TRANSIMS Simulation Output Subsystem for IOC-1

K. P. Berkbigler
Computer Research and Applications Group
Los Alamos National Laboratory
and
B. W. Bush
Energy and Environmental Analysis Group
Los Alamos National Laboratory

24 April 1997

Abstract

The output subsystem collects data from a running microsimulation, stores the data for future use, and manages the subsequent retrieval of the data. It forms a layer separating the other subsystems from the actual data files so that the other subsystems do not need to access the data files at the physical level or deal with the physical location and organization of the files. This subsystem also allows the user to specify what data is collected and retrieved, and to filter it by space and time. The collection occurs in a distributed manner such that the subsystem's impact on the microsimulation performance is minimized; the retrieval provides a unified view of the distributed data.

I. Introduction 5
II. Design 7
A. Concepts 7
1. Types of Data Collection 7
2. Filtering 8
B. Classes 8
1. TOutDispatcher 11
2. TOutFactory 12
3. TOutGeneralSpecification 13
4. TOutGeneralSpecificationReader 14
5. TOutSpecificationReader 15
6. TOutProcessor 16
7. TOutEvolutionProcessor 16
8. TOutEventProcessor 18
9. TOutSummaryProcessor 19
10. TOutObserver 20
11. TOutVehicleObserver 20
12. TOutNodeEvolutionObserver 21

13. TOutLinkEvolutionObserver 21
14. TOutSignalCoordinatorEvolutionObserver 21
15. TOutSignalizedControlObserver 21
16. TOutIntersectionObserver 22
17. TOutLinkSpaceObserver 22
18. TOutLinkTimeObserver 23
19. TOutRetriever 23
20. TOutEvolutionRetriever 24
21. TOutEventRetriever 24
22. TOutSummaryRetriever 24
23. TOutWriter 24
24. TOutTextWriter 25
25. TOutStorage 25
26. TOutRecord 26
27. TOutException 28
III. Implementation 28
A. C++ Libraries 28
B. File System 29
C. Integration into the Microsimulation 30
IV. Usage 31
A. Specification Formats 31
1. Output Specification 31
2. Output Node Specification 33
3. Output Link Specification 33
B. Data Retrieval 34
C. Output Formats 35
1. Evolution Data 35
2. Event Data 36
3. Summary Data 36
D. Example of Retrieval Using C++ 37
E. Notes 38
1. Database Setup 38
2. Empty Storage Files 39
V. Future Work 39
VI. References 39
VII. APPENDIX: Booch Notation Diagrams 39
VIII. APPENDIX: Source Code 43
A. TOutDispatcher Class 43
1. Dispatcher.h 43
2. Dispatcher.C 45
B. TOutEventProcessor Class 51
1. EventProcessor.h 51
2. EventProcessor.C 52
C. TOutEventRetriever Class 53
1. EventRetriever.h 53

2. EventRetriever.C 53
D. TOutEvolutionProcessor Class 54
1. EvolutionProcessor.h 54
2. EvolutionProcessor.C 56
E. TOutEvolutionRetriever Class 59
1. EvