replicates_totals[VARIANCE_STAT],
            entry->d,
            entry->converger);
    }
    fprintf(stats_outfile, "\n\n");

    stats_interval_run_nbr += 1;
}

/* ----- DocComment----- */
/* statsmgr_clear_accum_totals()
 * clear the statistical totals for this interval
 */
/* ----- DocHeading----- */

```

30 June 1999

extern void statsmgr_clear_accum_totals()

```

/* ----- EndDocHead---*/
{
    int                i;
    STATS_DATA_TYPE    *entry;

    for (i=0; i< statsmgr_nbr_table_entries; i++)
    {
        entry = &statistics_table[i];
        entry->samples_count =
            entry->replicates_count =
            entry->converger = 0;

        entry->samples_totals[MEAN_STAT] =
            entry->samples_totals[MIN_STAT] =
            entry->samples_totals[MAX_STAT] =
            entry->samples_totals[SMPL_CUM_TOTL] =0.0;

        entry->replicates_totals[MIN_STAT] =
            entry->replicates_totals[MAX_STAT] =
            entry->replicates_totals[SMPL_CUM_TOTL] =
            entry->replicates_totals[MEAN_STAT] =
            entry->replicates_totals[VAR_CUM_TOTL_SEC_TERM] =
            entry->replicates_totals[VARIANCE_STAT] =0.0;

        entry->d = 0.0;
    }
}

/* ----- DocComment---*/
/* statsmgr_cleanup:
*/
/* ----- DocHeading---*/

```


30 June 1999

```
extern void statsmgr_cleanup()
/* ----- EndDocHead---*/
{

    fclose(stats_outfile);

}

/* ----- DocHeading ---*/
```

30 June 1999

/* file: time_mgmt_svc.c */

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

```
#include "serv_crit.h"
#include "event.h"
#include "rti.h"
#include "rtimgr.h"
#include "rti_services.h"
#include "proto.h"
#define Tmp 0
```

```
extern double tm_time_adv_grant_setup(int federate_nbr,
                                       int *nbr_federates,
                                       EVENT_MESSAGE_TYPE *event_msg_info_ptr);
```

```
extern void tm_forward_LBTS_info(int from_federate_nbr,
                                  int fwd_federate_nbr,
                                  EVENT_MESSAGE_TYPE *event_msg_info_ptr,
                                  double statistic);
```

/* ----- DocHeading ---*/

30 June 1999

```

extern double tm_LBTS_requests_setup(int federate_nbr,
                                     EVENT_MESSAGE_TYPE *event_msg_info_ptr,
                                     int *nbr_federates)
/* ----- EndDocHead----- */
{
/* ----- DocComment ----*/
/* tm_LBTS_requests_setup:
 *   setup of RTI_QUERY_FED_LBTS event to each federate
 */

    int i;
    FEDERATE_DESTINS_TYPE *dest_element, *destinations_list;
    double statistic=0.0;
    EVENT_MESSAGE_TYPE *aNewM;
    char str[12];
    if (TmP) {printf("tm_LBTS_requests_setup- federate_nbr:%d\n",federate_nbr);
    }

    *nbr_federates = FedExdb.nbr_member_federates;

    statistic =
        rtimgr_get_RTI_ambsvc_time(federate_nbr, RTI_LBTS_QUERY_SETUP, *nbr_federates);

    event_msg_info_ptr->Rti.rti_svc_nbr = RTI_QUERY_FED_LBTS;
    eventmgr_change_processing_mode(event_msg_info_ptr, RTI_PROCESSING);
    destinations_list = event_msg_info_ptr->destinations_list;

    for (i=1; i <= *nbr_federates; i++)
    {
        if (i == FOMdb.LBTS_controller)
        {
            if (TmP) {printf("adding federate:%d rti_event to local rtiq statistic %8.5f \n", i,
statistic);
            }

            aNewM = c_Duplicate_Event_Message(event_msg_info_ptr);
            aNewM->Time.PhysicalTime = statistic + CurrentFederateTime( federate_nbr );
            // dest_element = om_create_destinations_element(i);
            //aNewM->destinations_list = dest_element;
            /* add event back to my local rti q */
            AddPriorityEvent(stdout, "InToRTI", aNewM);
            //gets(str);
            // continue;
        }
        /* otherwise, accumulate destinations for outq events */
        dest_element = om_create_destinations_element(i);
        dest_element->next = destinations_list;
        destinations_list = dest_element;
        if (TmP) { printf("added federate: %2d to destinations list\n", i);}
    }
    if (*nbr_federates > 1)
    {
        event_msg_info_ptr->destinations_list = destinations_list;
        /* send event as priority==TRUE for admin msgs */
        QueuesPrint(stdout,-5);
        if (TmP) {printf(" tm_LBTS_requests_setup statistic %8.5f Press Enter \n", statistic ) ;}
        // gets(str);
        iomgr_send_ioevent(event_msg_info_ptr, statistic, TRUE );
    }

    return(statistic);
}

/* ----- DocHeading-----*/

```

30 June 1999

```

extern double tm_time_adv_request(int federate_nbr,
                                  int fedrtn_time,
                                  int *nbr_federates,
                                  EVENT_MESSAGE_TYPE *event_msg_info_ptr)
/* ----- EndDocHead----- */
{
/* ----- DocComment----- */
/* tm_time_adv_request :
 * request from a federate to advance its LBTS
 */
    double      svc_statistic=0.0;
    int          color;

    svc_statistic = rtimgr_get_RTI_ambsvc_time(federate_nbr, RTI_TIME_ADV_RQST, 1);

    if (TMP) {printf("tm_time_adv_request: processing fed: %2d proposed fed time: %4d statistic
%8.4f\n",
                    federate_nbr, fedrtn_time, svc_statistic);
    }

    /* if LBTS controller, begin the LBTS process */
    if (federate_nbr == FOMdb.LBTS_controller)
    {
        eventmgr_change_processing_mode(event_msg_info_ptr,DONE_PROCESSING);

        if (FedExdb.fedrtn_time_mgmt_info.current_state == ACTIVE)
        {
            printf("error- already processing an advance request \n");
        }
        else
        {
            /* toggle the fedrtn mode odd/even to begin LBTS compute */
            color = StartLBTSCalculation();
            //printf("tm_time_adv_request: new fedrtn color-%d\n",color);

            FedExdb.fedrtn_time_mgmt_info.current_state = ACTIVE;
            FedExdb.fedrtn_time_mgmt_info.LBTS_proposed = fedrtn_time;

            /* setup request/queries to each federate for their LBTS info */

            svc_statistic +=
                tm_LBTS_requests_setup(federate_nbr, event_msg_info_ptr, nbr_federates);
            if (TMP) {printf("tm_time_adv_request: RequestSetup fed: %2d proposed fed time: %4d
statistic %8.4f\n",
                            federate_nbr, fedrtn_time, svc_statistic);
            }
        }
    }

    /* forward the request to the LBTS controller federate */
    else
    {
        tm_forward_LBTS_info(federate_nbr,
                            FOMdb.LBTS_controller,
                            event_msg_info_ptr,
                            svc_statistic);
    }

    return(svc_statistic);
}

/* ----- DocComment----- */
/*
 * all_my_subfederates_reported
 * flag TRUE when all federates have reported at least once
 * to their parent
 * Note: this is intended to keep track of which federate has reported
 * its initial msg counts after waiting for all its subordinate reportings.

```

30 June 1999

*/

/* ----- DocHeading---*/

30 June 1999

```
static int all_my_subfederates_reported(int federate_nbr)
{
    int i;
    LBTS_REDCTN_NETWORK_INFO *redctn_ntwk_federate_info;

    redctn_ntwk_federate_info =
        &FedExdb.federates[federate_nbr].reduction_network_info;
    for (i=0; i< redctn_ntwk_federate_info->nbr_children; i++)
    {
        if (!redctn_ntwk_federate_info->children_reported[i])
            return(FALSE);
    }
    return(TRUE);
}

/* ----- DocHeading---*/
```

30 June 1999

```

extern double tm_controller_LBTS_compute(int federate_nbr,
                                         int *nbr_federates,
                                         EVENT_MESSAGE_TYPE *event_msg_info_ptr)
/* ----- EndDocHead----- */
/* ----- DocComment----- */
/*
 * tm_controller_LBTS_compute: RTI_RPTNG_FED_LBTS
 * function entered when a federate reports its LBTS info of
 * LBTS time, rcvd and send msgs.
 * This function then adds the info to the global info for the fedrtn
 * and tests to see if computation is complete
 * before granting the advance request
 * The global virtual time (global LBTS) is madeup of:
 * 1. the smallest time stamp of any unprocessed event within
 * a federate at its cut point plus the federate's lookahead time
 * 2. the smallest time stamp of any transient message crossing the
 * cut from past to future.
 * where transient msgs are those white msgs recvd by a federate
 * when in red mode
 */
int i, child_nbr;
double svc_statistic=0.0;
RTI_EVENT_MSG_TYPE *rptng_federate_info;
LBTS_REDCNTN_NETWK_INFO *redctn_ntwk_federate_info;
int any_remaining ;
char str[12];

/* received LBTS report from a federate via the admin svc-
 * RTI_RPTNG_FED_LBTS, (this is not a RTI Spec svc).
 * RTI_RPTNG_RCV_LBTS, RTI_RPTNG_SND_LBTS, (these are not a RTI Spec svc).
 * Take the values and add them to totals so far
 */

rptng_federate_info = &event_msg_info_ptr->Rti;

if (Tmp) { printf("tm_controller_LBTS:LBTS_cur:%8.3f LBTS_pro:%8.3f",
                 FedExdb.fedrtn_time_mgmt_info.LBTS_current,
                 FedExdb.fedrtn_time_mgmt_info.LBTS_proposed );
printf(" fed:%2d orig:%2d prnt:%2d LBTS:%8.3f fedrtn:%8.3f",
       federate_nbr,
       rptng_federate_info->origin_fed_name,
       FOMdb.nodes[federate_nbr].parent,
       rptng_federate_info->LBTS_time,
       rptng_federate_info->fedrtn_time);

printf(" Report rcvd:%5d sent:%5d\n",
       rptng_federate_info->nbr_rcvd_msgs,
       rptng_federate_info->nbr_sent_msgs);
}
if (FedExdb.fedrtn_time_mgmt_info.current_state==INACTIVE)
{
    printf("ERROR- receiving msg reports when not in an Advance Request/LBTS computation.
%s\n",
        "...ignoring msg count report.");
    return(svc_statistic);
}

svc_statistic += rtimgr_get_RTI_ambsvc_time(federate_nbr, RTI_RPTNG_FED_LBTS,1);

/* default to done processing this event */
eventmgr_change_processing_mode(event_msg_info_ptr,DONE_PROCESSING);

redctn_ntwk_federate_info = &FedExdb.federates[federate_nbr].reduction_network_info;

/* When info is reported from a subordinate

```

30 June 1999

```

* within the reduction network, wait on compute
* until all subordinates have responded, then
* pass on info to superior node
*/
/* accumulate child reports of msgs */
redctn_ntwk_federate_info->total_rcvd_msgs += rptng_federate_info->nbr_rcvd_msgs;
redctn_ntwk_federate_info->total_sent_msgs += rptng_federate_info->nbr_sent_msgs;

/* save off the lowest LBTS of received messages :
* if best LBTS this far not setup, or this subordinate
* is reporting a new lowest LBTS,
* then make this LBTS time the new best LBTS.
*/
if ((!redctn_ntwk_federate_info->best_LBTS) ||
    (rptng_federate_info->LBTS_time < redctn_ntwk_federate_info->best_LBTS))
{
    redctn_ntwk_federate_info->best_LBTS= rptng_federate_info->LBTS_time;
    // will printf("new best LBTS:%f \n", redctn_ntwk_federate_info->best_LBTS);
}

/* Keep track of which subordinate has reported initial msg counts,
* but exclude straggler msgs (RTI_RPTNG_RCV_LBTS or RTI_RPTNG_SND_LBTS)
* from the initial msg count reportings
*/
if (rptng_federate_info->rti_svc_nbr== RTI_RPTNG_FED_LBTS)
{
    /* if this federate is the federate reporting,
    * then I am the federate rptng */
    if (federate_nbr == rptng_federate_info->origin_fed_name)
    { redctn_ntwk_federate_info->i_reported = TRUE; }
    else /* otherwise, I am a parent, flag which child is reporting */
    {
        for (i=0; i < redctn_ntwk_federate_info->nbr_children; i++)
        {
            child_nbr = redctn_ntwk_federate_info->children_names[i];
            if (child_nbr == rptng_federate_info->origin_fed_name){
                redctn_ntwk_federate_info->children_reported[i] = TRUE;
            }
        }
    }
}
if (federate_nbr != FOMdb.LBTS_controller)
{
    /* only forward the LBTS info when all subordinates and I have reported */
    if (!all_my_subfederates_reported(federate_nbr)
        || (!redctn_ntwk_federate_info->i_reported))
    {
        // will printf("waiting for all subs before reporting \n");
    }
    else
    {
        if (TMP) {printf("federate:%d all my subs reported statistic %8.3f \n", federate_nbr,
svc_statistic);
        }
        rptng_federate_info->nbr_rcvd_msgs = redctn_ntwk_federate_info->total_rcvd_msgs;
        rptng_federate_info->nbr_sent_msgs = redctn_ntwk_federate_info->total_sent_msgs;
        /* only keep and report the lowest LBTS from messages */
        rptng_federate_info->LBTS_time = redctn_ntwk_federate_info->best_LBTS;

        rptng_federate_info->origin_fed_name= federate_nbr;

        if (TMP) {printf("reporting msgs rcvd:%d sent:%d LBTS:%f \n",
            rptng_federate_info->nbr_rcvd_msgs,
            rptng_federate_info->nbr_sent_msgs,
            rptng_federate_info->LBTS_time);
        }
        tm_forward_LBTS_info(federate_nbr,
            FOMdb.nodes[federate_nbr].parent,
            event_msg_info_ptr,

```


30 June 1999

```

        svc_statistic    );
    /* reset for next advance */
    redctn_ntwk_federate_info->total_rcvd_msgs = redctn_ntwk_federate_info-
>total_sent_msgs = 0;
    redctn_ntwk_federate_info->sent_initial_counts = TRUE;
    } /* all subs have reported */
} /* when sub reporting is not the controller */
/* Otherwise, info is reported either:
 *   locally from controller or
 *   off the network from controller's immediate subordinate
 *   within the reduction network,
 *   add values to totals
 */
else
{
    /* only consider grant when all subordinates have reported */
    if (!all_my_subfederates_reported(federate_nbr) || (!redctn_ntwk_federate_info-
>i_reported)){
        if (TMp) {printf("waiting for all subs initial reports \n");}
    }
    else
    {
        any_remaining = AnyPotentialMessagesToReceiveWithOldColorTag();

        if (TMp) {printf("COLORAnyRemaining %2d    %5d received    %5d sent \n",
            any_remaining,
            redctn_ntwk_federate_info->total_rcvd_msgs,
            redctn_ntwk_federate_info->total_sent_msgs );
        }
        // gets(str) ;
        if (redctn_ntwk_federate_info->total_rcvd_msgs ==
            redctn_ntwk_federate_info->total_sent_msgs ||
            any_remaining == 0 )
        {
            if (redctn_ntwk_federate_info->best_LBTS <
                FedExdb.fedrtn_time_mgmt_info.LBTS_proposed)
            {
                if (TMp) {printf("advance proposed:%f changed to new lowest:%f\n",
                    FedExdb.fedrtn_time_mgmt_info.LBTS_proposed,
                    redctn_ntwk_federate_info->best_LBTS);
                }
                FedExdb.fedrtn_time_mgmt_info.LBTS_current = redctn_ntwk_federate_info-
>best_LBTS;
            }
            else
            {
                if (TMp) {printf("advancing to proposed time:%f \n",
                    FedExdb.fedrtn_time_mgmt_info.LBTS_proposed); }
                FedExdb.fedrtn_time_mgmt_info.LBTS_current =
                    FedExdb.fedrtn_time_mgmt_info.LBTS_proposed;
            }
            svc_statistic += tm_time_adv_grant_setup(federate_nbr, nbr_federates,
                event_msg_info_ptr);
            /* reset reporting counts and status */
            rtingr_clear_reduction_network_info();
            tm_clear_LBTS_info();
        } /* msgs equal */
        else /* not all federates have reported LBTS info yet */
        {
            if (TMp) {printf("controller reported, still accumulating totals. \n");}
        } /* all subs have reported */
    } /* controller federate reporting */

    return(svc_statistic);
}

```

30 June 1999

```
/* ----- DocComment---*/  
/*  tm_forward_LBTS_info  
*/  
/* ----- DocHeading---*/
```

30 June 1999

```

extern void tm_forward_LBTS_info(int      from_federate_nbr,
                                   int      fwd_federate_nbr,
                                   EVENT_MESSAGE_TYPE *event_msg_info_ptr,
                                   double     statistic )
/* ----- EndDocHead----- */
{
    RTI_EVENT_MSG_TYPE    *rti_info;
    FEDERATE_DESTINS_TYPE *dest_element;
    char str[12];
    if (TMP) {printf("tm_forward_LBTS_info for advance or report  time %7.3f stat %7.3f Press
Enter\n",
        event_msg_info_ptr->Time.PhysicalTime, statistic);
    }
    //gets(str);
    /* forward the event - RTI_RPTNG_FED_LBTS or Advance Request
    */
    rti_info = &event_msg_info_ptr->Rti;
    /* change the destination to report info to */
    dest_element = om_create_destinations_element(fwd_federate_nbr);
    event_msg_info_ptr->destinations_list= dest_element;

    if (TMP) {printf("added federate:%d to destinations list\n", fwd_federate_nbr);
    }
    /* change the reporting federate nbr */
    rti_info->federate_name = from_federate_nbr;

    eventmgr_change_processing_mode(event_msg_info_ptr,RTI_PROCESSING);

    iomgr_send_ioevent(event_msg_info_ptr, statistic, TRUE );

}
/* ----- DocComment----- */
/*
* tm_time_adv_grant_setup
*/
/* ----- DocHeading----- */

```

30 June 1999

```

extern double tm_time_adv_grant_setup(int    federate_nbr,
                                       int      *nbr_federates,
                                       EVENT_MESSAGE_TYPE *event_msg_info_ptr)
/* ----- EndDocHead----- */
{
    FEDERATE_DESTINS_TYPE *dest_element, *destinations_list;
    double statistic=0.0;
    int i;
    char str[12];

    if (Tmp) {printf("tm_time_adv_grant_setup \n"); }
    destinations_list= event_msg_info_ptr->destinations_list;
    for (i=1; i <= FedExdb.nbr_member_federates; i++)
    {
        dest_element = om_create_destinations_element(i);
        if (dest_element)
        {
            if (Tmp) {printf("adding federate:%d as new element:%x onto list:%x\n",
                            i, dest_element, destinations_list);
            }
            dest_element->next =destinations_list;
            destinations_list = dest_element;
        }
        else
            printf("error- unable to create new dest element\n");
    }
    *nbr_federates = i;
    if (*nbr_federates) /* if member federates exist */
    {
        event_msg_info_ptr->Rti.rti_svc_nbr = RTI_TIME_ADV_GRANT;
        event_msg_info_ptr->destinations_list = destinations_list;
        statistic =
            rtimgr_get_RTI_ambsvc_time(federate_nbr,
                                      RTI_TIME_ADV_GRANT_SETUP,
                                      *nbr_federates);
    }
    if (Tmp) {printf("tm_time_adv_grant_setup Statistic %8.5f Press Enter \n", statistic );}
    //gets(str);
    eventmgr_change_processing_mode(event_msg_info_ptr,RTI_PROCESSING);
    iomgr_send_ioevent(event_msg_info_ptr, statistic, TRUE);

    // WILL: I added to iomgri_send_ioevent priority to TRUE for new AddPriorityEvent
    //          event_msg_info_ptr->Time.PhysicalTime = statistic +
    //          CurrentFederateTime( event_msg_info_ptr->Rti.federate_name );
    //          AddPriorityEvent(stdout, "Out", event_msg_info_ptr );
    }
    else
        statistic = 0.0;

    return(statistic);
}

/* fed amb services */
/* ----- DocHeading----- */

```

30 June 1999

```

extern double tm_query_fed_LBTS(int federate_nbr,
                                EVENT_MESSAGE_TYPE
                                int
                                *event_msg_info_ptr,
                                *nbr_federates)
/* ----- EndDocHead----- */
{
/* ----- DocComment----- */
/* tm_query_fed_request:
*   Received a request for LBTS from the LBTS time Controller federate,
*   therefore, return the desired federate info via a reporting event.
*   future : only report info when have recvd child node values
*/

RTI_EVENT_MSG_TYPE    *rti_info;
FEDERATE_DESTINS_TYPE *dest_element;
double    svc_statistic=0.0;
int    federate_rcvd_msgs, federate_sent_msgs;
double federate_LBTS;
int    all_nodes_reported=FALSE;
EVENT_MESSAGE_TYPE    *NewMsgP;

if (TmP) {printf("tm_query_fed_LBTS- federate_nbr:%d\n", federate_nbr);}

    eventmgr_retrieve_LBTS_info(federate_nbr,
                                &federate_rcvd_msgs,
                                &federate_sent_msgs,
                                &federate_LBTS,
                                &all_nodes_reported);

if (TmP) { printf("retrieved values: federate_rcvd_msgs:%5d sent_msgs:%5d LBTS:%8.5f all_nodes:
%2d\n",
                federate_rcvd_msgs,
                federate_sent_msgs,
                federate_LBTS,
                all_nodes_reported);
}

/*   create event to return to RTI- RTI_RPTNG_FED_LBTS
    containing LBTS info: the LBTS of unprocessed events on TSO queue
    and counts of current rcvd and sent msgs
*/
    rti_info = &event_msg_info_ptr->Rti;
    rti_info->nbr_rcvd_msgs = federate_rcvd_msgs;
    rti_info->nbr_sent_msgs = federate_sent_msgs;
    rti_info->LBTS_time = federate_LBTS;
    rti_info->origin_fed_name = federate_nbr;
    event_msg_info_ptr->Rti.rti_svc_nbr = RTI_RPTNG_FED_LBTS;
    eventmgr_change_processing_mode(event_msg_info_ptr, RTI_PROCESSING);
    svc_statistic = rtimgr_get_Fed_ambsvc_time(RTI_QUERY_FED_LBTS, 1);

/* if I am not a parent, send report to parent */
if (!rtimgr_federate_is_parent(federate_nbr))
{
    dest_element =
        om_create_destinations_element(FOMdb.nodes[federate_nbr].parent);
    if (TmP) {printf("added federate: %2d to destinations list  Statistic %8.5f %8.5f
\n",
                    FOMdb.nodes[federate_nbr].parent, svc_statistic,
                    event_msg_info_ptr->Time.PhysicalTime );
    }
    event_msg_info_ptr->destinations_list= dest_element;
    iomgr_send_ioevent(event_msg_info_ptr, svc_statistic, TRUE );
// WILL - set true for priority instead
//     event_msg_info_ptr->Time.PhysicalTime = svc_statistic +
//     CurrentFederateTime( event_msg_info_ptr->Rti.federate_name );
//     AddPriorityEvent(stdout, "Out", event_msg_info_ptr );
}

```

30 June 1999

```

else
/* otherwise, I am a parent, just send info back to me on my local rti q */
{
    NewMsgP = c_Duplicate_Event_Message(event_msg_info_ptr);
    NewMsgP->Time.PhysicalTime =svc_statistic +
        CurrentFederateTime( federate_nbr );
    // dest_element =
    //   om_create_destinations_element(federate_nbr);
    if (TMp) {printf("adding federate: %2d rti_event to local rtiq sched for %8.5f \n",
        federate_nbr,      NewMsgP->Time.PhysicalTime );
    }
    eventmgr_change_processing_mode(event_msg_info_ptr,DONE_PROCESSING);

    // NewMsgP->destinations_list=dest_element;
    /* return msg back to rti q for local processing */
    AddPriorityEvent(stdout, "InToRTI", NewMsgP );
}

return(svc_statistic);
}

/* ----- DocComment---*/
/* tm_time_adv_grant :
*/
/* ----- DocHeading---*/

```

30 June 1999

```
extern double tm_time_adv_grant(int federate_nbr,
                                double fedrtn_time)
/* ----- EndDocHead---*/
{
    double      svc_statistic=0.0;

    if (TMP) { printf("tm_time_adv_grant: processing fed:%d fed time:%d \n",
                      federate_nbr, fedrtn_time);
    }
    /* adjust LBTS to granted time value */

    FedExdb.federates[federate_nbr].federate_LBTS = fedrtn_time;

    /* A NOTE from Will:  This Time has to be computed differently than the others */

    svc_statistic = rtimgr_get_RTI_ambsvc_time(federate_nbr, RTI_TIME_ADV_GRANT, 1);

    return(svc_statistic);
}

/* ----- DocComment---*/
/* tm_clear_LBTS_info:
*/
/* ----- DocHeading---*/
```

```
extern void  tm_clear_LBTS_info()
{
```

```
    FedExdb.fedrtn_time_mgmt_info.current_state = INACTIVE;  /* inactive */
```

```
}
/* DocHeading */
```


/* file: event.h */ /* network types */

```
#define ETHERNET      0
#define ATM           1
```

```
/* distribution of fedex mananagement*/
#define DISTRIBUTED   1
#define CENTRALIZED   0
```

```
/*
--* event.h definition of fundamental Event Mgr data types
--*
*/
```

```
/* WhoGetsIt modes used by eventmgr_change_processing_mode */
#define DONE_PROCESSING      0
#define SIM_PROCESSING       1
#define RTI_PROCESSING       2
```

```
/* rti model - rti svc msg */
typedef struct {
    double                JustBecause;
    int    rti_svc_nbr ;      /* action to perform */
    int    fedrtn_exname ;    /* name is number */
    int    federate_name ;    /* name is number of node */
    int    origin_fed_name;    /* originating federate for cmd */
    int    fedrtn_type ;
    int    fedrtn_save_label ; /* name is number */
    int    obj_class_nbr ;
    int    obj_instance_nbr ;
    int    interact_class_nbr ;
    int    interact_instance_nbr ;
    int    tag_name ;          /* name is number */
    int    passive_subscription_indicator ;
    double fedrtn_time ;
    int    transportation_type ;
    int    routing_space_nbr ;
    int    region_nbr ;
    int    nbr_rcvd_msgs ;
    int    nbr_sent_msgs ;
    double LBTS_time ;
    int    SpareForDoubleWordBoundary;
} RTI_EVENT_MSG_TYPE;
```

```
typedef struct Federate_Destination {
    double                JustBecause;
    int    federate ;
    struct Federate_Destination *next ;
} FEDERATE_DESTINS_TYPE;
```

```
typedef struct ColorTag {
    double                LowestUnprocessedTSO;
    unsigned int    ColorTag;
    unsigned int    Sent;
    unsigned int    Received;
    unsigned int    Boundary;
} ColorTagType ;
```

```
/* -- typedef Event_Times -- */
typedef struct {
    double    PhysicalTime ;
    double    VirtualTime ;
```

30 June 1999

```

int      Label ;
int      UniqueMsgId ;
double   OutEnter ;
double   OutService ;
double   OutComplete ;
double   RTIEnter ;
double   RTIService ;
double   RTIComplete ;
double   TsoEnter ;
double   TsoRtService ;
double   TsoService ;
double   TsoComplete ;
double   TsoRtComplete ;
} Event_Times_Type ;

typedef struct Sim_Control_Data {
int      UnitId ;
int      Effect ;
int      ExtentOfEffect ;
int      InteraClass ;
int      ObjectClass ;
} SIMCONTROLDATATYPE;

typedef struct Control_Flow {
double   JustBecause;
int      RTI ;
int      SIM ;
} ControlFlowType ;

/* -- typedef Event_Message -- moved to .h */
typedef struct Event_Message {
Event_Times_Type      Time ;
ControlFlowType      WhoGetsIt;
RTI_EVENT_MSG_TYPE    Rti ;
ColorTagType          Color;
struct Federate_Destination *destinations_list;
SIMCONTROLDATATYPE    Sim ;
struct Event_Message  *nqep ; /* next queue element pointer */
int                    Marked_Delete ;
int                    marker_mode; /* count between red=odd white=even */
} EVENT_MESSAGE_TYPE;

extern void eventmgr_create_event(int service_nbr);
extern void eventmgr_retrieve_LBTS_info(int federate_nbr,
int *federate_rcvd_msgs,
int *federate_sent_msgs,
double *federate_LBTS,
int *all_nodes_reported);

extern void eventmgr_change_processing_mode(EVENT_MESSAGE_TYPE *event_msg,
int new_mode);

extern int eventmgr_get_destination(int federate_nbr,
EVENT_MESSAGE_TYPE *event_msg_info_ptr);

/* DocMethod */
/* eventmgr.h */
/* file: eventmgr.h */

```

```

#include "iomgr.h"

/* rti model - rti svc msg */
typedef struct {

    int    rti_svc_nbr;           /* action to perform */
    int    fedrtn_exname;        /* name is number */
    int    federate_name;        /* name is number */
    int    origin_fed_name;      /* originating federate for cmd */
    int    fedrtn_type;
    int    fedrtn_save_label;     /* name is number */
    int    obj_class_nbr;
    int    obj_instance_nbr;
    int    interact_class_nbr;
    int    interact_instance_nbr;
    int    tag_name;             /* name is number */
    int    passive_subscription_indicator;
    int    fedrtn_time;
    int    transportation_type;
    int    routing_space_nbr;
    int    region_nbr;
    int    nbr_rcvd_msgs;
    int    nbr_sent_msgs;
    int    LBTS_time;

} RTI_EVENT_MSG_TYPE;

/* time mgmt info */
typedef struct {

    int    marker_mode;          /* toggle between red=1 white=0 */
} TIME_MGMT_TYPE;

/* main event queue element */
typedef struct {

    int                processor_type;    /* done=0, sim=1, rti=2 */
    int                sim_model_info;
    RTI_EVENT_MSG_TYPE rti_svc_msg;
    IOMGR_IOEVENT_INFO io_info;
    int                time_des_info;
    TIME_MGMT_TYPE     time_mgmt_info;
} EVENT_MESSAGE_TYPE;

extern void    eventmgr_create_event(int    service_nbr);
extern void    eventmgr_retrieve_LBTS_info(int    federate_nbr,
                                             int    *federate_rcvd_msgs,
                                             int    *federate_sent_msgs,
                                             int    *federate_LBTS,
                                             int    *all_nodes_reported);

extern void    eventmgr_change_processing_mode(EVENT_MESSAGE_TYPE *event_msg,
                                             int    new_mode);

/*    DocMethod    */

```

```

/* io_mgr.h */

/* network types */

#define ETHERNET 0
#define ATM 1

/* distribution of fedex mananagement*/
#define DISTRIBUTED 1
#define CENTRALIZED 0

typedef struct federate_destinations {

    int federate_nbr; /* node */
    struct federate_destinations *next;
} FEDERATE_DESTINS_TYPE;

typedef struct {

    FEDERATE_DESTINS_TYPE *destinations_list;
} IOMGR_IOEVENT_INFO;

/* DocMethod */
/* eventmgr.h */
/* file: eventmgr.h */

#include "iomgr.h"

/* rti model - rti svc msg */
typedef struct {

    int rti_svc_nbr; /* action to perform */
    int fedrtn_exname; /* name is number */
    int federate_name; /* name is number */
    int fedrtn_type;
    int fedrtn_save_label; /* name is number */
    int obj_class_nbr;
    int obj_instance_nbr;
    int interact_class_nbr;
    int interact_instance_nbr;
    int tag_name; /* name is number */
    int passive_subscription_indicator;
    int fedrtn_time;
    int transportation_type;
    int routing_space_nbr;
    int region_nbr;

} RTI_EVENT_MSG_TYPE;

/* main event queue element */
typedef struct {

    int sim_model_info;
    RTI_EVENT_MSG_TYPE rti_svc_msg;
    IOMGR_IOEVENT_INFO io_info;

```

```
int          time_des_info;
int          time_mgmt_info;

} EVENT_MESSAGE_TYPE;

extern void  eventmgr_create_event(int  service_nbr);

/*  file: serv_crit.h  */

#include  "rti_services.h"

#define  SERVICE_CRITERIA_MAX_DATA_LINES 70  /* number of RTI services */
#define  CRITERIA_MAX 80  /* number of criteria */

typedef struct {
    char  fedrtn_exname_exists;
    char  fedrtn_type_required;
    char  save_in_process;
    char  restore_in_process;
    char  fedrtn_member;
    char  obj_ownership_release;
    char  delete_objs_owned;
    char  fedrtn_save_label_exists;
    char  fedrtn_save_announced;
    char  fedrtnsave_label;
    char  fed_save_initiated;
    char  fed_save_success;
    char  fed_restore_success;
    char  object_class_exists;
    char  attribs_exist;
    char  obj_instance_owned;
    char  obj_class_published;
    char  obj_instance_exists;
    char  instance_attribs_exist;
    char  obj_class_subscribed;
    char  tag_name;
    char  acquiring_instance_attribs;
    char  interact_class_exists;
    char  interact_class_params_exist;
    char  interact_class_published;
    char  passive_subscription_indicator;
    char  fedrtn_time;
    char  transportation_type;
} RTI_CRITERIA_TYPE;

typedef struct {
    int  service_nbr;
    char  service_type[4];
    char  criteria[CRITERIA_MAX];  /* positions 2-29 */
    enum RTI_SERVICE_TYPE  rtiamb_action;
    char  rtiamb_action_name[30];
    float  rtiamb_cpu_svc_time;  /* time elapsed for this action */
    enum RTI_SERVICE_TYPE  fedamb_reaction;
    char  fedamb_reaction_name[30];
    float  fedamb_cpu_svc_time;
} RTI_SERVICE_TBL_ENTRY_TYPE;
```

```
extern int CriteriaCreate(RTI_SERVICE_TBL_ENTRY_TYPE *rtisvc_tbl_ptr);
/* file: rti_services.h */
```

```
enum RTI_SERVICE_TYPE {
    RTI_CREATE_FEDEX = 1001,
    RTI_JOIN_FEDEX = 1002,
    RTI_RQST_FEDRTN_SAVE = 1010,
    RTI_FED_SAVE_BEGUN = 1012,
    RTI_FED_SAVE_COMPLETE = 1013,
    RTI_PUBLISH_OBJCLSS = 2001,
    RTI_PUBLISH_INTCLSS = 2003,
    RTI_SUBSCRIBE_INTCLSS = 2008,
    RTI_SUBSCRIBE_OBJCLSS = 2005,
    RTI_REGISTER_INST = 3001,
    RTI_UPDATE_ATTRIB = 3003,
    RTI_SEND_INT = 3005,
    RTI_RQST_ATTRIB_VALS = 3014,
    RTI_CREATE_UPDATE_REGION = 6001,
    RTI_ASSOC_REGION = 6005,
    /* fed amb svcs - pairs */
    RTI_INITIATE_FED_SAVES = 1011,
    RTI_FEDRTN_SAVED = 1014,
    RTI_DISCVR_OBJ = 3002,
    RTI_REFLECT_ATTRIB = 3004,
    RTI_RECEIVE_INT = 3006,
    RTI_PRVD_ATTRIB_VALS = 3015
};
```

```
/* file: rti.h */
/* This file contains the structures for the rti model */
```

```
#define TRUE 1
#define FALSE 0
#define ACTIVE 1
#define INACTIVE 0
#define MAX_NBR_FEDERATIONS 1
#define MAX_NBR_FEDERATES 10
#define MAX_NBR_OBJ_CLASSES 10
#define MAX_NBR_INTERACT_CLASSES 10
#define MAX_NBR_OBJ_INSTANCES 2000
#define MAX_NBR_INTERACT_INSTANCES 2000
#define MAX_NBR_SAVE_LABELS 4
```

```
/* all possible RTI and FED Ambassador services */
```

```
#define UPDATE 1
#define REFLECT 2
#define DISCOVER 3
```

```
#include "regions.h"
```

```
/* RID: RTI Initialization Data for configuration */
```

```
typedef struct {
    int distrib_fedex; /* boolean true=distributed false=centralized */
    int network_type; /* 0=ethernet 1=ATM 2= other */
};
```

30 June 1999

```
} RID_INFO_TYPE;
```

```
typedef struct {
```

```
    int    associated_routing_space;
    int    associated_nbr_attrib_parms;
```

```
} CLASS_TYPE;
```

```
/* info from FOM file: Federation Object Model,
   used for initialization of Federation scenario */
typedef struct {
```

```
    int    fedrtn_name;
    float  fedrtn_rti_version;
    int    nbr_routing_spaces;
    int    nbr_fedrtn_objclasses;
    CLASS_TYPE  object_classes[MAX_NBR_OBJ_CLASSES];
    int    nbr_fedrtn_interact_classes;
    CLASS_TYPE  interact_classes[MAX_NBR_INTERACT_CLASSES];
```

```
} FOM_INFO_TYPE;
```

```
typedef struct {
```

```
    int    associated_routing_space;
    int    nbr_attribs;
    int    owner_federate;
    int    published;                /* boolean */
    int    nbr_subscribed_federates;
```

```
} OBJECT_CLASS_TYPE;
```

```
typedef struct {
```

```
    int    associated_routing_space;
    int    nbr_parms;
    int    owner_federate;
    int    published;                /* boolean */
    int    nbr_subscribed_federates;
```

```
} INTERACT_CLASS_TYPE;
```

```
typedef struct instance_node {
```

```
    int                owner_federate;                /* federate nbr */
    int                objclass_nbr;
    REGIONS_LIST_TYPE *associated_regions;
    struct instance_node *next;
```

```
} OBJECT_INSTANCE_TYPE;
```

```
typedef struct interact_node {
```

```
    int                owner_federate;                /* federate nbr */
    int                interact_class_nbr;
    REGIONS_LIST_TYPE *associated_regions;
    struct interact_node *next;
```

```
} INTERACT_INSTANCE_TYPE;
```

```
/* info on each federate */
```

```
typedef struct {
```

30 June 1999

```

    int    assoc_fedrtn_name;
    int    federate_state;
    int    federate_rti_version;
    int    receives_updates;
    OBJECT_INSTANCE_TYPE    *registered_obj_instances; /* obj instances active */
    INTERACT_INSTANCE_TYPE    *active_interactions;    /* interactions active */
    int    saved_status;

} FEDERATE_INFO_TYPE;

typedef struct {
    int    FedEx_name_exists;
    int    save_in_process;
    int    restore_in_process;
    int    processing_release_ownership;
    int    synch_in_process;
    int    synch_point_announced;
    int    acquiring_attribs_in_process;
    /*    ...more */
} FEDEX_STATE_INFO;

/* FEDEX info: Federation Execution info */
/* info kept on the Federation when Executing */
typedef struct {

    int                fedrtn_name; /* name of federation */
    OBJECT_CLASS_TYPE    fedrtn_objclasses[MAX_NBR_OBJ_CLASSES];
    INTERACT_CLASS_TYPE    fedrtn_interact_classes[MAX_NBR_INTERACT_CLASSES];
    int                nbr_member_federates; /* nbr of active federates */
    FEDERATE_INFO_TYPE    federates[MAX_NBR_FEDERATES]; /* member federates */
    int                nbr_fedrtn_obj_instances;
    OBJECT_INSTANCE_TYPE    fedrtn_obj_instances[MAX_NBR_OBJ_INSTANCES];
    int                nbr_fedrtn_interact_instances;
    INTERACT_INSTANCE_TYPE    fedrtn_interact_instances[MAX_NBR_INTERACT_INSTANCES];
    FEDEX_STATE_INFO    fedex_state_status; /* status of fexex for cmp with
criteria */
    int                nbr_fedrtn_regions;
    REGIONS_INFO_TYPE    fedrtn_regions[MAX_NBR_REGIONS];
    int                nbr_save_names;
    int                save_names[MAX_NBR_SAVE_LABELS]; /* save points */

} FEDEX_INFO_TYPE;

extern FOM_INFO_TYPE    FOMdb;
extern FEDEX_INFO_TYPE    FedExdb;
extern RID_INFO_TYPE    RIDdb;

```

```
/* file: regions.h */
```

```
#define    MAX_NBR_REGIONS    100
```

```
typedef struct subscribed_node {
    int    objclass_nbr;
    int    federate_name;
    struct subscribed_node *next;

```



```

} SUBSCRIBED_INFO_TYPE;

typedef struct assoc_regions_node {
    int            region_nbr;
    struct assoc_regions_node *next;
} REGIONS_LIST_TYPE;

typedef struct regions_node {
    SUBSCRIBED_INFO_TYPE    *subscribed_objects;
    SUBSCRIBED_INFO_TYPE    *subscribed_objects_tail;
    SUBSCRIBED_INFO_TYPE    *subscribed_interacts;
    SUBSCRIBED_INFO_TYPE    *subscribed_interacts_tail;
} REGIONS_INFO_TYPE;

/* file: io_mgr.h      */

/* network types */

#define ETHERNET    0
#define ATM         1

/* distribution of fedex mananagement*/
#define DISTRIBUTED  1
#define CENTRALIZED  0

typedef struct federate_destinations {
    int            federate_nbr;
    struct federate_destinations *next;
} FEDERATE_DESTINS_TYPE;

typedef struct {
    FEDERATE_DESTINS_TYPE    *destinations_list;
    FEDERATE_DESTINS_TYPE    *destinations_list_tail;
} IOMGR_IOEVENT_INFO;

/* file: stats.h */

#define MAX_STATISTIC_LABEL_CHARS    15
/* DocMethod */
/* proto.h */
#define SCENARIOHigh 3
#define SCENARIOLow 7
#define SCENARIOLimitsOnFederates (SCENARIOHigh+SCENARIOLow)
#define START 40.05
#define STARTDISPLAYINTERVAL 1540.0
#define TIMEADVANCE 1.0
#define TIMEadvTIMEOUT 0.7
#define REPEATTIMEADVANCE 1.0
#define HIGHFACTOR 4.0
#define MAXBATTALIONS 80

```

30 June 1999

```

#define      FIXSUBORDINATES      0
#define      FIXCOMMANDERS      0
#define      ENDitALL      1440.0
#define      EPOCH      45.0

extern double CompleteNextServicePhysicalTime();
extern double EventManager( FILE *out, FILE *Lg , int *AreQueuesEmpty, struct Region_Node_Handle
*RNH ) ;
extern double CurrentFederateTime( int Fed ) ;
extern double CurrentPhysicalTime();
extern double GetLBTSfromFederate( int FedIdPlus ) ;
extern double LowestTimeOfQueueEtherNet( int *Fed, int Que );
extern double LowestTimeOfQueueWithDelay( int *Fed, int Que );
extern double NextEndTime( int *Federate, int *Que );
extern double QueueEnd( struct Event_Message *Pptr );
extern double SimModel( struct Event_Message *P, double PhysTime, struct Region_Node_Handle
*RNH ) ;
extern double triangle( double c ) ;

extern int AnyPotentialMessagesToReceiveWithOldColorTag( );
extern int AddToFilterSubordinates( FILE *out, struct Unit_List *ULp, struct Filter_Unit_List
*top ) ;
extern int AddToNode( FILE *out, struct Nodes_of_Fed_List *NodeOfFed, struct
Unit_Characteristics *UnitChar ) ;
extern int AddToRegionElements( FILE *out, struct Region_List *RegList, struct
Region_Element_List *RegEleList, struct Unit_Characteristics *UnitChar );
extern int BuildInitState( struct Comm_Net_Association **CommNet, struct Truth_Group_List
**PvTruth, struct Unit_Characteristics **UnitChar, struct Unit_List
**UnitList,
char *filename, int Force );
extern int ColorRTIService( struct Event_Message *MsgPtr, int *Colorid, int *LineOffset );
extern int ColorSIMinRTI( struct Event_Message *MsgPtr, int *ColorSel, int *LineOffset );
extern int CountSubrEquip( FILE *out, struct Unit_List *ULp );
extern int CpuOfPvT( struct Truth_Group_List *PvTp );
extern int CpuOfUnit( struct Unit_List *UnitListp );
extern int GetTotalEquipByLevel( int i );
extern int GetTotalPersonByLevel( int i );
extern int MaxEcheleon( FILE *out, struct Unit_Characteristics *UnCrA , char *str );
extern int PickSome( int LowBound, int UpBound );
extern int PublishByFederate( FILE *out, struct Nodes_of_Fed_List *NodeOfFedList );
extern int RegisterRegions( FILE *out, struct Region_List *RegList );
extern int RemoveRegionReference( struct Unit_Characteristics *UnitChar, struct Region_List
*lxtReg );
extern int SubscribeByFederate( FILE *out, struct Nodes_of_Fed_List *NodeOfFedList );
extern int TestForDiffColor( unsigned int ColorTag );
extern int TSOBoundMessage( struct Event_Message *Msg );
extern int UnitCharacter( struct Unit_Characteristics *uptr, char *line, int Who_is_it
);
extern int UnitHierBuild( struct Comm_Net_Association **CommNet, struct
Truth_Group_List **PvTruth, struct Unit_Characteristics **UnitChar, struct Unit_List
**UnitList, FILE *in, int Who_is_it );
extern int imax( int A, int B );
extern int imin( int A, int B );

extern struct Event_Message *GetLowestTimeMessage( int Fed, int Que );
extern struct Event_Message *NextEvent( FILE *out, double *P, double TmOfSer );
extern struct Event_Message *NextMessage( int *i, int *j );
extern struct Event_Message *SetEventMessage( int Action, int Federate, int RTIcommand, int
SIMcommand, double lPhysicalTime, double lVirtualTime, char *sptr );
extern struct Event_Message *SetExtendEventMessage( int RTIcommand, int SIMcommand, int Action,
int f_xn, int Fed, int o_c_n, int o_i_n, int i_c_n, int i_i_n, double fed_t, int reg_n, int
r_s_n, int n_r_m, int n_s_m, double LBTS_t, double lPhyT, double lVirT, char *sptr );
extern struct Comm_Net_Association *c_Comm_Net_Association( char *sptr );
extern struct Comm_Net_List *c_Comm_Net_List( char *sptr );
extern struct Event_Message *c_Duplicate_Event_Message( struct Event_Message *A );
extern struct Event_Message *c_Event_Message( char *sptr );
extern struct Federate_Destination *c_Federate_Destination( char *s );
extern struct Filter_Unit_List *CmdUnitNotInRegion( struct Region_Element_List *R );

```

30 June 1999

```

extern struct Filter_Unit_List *FilterByEchEquip( FILE *out, struct Unit_Characteristics
*UnCharOrigin, int Echeleon, int TotalEquip);
extern struct Filter_Unit_List *FilterByEcheleon( FILE *out, struct Unit_Characteristics
*UnCharOrigin, int Echeleon);
extern struct Filter_Unit_List *FilterNotAssignedToFed( FILE *out, struct Unit_Characteristics
*UnCharOrigin );
extern struct Filter_Unit_List *FilterNotInRegion( FILE *out, struct Unit_Characteristics
*UnCharOrigin );
extern struct Filter_Unit_List *PutAllInRegBySideInFilterList( int Side, struct
Region_Element_List *RegEleList ) ;
extern struct Filter_Unit_List *PutNumInRegBySideInFilterList( int Number, int S, struct
Region_Element_List *RgElst );
extern struct Filter_Unit_List *PutRegionInFilterList( struct Region_Element_List *R);
extern struct Filter_Unit_List *PutCoInRegionInFilterList(struct Region_Element_List *e);
extern struct Filter_Unit_List *c_Filter_Unit_List( char *sptr );
extern struct InterestList *c_InterestList( char *sptr ) ;
extern struct Node_List *c_Node_List( char *sptr ) ;
extern struct Node_Table_Def *c_Node_Table_Def( char *sptr ) ;
extern struct Nodes_of_Fed_List *c_Nodes_of_Fed_List( char *sptr ) ;
extern struct Nodes_of_Fed_List *FindNode( int Id, struct Nodes_of_Fed_List *FedList
);
extern struct Nodes_of_Region_List *c_Nodes_of_Region_List( char *sptr );
extern struct Nodes_wrt_Region_List *c_Nodes_wrt_Region_List(char *s) ;
extern struct Order_Packet *c_Order_Packet( char *sptr ) ;
extern struct Region_Definition *c_Region_Definition( char *sptr ) ;
extern struct Region_Element_List *c_Region_Element_List(char *sptr ) ;
extern struct Region_List *FindRegion( int RegId, struct Region_List *RegList ) ;
extern struct Region_List *c_Region_List( char *sptr ) ;
extern struct Region_Node_Handle *c_Region_Node_Handle( char *sptr ) ;
extern struct Serv_Characteristics *c_Serv_Characteristics( char *sptr ) ;
extern struct Serv_List *c_Serv_List( char *sptr ) ;
extern struct Serv_Stack *c_Serv_Stack( char *sptr ) ;
extern struct Task_List *c_Task_List( char *sptr ) ;
extern struct Truth_Group_List *c_Truth_Group_List( char *sptr ) ;
extern struct Truth_Group_List *PvToUnit( struct Unit_List *UnitListp ) ;
extern struct Unit_Characteristics *c_Unit_Characteristics( char *sptr ) ;
extern struct Unit_Characteristics *FillRegion(FILE *out, int ApproxEquip, int
*ActualEquip, struct Region_List *CurrentRegion, struct Unit_Characteristics
*UnCharOrigin, struct Region_Element_List **ElementOfRegion ) ;
extern struct Unit_Characteristics *FindUnit( int Id );
extern struct Unit_List *c_Unit_List( char *sptr ) ;
extern struct Unit_List *UnitLocate( int Index, int Who_is_it ) ;
extern struct Unit_Region_List *c_Unit_Region_List( char *sptr ) ;
extern struct Units_on_Node_List *c_Units_on_Node_List( char *sptr ) ;

extern unsigned int CurrentFederationColor();
extern unsigned int GetColorTag( int Federate );
extern unsigned int GetTotalEquip();
extern unsigned int StartLBTSCalculation();

extern void AddCommandRegions( FILE *out, char Which, struct Region_List *RegionList, struct
Filter_Unit_List *FilterList );
extern void AddEvent( FILE *out, char *Type, struct Event_Message *Add );
extern void AddEventToQueue( FILE *out, struct Event_Message **Top, int Fed, int Qid, struct
Event_Message *Add );
extern void AddNewRegion( FILE *out, char Which, struct Region_List *RegionList, struct
Filter_Unit_List *FilterList );
extern void AddNodeLinksToRegionsByUnit( FILE *out, struct Unit_Characteristics
*UnitChar, struct Region_List *RegList, struct Nodes_of_Fed_List *FedList ) ;
extern void AddPriorityEvent( FILE *out, char *Type, struct Event_Message *Add );
extern void AddRatioUnitsToNode( FILE *out, struct Region_Node_Handle *Region_Node_Handles, int
Gn, int Ot, struct Filter_Unit_List **GnLst, struct Filter_Unit_List **OtLst ) ;
extern void AddRegionReference( struct Unit_Region_List **UntRegion, struct Region_List *lxtReg
) ;
extern void AddRegionToNode( struct Nodes_of_Fed_List *FedNode, struct Region_List *RegListEle
);

```

30 June 1999

```

extern void AddSupportUnitsToNode( FILE *out, struct Region_Node_Handle *RgNd_Hdl,
struct Filter_Unit_List **GnLt, struct Filter_Unit_List **OtLt );
extern void AddToRegion( FILE *out, char Which, struct Region_List *RegList, struct
Unit_Characteristics *UnitChar );
extern void ColorTag( int Federate, struct Event_Message *Add );
extern void CreateInteraction( struct Event_Message *p, double Pe, struct Region_Node_Handle
*RNH, double Et, struct Unit_Characteristics *Uar, double S, int Ty );
extern void CreateNodes( FILE *out, struct Region_Node_Handle *Region_Node_Handles );
extern void CreateRegions( FILE *out, int TotGnEquip, int TotOtEquip, struct
Region_Definition *Regions, struct Region_Node_Handle *Region_Node_Handles );
extern void CreateRTIreport(char *Which, int ColorTag, int NumDest, double Time, int Fed );
extern void Draw_InterestGroup(struct Unit_Characteristics *UnCrA );
extern void GetUnitsByCoTypeForRegion( FILE *out, int Cat, char Side, struct Region_List *Regs,
struct Region_Element_List *RgEls, struct Filter_Unit_List **CtTyL);
extern void Grow_Echeleon( FILE *out, int NumberOfPeers, int Side, struct Unit_List *Cmdr,
struct Unit_List **ULp, struct Unit_Characteristics **UnCharG, int *Level );
extern void GrowInitArmy( int GreenBattalions, int OtherBattalions, struct
Region_Node_Handle *RegNdH );
extern void GrowInterestGroup( struct Unit_Characteristics *UnCrA, struct Unit_Characteristics
*UnCrB, int MaxInList);
extern void Initialize_Friends();
extern void Initialize_Others();
extern void Initialize_RTI();
extern void InitSimModel( struct Region_Node_Handle *RNH );
extern void LalaClear();
extern void LalaDraw( int Node, int ObjId, int State, double x, double y, double xbase,
double xrange, double ybase, double yrange );
extern void LalaDrawLink( int State, int x1, int y1, int x2, int y2 );
extern void LalaFinished();
extern void LalaInit( int TotNodes, int TotObjects);
extern void LalaPlace( int State, int X, int Y );
extern void LalaTimeQueue(int Node, int State, double xS, double xE, double xb, double xr, int
Tag, int Offset );
extern void LalaUpdate( int Node, int ObjId, int State );
extern void MergeFilterList( struct Filter_Unit_List *List1, struct Filter_Unit_List *List2
);
extern void PrintEventsInSystem( FILE *out);
extern void PrintEventsProcessed( FILE *out);
extern void PrintFedTime(FILE *out, int Fed, struct Event_Message *ptr );
extern void PrintFilterList( FILE *out, struct Filter_Unit_List *FilteredList, char *Emark );
extern void PrintNodesOfFed( FILE *out, struct Nodes_of_Fed_List *NodeOfFedList, char *Emark
);
extern void PrintOneRegion( FILE *out, struct Region_List *RegList, char *Emark);
extern void PrintQueue( FILE *out, struct Event_Message *Pptr, char *s );
extern void PrintRegions( FILE *out, struct Region_List *RegList, char *Emark );
extern void PrintRegionsNodes( FILE *out, struct Region_List *RegList, char *Emark );
extern void PrintRegionElements( FILE *out, struct Region_Element_List *RegEle, char *E);
extern void PrintRTIEchelon( FILE *out, struct Unit_List *ULp);
extern void PrintRTIInstanceEchelon( FILE *out, struct Unit_List *ULp);
extern void PrintUnitNamePvT( FILE *out, struct Truth_Group_List *PvTp, char *Emark );
extern void PrintUnitsOfFed( FILE *out, struct Nodes_of_Fed_List *NodeOfFedList, char
*Emark );
extern void PrintUtilizationResourceTime( FILE *out, double interval, int Replicate );
extern void Print_2AlterEcheleon( FILE *out, struct Unit_List *ULp );
extern void Print_3AlterEcheleon( FILE *out, struct Unit_List *ULp );
extern void Print_AlterEcheleon( FILE *out, struct Unit_List *ULp );
extern void Print_CommAssoc( FILE *out, struct Comm_Net_Association *CmNetp, char *Emark )
;
extern void Print_EchSummary( FILE *out );
extern void Print_Echeleon( FILE *out, struct Unit_List *ULp);
extern void Print_InterestGroup( FILE *out, struct Unit_Characteristics *UnCrA ); ;
extern void Print_UnitC( FILE *out, struct Unit_Characteristics *luptr, char *Emark )
;
extern void Print_UnitC_Alter( FILE *out, struct Unit_Characteristics *luptr, char *Emark )
;
extern void Print_UnitC_File( char filename[], struct Unit_Characteristics *uptr );

```

30 June 1999

```

extern void Print_UnitC_comma( FILE *out, struct Unit_Characteristics *luptr, char *Emark )
;
extern void QueuesInitialize() ;
extern void QueuesPrint( FILE *out, int Replicate ) ;
extern void ResetEcheleon( int Iset ) ;
extern void ChangeFederateColorTag( int Federate, int *Sent, int *Received ) ;
extern void SetBaseResourceTime();
extern void TallyClearEch() ;
extern void TallyEcheleon( struct Unit_List *ULp ) ;
extern void TallyPrintEch( FILE *out, char *label ) ;
extern void UpdateEntity( struct Event_Message *ptr, double Time, struct Region_Node_Handle
*RNH, double EvTm, struct Unit_Characteristics *UnChar, double st );
extern void ViewEcheleonLeft( struct Unit_List *ULp ) ;
extern void ViewEcheleonRight( struct Unit_List *ULp ) ;
extern void ViewNew() ;
extern void ViewNext() ;
extern void XQueuesPrint(double PTime, int Fed, int Queue, int Tag, int Offset );
extern void ViewRefresh( struct Unit_List *ULp ) ;

```

```

extern void d_Comm_Net_Association( struct Comm_Net_Association *dptr );
extern void d_Comm_Net_List( struct Comm_Net_List *sptr );
extern void d_Event_Message( struct Event_Message *sptr );
extern void d_Filter_Unit_List( struct Filter_Unit_List *sp );
extern void d_Node_List( struct Node_List *sptr ) ;
extern void d_Node_Table_Def( struct Node_Table_Def *sptr );
extern void d_Order_Packet( struct Order_Packet *sptr );
extern void d_Serv_Characteristics( struct Serv_Characteristics *sptr );
extern void d_Serv_List( struct Serv_List *sptr ) ;
extern void d_Serv_Stack( struct Serv_Stack *sptr ) ;
extern void d_Task_List( struct Task_List *sptr ) ;
extern void d_Truth_Group_List( struct Truth_Group_List *sptr );
extern void d_Unit_Characteristics( struct Unit_Characteristics *sptr );
extern void d_Unit_List( struct Unit_List *sptr ) ;
extern struct Unit_Region_List d_Unit_Region_List(struct Unit_Region_List *s);
extern void rtimgr_clear();
extern void rtimgr_final_cleanup();
extern void d_Federate_Destination( struct Federate_Destination *sptr );

```

```

/* DocMethod */
/* regions.h */
/* file: regions.h */

```

```

#define MAX_NBR_FEDRTN_REGIONS 100

```

```

typedef struct subscribed_node {
    int class_nbr;
    int federate_name;
    struct subscribed_node *next;
} SUBSCRIBED_INFO_TYPE;

```

```

typedef struct assoc_regions_node {
    int region_nbr;
    struct assoc_regions_node *next;
} REGIONS_LIST_TYPE;

```

```

typedef struct regions_node {
    SUBSCRIBED_INFO_TYPE *subscribed_objects;
    SUBSCRIBED_INFO_TYPE *subscribed_interactions;
} REGIONS_INFO_TYPE;

```

```

/* DocMethod */
/* rti.h */
/* file: rti.h */
/* This file contains the structures for the rti model */

#include "regions.h"

#define RTI_DONE 0
#define SIM_PROCESSING 1
#define RTI_PROCESSING 2
#define OBJECT_TYPE 0
#define INTERACTION_TYPE 1

#define TRUE 1
#define FALSE 0
#define ACTIVE 1
#define INACTIVE 0
#define MAX_NBR_FEDERATIONS 1
#define MAX_NBR_FEDERATES 10
#define MAX_NBR_OBJ_CLASSES 10
#define MAX_NBR_INTERACT_CLASSES 10
#define MAX_NBR_OBJ_INSTANCES 30000
#define MAX_NBR_INTERACT_INSTANCES 5000
#define MAX_NBR_SAVE_LABELS 4
#define MAX_NBR_CHILDREN 10

/* federate status states */
#define ACTIVE 1

/* federate saved status */
#define SAVE_COMPLETE 0
#define SAVING 1

/* RID: RTI Initialization Data for configuration */
typedef struct {
    int distrib_fedex; /* boolean true=distributed false=centralized */
    int network_type; /* 0=ethernet 1=ATM 2= other */
} RID_INFO_TYPE;

typedef struct {
    int associated_routing_space;
    int associated_nbr_attrib_parms;
} CLASS_TYPE;

typedef struct { /* tree structured reduction network for reporting LBTS info */
    int parent; /* nbr of which parent node for reporting */
    int children; /* TRUE/FALSE children reporting to this node */
} LBTS_CONFIG_TYPE;

/* info from FOM file: Federation Object Model,
   used for initialization of Federation scenario */
typedef struct {
    int fedrtn_name;
    float fedrtn_rti_version;
    int nbr_routing_spaces;
    int nbr_fedrtn_objclasses;

```

```

    CLASS_TYPE  object_classes[MAX_NBR_OBJ_CLASSES];
    int  nbr_fedrtn_interact_classes;
    CLASS_TYPE  interact_classes[MAX_NBR_INTERACT_CLASSES];
    LBTS_CONFIG_TYPE  nodes[MAX_NBR_FEDERATES+1];
    int  LBTS_controller;
    int  fedrtn_global_lookahead_value;

} FOM_INFO_TYPE;

typedef struct {

    int  associated_routing_space;
    int  nbr_attribs;
    int  owner_federate;
    int  published;                /* boolean */
    int  nbr_subscribed_federates;

} OBJECT_CLASS_TYPE;

typedef struct {

    int  associated_routing_space;
    int  nbr_parms;
    int  owner_federate;
    int  published;                /* boolean */
    int  nbr_subscribed_federates;

} INTERACT_CLASS_TYPE;

typedef struct instance_node {
    int  owner_federate;                /* federate nbr */
    int  objclass_nbr;
    REGIONS_LIST_TYPE  *associated_regions;
} OBJECT_INSTANCE_TYPE;

typedef struct interact_node {
    int  owner_federate;                /* federate nbr */
    int  interact_class_nbr;
    REGIONS_LIST_TYPE  *associated_regions;
    struct interact_node  *next;
} INTERACT_INSTANCE_TYPE;

typedef struct {
    int  i_reported;
    int  nbr_children;
    int  children_names[MAX_NBR_CHILDREN];
    int  children_reported[MAX_NBR_CHILDREN];
    int  total_rcvd_msgs;
    int  total_sent_msgs;
    double best_LBTS;
    int  sent_initial_counts;

} LBTS_REDCTN_NETWK_INFO;

/* info on each federate */
/* NOTE: not all of these elements are functional currently.
 * These record elements will be utilized with future development
 */
typedef struct {
    int  assoc_fedrtn_name;

```

30 June 1999

```

    int    federate_state;          /* actively joined = TRUE */
    int    federate_rti_version;    /* currently not referenced, but setup */
    int    receives_updates;
    OBJECT_INSTANCE_TYPE             *registered_obj_instances; /* currently not used */
    INTERACT_INSTANCE_TYPE           *active_interactions;      /* currently not used */
    int    saved_status;             /* currently not used */
/* time mgmt info */
    double federate_logical_time;
    double federate_LBTS;
    int    federate_marker_mode;
    LBTS_REDCNTN_NETWK_INFO reduction_network_info;

} FEDERATE_INFO_TYPE;

/* conditional criteria ...add more later */
typedef struct {
    int    FedEx_name_exists;
    int    save_in_process;
    int    restore_in_process;
    int    processing_release_ownership;
    int    synch_in_process;
    int    synch_point_announced;
    int    acquiring_attribs_in_process;
} FEDEX_STATE_INFO;

typedef struct {
    double global_lookahead_value;
    double LBTS_current;
    double LBTS_proposed;
    int    current_state;          /* 1 = Federation LBTS compute active,
                                   0 = no pending compute */
} TIME_MGMT_INFO;

/* FEDEX info: Federation Execution info */
/* info kept on the Federation when Executing */
typedef struct {

    int    fedrtn_name;            /* name of federation */
    OBJECT_CLASS_TYPE             fedrtn_objclasses[MAX_NBR_OBJ_CLASSES];
    INTERACT_CLASS_TYPE           fedrtn_interact_classes[MAX_NBR_INTERACT_CLASSES];
    int    nbr_member_federates;   /* nbr of active federates */
    FEDERATE_INFO_TYPE            federates[MAX_NBR_FEDERATES+1]; /* member federates */
    int    nbr_fedrtn_obj_instances;
    OBJECT_INSTANCE_TYPE          fedrtn_obj_instances[MAX_NBR_OBJ_INSTANCES];
    int    nbr_fedrtn_interact_instances;
    INTERACT_INSTANCE_TYPE        fedrtn_interact_instances[MAX_NBR_INTERACT_INSTANCES];
    FEDEX_STATE_INFO              fedex_state_status; /* status of fedex for cmp with
criteria */
    int    nbr_fedrtn_regions;
    REGIONS_INFO_TYPE             fedrtn_regions[MAX_NBR_FEDRTN_REGIONS];
    int    nbr_federates_saving;
    int    nbr_save_names;
    int    save_names[MAX_NBR_SAVE_LABELS]; /* save points */
    TIME_MGMT_INFO                fedrtn_time_mgmt_info;

} FEDEX_INFO_TYPE;

extern FOM_INFO_TYPE             FOMdb;
extern FEDEX_INFO_TYPE          FedExdb;
extern RID_INFO_TYPE            RIDdb;

```



```
extern int      get_string_peices( char *lstr, char *pieces[], char *delimiter);
```

```
extern int      change_strgpeices_to_onestrng(int      num_peices,
                                                char      *pieces[128],
                                                char      criteria[80]);
```

```
/* DocMethod      */
```

```
/* rti_services.h */
```

```
/* file: rti_services.h */
```

```
enum RTI_SERVICE_TYPE {
    RTI_CREATE_FEDEX = 1001,
    RTI_JOIN_FEDEX = 1002,
    RTI_RQST_FEDRTN_SAVE = 1010,
    RTI_FED_SAVE_BEGUN = 1012,
    RTI_FED_SAVE_COMPLETE = 1013,
    RTI_PUBLISH_OBJCLSS = 2001,
    RTI_PUBLISH_INTCLSS = 2003,
    RTI_SUBSCRIBE_INTCLSS = 2008,
    RTI_SUBSCRIBE_OBJCLSS = 2005,
    RTI_REGISTER_INST = 3001,
    RTI_UPDATE_ATTRIB = 3003,
    RTI_SEND_INT = 3005,
    RTI_RQST_ATTRIB_VALS = 3014,
    RTI_CREATE_UPDATE_REGION = 6001,
    RTI_ASSOC_REGION = 6005,
    RTI_TIME_ADV_RQST = 5007,
    RTI_TIME_ADV_RQST_AVAIL = 5008,
    RTI_NXT_EVENT_RQST = 5009,
    RTI_NXT_EVENT_RQST_AVAIL = 5010,
    RTI_RPTNG_FED_LBTS = 5026,
    RTI_RPTNG_RCV_LBTS = 5027,
    RTI_RPTNG_SND_LBTS = 5028,

    RTI_DISCVR_SETUP = 3002,
    RTI_REFLECT_SETUP = 3004,
    RTI_RECEIVE_INT_SETUP = 3006,
    RTI_INITIATE_FED_SAVES_SETUP = 1011,
    RTI_LBTS_QUERY_SETUP = 5025,
    RTI_TIME_ADV_GRANT_SETUP = 5012,

    /* fed amb svcs - pairs */
    RTI_INITIATE_FED_SAVE = 1011,
    RTI_FEDRTN_SAVED = 1014,
    RTI_DISCVR_OBJ = 3002,
    RTI_REFLECT_ATTRIB = 3004,
    RTI_RECEIVE_INT = 3006,
    RTI_PRVD_ATTRIB_VALS = 3015,
    RTI_QUERY_FED_LBTS = 5025,
    RTI_TIME_ADV_GRANT = 5012
};
```

```
/* DocMethod      */
```

```
/* rtimgr.h      */
```

```
/* file: rtimgr.h      */
```

```
/* This file contains the structures for the rti model */
```

```
extern void      rtimgr_init(RTI_SERVICE_TBL_ENTRY_TYPE      *rtisvc_tbl_ptr);
```

30 June 1999

```

extern void    rtimgr_final_cleanup();

extern double  rtimgr_RTIEvent(EVENT_MESSAGE_TYPE *event_msg_info_ptr);

extern double  rtimgr_get_RTI_ambsvc_time(int    federate_nbr,
                                         int    RTIambsvc,
                                         int    nbr_federates);

extern double  rtimgr_get_Fed_ambsvc_time(int    fed_ambsvc,
                                         int    nbr_federates);

/* RTI Ambassador Services and setups */

extern double  fm_create_fedrtn_execution(int    federate_nbr,
                                         int    fedrtn_name);

extern double  fm_join_fedrtn_execution(int    federate_name,
                                         int    fedrtn_name);

extern double  fm_request_fedrtn_save(int    federate_nbr,
                                     int    *active_federates,
                                     EVENT_MESSAGE_TYPE *event_msg_info_ptr);

extern double  fm_federate_save_begun(int    federate_nbr,
                                     int    *nbr_federates);

extern double  fm_federate_save_achieved(int    federate_nbr);

extern double  fm_fedrtn_save_achieved();

extern double  fm_setup_fedrtn_complete_events(int    federate_nbr,
                                               int    *saving_feds,
                                               EVENT_MESSAGE_TYPE *event_msg_info_ptr);

extern double  dm_publish_objclass(int    federate_nbr,
                                   int    class_nbr);

extern double  dm_publish_interact_class(int    federate_nbr,
                                         int    class_nbr);

extern double  dm_subscribe_objclass(int    obj_class,
                                   int    federate_name,
                                   int    region_nbr);

extern double  dm_subscribe_interact_class(int    federate_name,
                                         int    class_nbr,
                                         int    region_nbr);

extern double  om_register_instance(int    class_nbr,
                                   int    instance_nbr,
                                   int    region_nbr,
                                   int    federate_nbr,
                                   EVENT_MESSAGE_TYPE *event_msg_info_ptr,
                                   int    *rtiamb_nbr_federates);

extern double  om_setup_discover_events(int    federate_nbr,
                                       int    class_nbr,
                                       int    region_nbr,
                                       EVENT_MESSAGE_TYPE *event_msg_info_ptr,
                                       int    *subscribed_nbr);

extern double  om_update_attrib_values(int    federate_nbr,

```

30 June 1999

```

        int instance_nbr,
        int tag_nbr,
        int fedrtn_time,
        EVENT_MESSAGE_TYPE *event_msg_info_ptr,
        int *rtiamb_nbr_federates);

extern double om_setup_reflect_events(int federate_nbr,
        int instance_nbr,
        int tag_nbr,
        int fedrtn_time,
        EVENT_MESSAGE_TYPE *event_msg_info_ptr,
        int *subscribed_nbr);

extern double om_send_interaction(int federate_nbr,
        int class_nbr,
        int region_nbr,
        int interact_instance_nbr,
        int tag_nbr,
        int fedrtn_time,
        EVENT_MESSAGE_TYPE *event_msg_info_ptr,
        int *rtiamb_nbr_federates);

extern double om_setup_receive_interaction_events(int federate_nbr,
        int class_nbr,
        int instance_nbr,
        int region_nbr,
        int tag_nbr,
        int fedrtn_time,
        EVENT_MESSAGE_TYPE *event_msg_info_ptr,
        int *subscribed_federates);

extern double tm_time_adv_request(int federate_nbr,
        int fedrtn_time,
        int *nbr_federates,
        EVENT_MESSAGE_TYPE *event_msg_info_ptr);

extern double tm_controller_LBTS_compute(int federate_nbr,
        int *nbr_federates,
        EVENT_MESSAGE_TYPE *event_msg_info_ptr);

extern double tm_query_fed_LBTS(int federate_nbr,
        EVENT_MESSAGE_TYPE *event_msg_info_ptr,
        int *nbr_federates);

extern double tm_time_adv_grant(int federate_nbr,
        double fedrtn_time);

extern double om_request_attrib_value_update();

extern double ddm_create_update_region(int federate_nbr,
        int region_nbr);

extern double ddm_associate_update_region(int federate_nbr);

/* Federate Ambassador Services */
extern double fm_initiate_federate_save();

extern double om_discover_object(int federate_nbr,
        int class_nbr,
        int region_nbr);

```

30 June 1999

```

extern double om_reflect_attrib_values(int instance_nbr,
                                       int class_nbr,
                                       int tag_nbr,
                                       int fedrtn_time);

extern double om_receive_interaction(int class_nbr,
                                     int instance_nbr);

extern double om_provide_attrib_value_update();

/* other util functions */

extern FEDERATE_DESTINS_TYPE *om_create_destinations_element(int federate_nbr);

extern int fm_is_fedrtn_saved();

extern void iomgr_send_ioevent(EVENT_MESSAGE_TYPE *event_msg_info_ptr,
                               double ServiceTime,
                               int priority);

extern void tm_clear_LBTS_info();

extern void rtimgr_clear();

extern void rtimgr_clear_reduction_network_info();

extern int rtimgr_federate_processed_initial_counts(int federate_nbr);

extern int rtimgr_federate_is_parent(int federate_nbr);

/* DocMethod */
/* serv_crit.h */
/* file: serv_crit.h */

#define SERVICE_CRITERIA_MAX_DATA_LINES 100 /* number of RTI services */
#define CRITERIA_MAX 80 /* number of criteria */

typedef struct {
    char fedrtn_exname_exists;
    char fedrtn_type_required;
    char save_in_process;
    char restore_in_process;
    char fedrtn_member;
    char obj_ownshp_release;
    char delete_objs_owned;
    char fedrtn_save_label_exists;
    char fedrtn_save_announced;
    char fedrtnsave_label;
    char fed_save_initiated;
    char fed_save_success;
    char fed_restore_success;
    char object_class_exists;
    char attribs_exist;
    char obj_instance_owned;
    char obj_class_published;
    char obj_instance_exists;
    char instance_attribs_exist;
    char obj_class_subscribed;

```

```

char tag_name;
char acquiring_instance_attribs;
char interact_class_exists;
char interact_class_params_exist;
char interact_class_published;
char passive_subscription_indicator;
char fedrtn_time;
char transportation_type;
} RTI_CRITERIA_TYPE;

```

```

typedef struct {
    int service_nbr;
    char service_type[4];
    char criteria[CRITERIA_MAX];          /* positions 2-29 */
    int rtiamb_action;
    char rtiamb_action_name[30];
    double rtiamb_cpu_svc_time;           /* time elapsed for this action */
    int fedamb_reaction;
    char fedamb_reaction_name[30];
    double fedamb_cpu_svc_time;
} RTI_SERVICE_TBL_ENTRY_TYPE;

```

```

typedef struct {
    int fedamb_reaction;
    char fedamb_reaction_name[30];
} RTI_FEDAMB_SERVICE_TYPE;

```

```

extern RTI_SERVICE_TBL_ENTRY_TYPE service_criteria[SERVICE_CRITERIA_MAX_DATALINES];

```

```

extern RTI_FEDAMB_SERVICE_TYPE fedamb_svcs[SERVICE_CRITERIA_MAX_DATALINES];
/* DocMethod */

```

```

/* soaGcnst.h */

```

```

/*
--* soaGcnst.h Constant values for a specific execution
--*
*/

```

```

/* Some parameters */

```

```

double DeadReconFactor = 0.000025 ; /* time per 'Remote Entity' */
double OwnedEntityFixed = 0.0 ; /* usually use service char times*/
double ATMsizeFixed = 0.0 ; /* Just for test */
double ATMdelayFixed = 0.016 ; /* Just for test */
double ATMchannels = 2016.0 ; /* Channels */
double ATMdataSize = 48.0 ; /* Bytes of data in cells */
double ATMheaderSize = 5.0 ; /* Header foreach cell */
double ATMcellpsec = 366792 ; /* Cells per second */
double ATMsecpcell = 0.000002726337; /* seconds per cell */
double SubscribedFactor = 0.00001; /* Time to update one entity */
double ReflectFactor = 0.00001 ; /* Time to ruct one entity update */
double DevelopingPlanFactor = 0.33 ; /* 1/3 of time allocated to plan B.X. */
double TaskConstantTime = 15.0; /* seconds */
double FuzzyReply = 21.0; /* seconds */
double MaxReply = 60.0;
double MaxSizeMsg = 4096.0;

```

```

struct Sim_Management SimCntrl ;

```

```

/* DocMethod */

```

```

/* soa_cnst.h */

```

```

/* Some parameters */

```

```

extern double DeadReconFactor ; /* 0.000025 ; time per 'Remote Entity' */
extern double OwnedEntityFixed ; /* 0.0 ; usually use service char times*/
extern double ATMsizeFixed ; /* 0.0 ; Just for test */
extern double ATMdelayFixed ; /* 0.016 ; Just for test */
extern double ATMchannels ; /* 2016.0 ; Channels */
extern double ATMdataSize ; /* 48.0 ; Bytes of data in cells */

```

30 June 1999

```

extern double    ATMheaderSize    ; /* 5.0 ; Header foreach cell */
extern double    ATMcellpsec      ; /* 366792 ; Cells per second */
extern double    ATMsecpcell      ; /* 0.000002726337; seconds per cell */
extern double    SubscribedFactor ; /* 0.00001; Time to update one entity */
extern double    ReflectFactor    ; /* 0.00001 ; Time to ruct one entity update */
extern double    DevelopingPlanFactor ; /* 0.33 ; 1/3 of time allocated to plan B.X. */
extern double    TaskConstantTime ; /* 15.0; seconds */
extern double    FuzzyReply       ; /* 21.0; seconds */
extern double    MaxReply         ; /* 60.0; */
extern double    MaxSizeMsg       ; /* 4096.0; */
extern struct Sim_Management SimCntrl ; /* */
/* DocMethod */
/* soa_defs.h */
#include "event.h"

/*
--* soa_defs.h definition of fundamental SOA data types
--*
*/

#define iUCp      if (0) /* turn on/off debugging or statistic collection f*/
#define iUHp      if (0) /* Also use as one perline for easy grep f*/
#define iTL       if (0) /* f*/
#define iTLpf     if (0) /* f*/
#define iPvTp     if (0) /* f*/
#define iCSn      if (0) /* f*/
/* The following defines may change for any scenario */
#define ORDER      1001
#define REPORT     2002
#define FIRE       3003
#define SENSE      4004
#define SUPPLY     5005
#define MOVE       7007

#define OffsetObject      0
#define ObjectsInSOM      5
#define RouteSOM          0
#define MissionSOM        1
#define UnitTypeSOM       2
#define PlanSOM           3
#define HealthSOM         4

#define OffsetInteraction  5
#define InteractionsInSOM 5
#define OrderSOM          0
#define ReportSOM         1
#define FireSOM           2
#define SenseSOM          3
#define SupplySOM         4

#define WarFighter        101
#define CombatSupport     202
#define CombatServiceSupport 303

#define ForceFixedAllocation 0.0 /* if # > 0 => {H,M,L: #%, #%, Rem} */
#define Friendlies 5000 /* maximums for SOA exercise f*/
#define Not_So 5000 /* f*/
#define LimitOnUnits (Friendlies+Not_So)
#define CmdOrder 1 /* Yes I could use enum(s) f*/
#define DataPacket 2 /* f*/
#define CommPacket 3 /* f*/
#define ReflectPacket 4 /* f*/
#define VTCPacket 5 /* f*/
#define AudioPacket 6 /* f*/

#define Encrypted 0 /* encrypted message State */

```

```

#define Data      1      /* just data no encryption State */

#define Friends    1      /* force codes f*/
#define Others     2      /* f*/
#define CpuNeighbor 3     /* f*/
#define Cpu_M      1      /* number of cpus per node f*/
#define Csn_M      32     /* each SP node has Cpu_M cpus f*/
#define Dvs_M      1      /* number of data vault fileservers f*/
#define Hub_M      1      /* number of ethernet Hubs f*/
#define Cws_M      14     /* number of Controller Workstations f*/
#define Cpi_M      4      /* number per WSM f*/
#define Nam_M      18     /* size of a Unit name f*/
#define Ntw_M      40     /* number of types of nodes in the Net f*/
#define Serv_Nam_M 48     /* size of a service name f*/
#define Unit_Size  64     /* minimum data size of a unit f*/

/* -- typedef Comm_Net_Association f*/
/* This will not include electronic reporting */
typedef struct Comm_Net_Association {
    double      MaxSize      ; /* system minimums */
    double      MaxTime      ;
    double      ReplyToCommandAt ;
    double      ReplyToPeerAt ;
    double      SubModeChangeAt ;
    int         SubModeOfActivity;
    int         SpecificUnit ;
    struct Order_Packet      *CmdOrdrp; /* to Cmd */
    struct Order_Packet      *PeerOrdrp; /* to peers */
    struct Unit_Characteristics *UChrp ; /* point to command unit */
    struct Comm_Net_Association *ngep ; /* just the next one */
    struct Comm_Net_List      *CmNLp ; /* if multi nets modeled */
} Comm_Net_Type ;

/* -- typedef Comm_Net_List f*/
typedef struct Comm_Net_List {
    double      AtTimeReply ; /* for live units */
    int         Id ; /* id of net */
    struct Order_Packet *nOrder; /* to peers */
    struct Comm_Net_List *ngep ; /* NULL no more nets */
    struct Unit_List    *ULstp ; /* add=issue remove=reply */
    char           Name[Nam_M] ; /* for now */
} Comm_Net_List_Type ;

/* -- typedef Truth_Group_List f*/
typedef struct Truth_Group_List {
    struct Unit_List    *ULstp;
    struct Truth_Group_List *TGLp ; /* another member or NULL */
    struct Node_Table_Def *NtDp ; /* point to owning cpu */
    struct Truth_Group_List *ngep ; /* just the 'next group element' */
                                   /* same cpu */
} Truth_Group_List_Type ;

/* -- typedef Node_Table_Def f*/
typedef struct Node_Table_Def {
    double      Cpufactor ; /* TBD additive */
    int         Id ; /* 1 to Csn_M */
    int         NumCategories; /* different types of reflection/subscription */
    double      Objects ; /* total */
    double      Subscribed; /* total */
    struct Truth_Group_List *TGLst; /* the ring of PTs for this node*/
    struct Node_List      *NdLstActp; /* Pt on NLstp of current service */
    struct Node_List      *NLstp ; /* borders */
} Node_Table_Def_Type ;

/* -- typedef Node_List f*/
typedef struct Node_List {

```

30 June 1999

```

double          Subscribe; /* percent to ghost      */
double          Reflect;   /* percent to ghost      */
int             Category;  /* category of this neighbor */
int             NTindex;   /* index or id of neighbor */
struct Node_List *nlep;
} Node_List_Type ;

/* -- typedef History -- */

typedef struct History {
    int         Fed ;
    int         Que ;
    struct Event_Message *EM ; /* next queue element pointer */
} History_Type ;

/* -- typedef Region_Definition          DocMethod          f*/
typedef struct Region_Definition { /* regions for node distribution */
double Commit ;
double Ratio ; /* Gn to Ot */
int     high ; /* category 1 warfighters */
int     other ; /* category 2 support etc...*/
} Region_Definition_Type;

/* -- typedef Node_Definition          DocMethod          f*/
typedef struct Node_Definition { /* define nodes of simulation */
int TotalGnEquip;
int TotalOtEquip;
int High;
int HighEquipment ;
int Low;
int LowEquipment ;
int HighGn;
int HighOt;
int LowGn;
int LowOt;
} Node_Definition_Type;

/* -- typedef Region_Node_Handle          DocMethod          f*/
typedef struct Region_Node_Handle {
    struct Region_Definition *RegionsDefined;
    struct Node_Definition *NodesDefined;
    struct Nodes_of_Fed_List *xtFed; /* list of federates */
    struct Region_List *xtReg; /* list of regions */
    struct Unit_Characteristics *UnitGn;
    struct Unit_Characteristics *UnitOt;
    struct Unit_List *UnListGn;
    struct Unit_List *UnListOt;
} Region_Node_Handle_Type;

/* -- typedef Nodes_of_Fed_List          end -> self DocMethod          f*/
typedef struct Nodes_of_Fed_List {
double Ratio;
int     Initialized ;
int     CountInteractions ;
int     Interact[InteractionsInSOM] ;
int     Objects[ObjectsInSOM] ;
int     NodeId; /* Node id */
int     NumGn ;
int     NumOt ;
int     box[4][2];
    struct Unit_Region_List *RegOnNode;
    struct Units_on_Node_List *UnitOnNode; /* Units owned by node */
    struct Nodes_of_Fed_List *xtNodeOfFed; /* next Federate node */
    char Name[Serv_Nam_M];
} Node_Region_List_Type ;

```



```

/* -- typedef Units_on_Node_List          end -> self DocMethod      f*/
typedef struct Units_on_Node_List {
    struct Unit_Characteristics      *UChrp ;    /* on to Unit characteristics */
    struct Units_on_Node_List      *xtUnitOnNode ; /* next element of list */
} Units_on_Node_List_Type ;

/* -- typedef Nodes_wrt_Region_List      end -> self DocMethod      f*/
typedef struct Nodes_wrt_Region_List {
    int boundary; /* double word boundary */
    int NodeId; /* Node id */
    struct Nodes_of_Fed_List *NodeOfFed; /* p to node of federate */
    struct Nodes_wrt_Region_List *xtNodeWRTRegion; /* next node; if any */
} Node_of_Region_List_Type ;

/* -- typedef Region_List          end -> self DocMethod      f*/
typedef struct Region_List {
    int Id ;
    int Category; /* category H M L */
    int box[4][2];
    int SubInteract[InteractionsInSOM] ;
    int PubInteract[InteractionsInSOM] ;
    int SubObjects[ObjectsInSOM] ;
    int PubObjects[ObjectsInSOM] ;
    int EquipGn ; /* actual equipment in Gn */
    int EquipOt ; /* " Ot */
    struct Nodes_wrt_Region_List *NodeWRTRegion ; /* nodes where region resides */
    struct Region_Element_List *xtGnRegEle ; /* next element for this region */
    struct Region_Element_List *xtOtRegEle ; /* next element for this region */
    struct Region_List *xtReg ; /* next region */
    char Name[Serv_Nam_M];
} Region_List_Type ;

/* -- typedef Region_Element_List      end -> self DocMethod      f*/
typedef struct Region_Element_List {
    int boundary; /* double word boundary */
    int Id;
    struct Unit_Characteristics *UChrp ; /* on to Unit characteristics */
    struct Region_Element_List *xtEle ; /* next element for this region */
} Region_Element_List_Type ;

/* -- typedef Unit_Region_List          end -> self DocMethod      f*/
typedef struct Unit_Region_List {
    struct Region_List *xtReg ; /* next element for this region */
    struct Unit_Region_List *nxtRegOfUnit ; /* next region unit is in */
} Unit_Region_List_Type ;

/* -- typedef Filter_Unit_List          DocMethod      f*/
typedef struct Filter_Unit_List {
    struct Unit_Characteristics *UChrp ;
    struct Filter_Unit_List *nxtFilteredUnitp; /* next ring element (Same Echeleon) */
} Filter_Unit_List_Type ;

/* comms go direct by passing the ptr as the id in the message queue */
/* if Id = 0 then this unit's character is rolled up to a superior */
/* -- typedef Unit_Characteristics          DocMethod      f*/
typedef struct Unit_Characteristics {
    double LastTime ;
    double NextTime ;
    double ReportRate ;
    double OrderRate ;
    double FireRate ;
    double SenseRate ;
    double MoveRate ;
    double Environment ;

```

```
typedef struct Network Route {
```

30 June 1999

```

int          Indx;          /* current point on the route */
int          Direction ;    /* increment of Indx */
struct Network_Node  NwNode[Ntw_M] ;
struct Network_Route *nlep;
char         Name[Serv_Nam_M];
} Network_Route_Type;

/* -- typedef Order_Packet f*/
/*   TimeToSend,TimeToAct,Order,Force,To, sendhrs,acthrs,nextTo,nextTohrs */

typedef struct Order_Packet {
double      Size ;          /* bytes */
double      Entities ;      /* Entities affected */
double      TimeToSend;     /* secs */
double      TimeToAct ;     /* secs */
double      ToActInHrs;     /* */
double      TravelStart;    /* time when message started */
double      TimeOnQueue;    /* time when message was put on queue*/

int         State;          /* Encrypted == 0 , normal Data == 1 */
int         Semaphore;      /* what it is */
int         PcktType;       /* # CmdOrder, DataPacket & more...*/
int         Force ;        /* # Friends | Others */
int         Order ;        /* CBS order | Task id number */
int         Activity ;     /* level | services (for future) */
int         To ;           /* Unit Id (Cmd | Comm) (data) */
int         From ;         /* Unit Id | Cpu Id */
int         Priority ;     /* for semaphore & selection */
int         XferType ;     /* Cs ->SP, Cs->Cs, Cs->Cpi etc...*/
int         McNet ;        /* Multicast Net */
int         DestLocale ;
int         DestSP ;       /* SPId Comm Packet (Comm ) */
int         DestCS ;
int         DestCpi ;
int         DestDVS ;
int         OrigLocale ;
int         OrigSP ;       /* SPId Comm Packet (Comm ) */
int         OrigCS ;
int         OrigCpi ;
int         OrigDVS ;
int         MsgId ;        /* SES likes double boundaries */
struct Truth_Group_List *PvTruthp ; /* Of Unit on Reception */
struct Order_Packet *nqep ; /* next queue element NULL terminated */
} Order_Packet_Type ;

/* -- typedef Task_List f*/
typedef struct Task_List {
double      Brigade ;
double      Battalion ;
double      Company ;
double      Platoon ;
int         Code ;         /* the task code */
int         Mode ;         /* 1:parallel | 2:sequential */
int         Activity ;     /* 1(high),2, or 3(low) or 0(select) */
int         ChainCode;    /* to other task (=> Periodicp == NULL)*/
struct Serv_List *AppListp;
struct Serv_List *Periodicp;
struct Task_List *ngep ;
char         Name[Serv_Nam_M] ;
} Task_List_Type ;
/* Active_Task(s):[id, Task_ptr, ServList_ptr] Defined_Task(s) */
/* Application Tasks & Periodic Service lists */

/* -- typedef Serv_Characteristics f*/
typedef struct Serv_Characteristics {
double      Cpu_Time ;
double      RAM_Size ;

```

30 June 1999

```

double      Dsk_Size ;
double      By_Entity ;          /* multiplier for number of entities */
double      Epsilon ;
double      IOPercent ;
int         FuncType ;
int         Interruptable ;      /* 1 is yes */
int         IO_needed ;
unsigned int FurtherEncoding; /* maybe */
int         Category_Index ;
struct      Serv_Characteristics *ngep ; /* next group element */
char        Name[Serv_Nam_M] ;
} Serv_Characteristics_Type ;

/* -- typedef Serv_List f*/
typedef struct Serv_List {
    struct Task_List      *TskLstp ; /* association */
    struct Serv_Characteristics *SChrp;
    struct Serv_List      *nlep;
    char                  Name[Serv_Nam_M] ;
} Serv_List_Type ;

/* -- typedef Serv_Stack f*/
typedef struct Serv_Stack {
    double      Stack ;          /* may not use */
    struct Serv_List      *SvLstp ;
    struct Serv_Stack      *nskp ;
} Serv_Stack_Type ;

/* -- typedef Sim_Factor f*/
typedef struct Sim_Factor {
    double      Max;
    double      Fuzzy ; /* Max - (Max * Fuzzy) for region of distn */
    int         DistnCode;
    int         NewValue;
} Sim_Factor_Type ;

/* -- typedef Sim_Management f*/
typedef struct Sim_Management {
    struct Sim_Factor      Time_To_Apply;
    struct Sim_Factor      SimInterval; /* interpret as delta from now */
    struct Sim_Factor      ConnectATM ;
    struct Sim_Factor      SizeATMXfer ;
    struct Sim_Factor      TimeATMXfer ;
    struct Sim_Factor      RAM;
    struct Sim_Factor      CPU;
    struct Sim_Factor      DeadReckoning;
    struct Sim_Factor      LOS;
    struct Sim_Factor      EffectSubscribed;
    struct Sim_Factor      EffectReflect;
    struct Sim_Factor      TimeSubscribed;
    struct Sim_Factor      TimeReflect;
    struct Sim_Factor      FreqComm; /* used to reset maximum */
    struct Sim_Factor      SizeComm; /* used to reset size */
    struct Sim_Factor      PlanDevelopment;
    struct Sim_Factor      LevelComm; /* switch on maximum unit to unit reporting */
    struct Sim_Factor      LevelUnit;
    int         NewSimParameters;
    char        SetupName[128];
} Sim_Management_Type ;

/* -- typedef State_Terms f*/
/* in SES each transaction has it's own one of the following */

typedef struct State_Terms {
    double      CycleStart; /* ctime of a period cycle start */
    double      CycleEnd; /* ctime of interruption or completion */
    double      ServTime; /* computed delays */
    double      CmlServTime; /* cumulative time 0.0 if no interrupts */

```

30 June 1999

```

double      Expire;      /* time that rate process expires */
double      UnitCnt;     /* just count the units for the cycle */
double      MemQuanta;
double      IoAvailCells ;
double      IoQuanta;
double      IoTime;
double      DiskQuanta;
double      CpuAccessTime;
double      XferTime;
double      DeadReckEffect; /* computed from Movement Task */
double      ReflectTime ; /* Time to prepare reflections */
double      SubscribeTime ; /* Time to update reflections */
double      ToReflect ; /* Number reflected from this cpu */
double      MemTime;     /* RAM or Disk access */
double      RR_Period;    /* Round Robin Ratio */
double      ServThisSlice; /* proportion of time slice */
double      MsgStartAt;  /* collect frequency */
double      MsgRecvdAt;
double      IoSent;      /* collect rate */
double      IoRcvd;
int         CsId;
int         CpiId;
int         Locale;
int         CpuIndex;
int         ServSet;     /* 0 no 1 yes; 0 => idle */
int         IdProc;
int         ServId;      /* index to service or method of unit task */
int         BlockId;     /* index into a block array to use */
struct Node_Table_Def *NodTblp;
struct Order_Packet *nOrder; /* messages */
struct Truth_Group_List *NodTblPvTp;
struct Truth_Group_List *cPvTp; /* for this cpu */
struct Serv_Characteristics *ServCharp;
struct Serv_List *cSLstp; /* top of the unit's current ServList */
int         IoCells ;    /* use will vary */

} State_Terms_Type ;
/* DocMethod */
/* statsmgr.h */
/* file: statsmgr.h */

```

```

#define MAX_STATISTIC_LABEL_CHARS 15
#define MAX_SVC_NAME_LEN 30
#define NBR_SAMPLES_IN_REPLICATE 6
/* sample types */
#define MIN_STAT 0
#define MAX_STAT 1
#define SMPL_CUM_TOTL 2
#define MEAN_STAT 3
#define VAR_CUM_TOTL_SEC_TERM 4
#define VARIANCE_STAT 5

typedef struct statistic_CPM {
    char      stat_label_name[MAX_SVC_NAME_LEN];
    int       statsarray_index;
} STATISTIC_CPM_TYPE;

typedef struct stats_data {
    char      stat_label_name[MAX_SVC_NAME_LEN];
    int       samples_count; /* nbr of samples so far */
    double    samples[NBR_SAMPLES_IN_REPLICATE]; /* collect 6 samples = 1 replicate */
}

```

30 June 1999

```

double      samples_totals[4];          /* min, max,
                                         cum samples sum,
                                         mean */
long int    replicates_count;           /* nbr of replicates so far */
double      replicates_totals[6];       /* totals of the sample replicates */
double      d;                         /* std deviation */
int         converger;                  /* true or false */
} STATS_DATA_TYPE;

/* statsmgr constant times, make into functions in future */
#define statsmgr_constraints_verification 0.00051
#define statsmgr_setup_events            0.0003
/* start advance */
#define statsmgr_start_LBTS              0.001
/* report */
#define statsmgr_accumulate_LBTS_info     0.0012
#define statsmgr_forward_LBTS_info        0.001
/* grant */
#define statsmgr_advance_LBTS             0.001
/* query */
#define statsmgr_retrieve_local_LBTS_info 0.002

extern int  initialize_statistic(STATISTIC_CPM_TYPE *stat_entry,
                                char *svcname,
                                int  federate_nbr );

extern int  statsmgr_get_statsarray_index(int      action,
                                           int      federate_nbr);

extern void statsmgr_collect_statistic(int      stat_entry_index,
                                       double     sample_time);

extern void statsmgr_print_accum_totals();

extern double statsmgr_lookup_in_fomdb(int  nbr_federates);
extern double statsmgr_lookup_in_fedexdb(int  nbr_federates);
extern double statsmgr_setup_fed_in_fedexdb(int  nbr_federates);
extern double statsmgr_setup_region_in_fedexdb(int  nbr_federates);
extern double statsmgr_setup_class_in_fedexdb(int  nbr_federates);
extern double statsmgr_setup_instance_in_fedexdb(int  nbr_federates);
extern double statsmgr_forward_to_sim_model(int  nbr_federates);
extern double statsmgr_compute_nbr_nodes(int  nbr_federates);
extern void   statsmgr_clear_accum_totals();

extern void statsmgr_init_statruns_file();
extern void statsmgr_cleanup();

```

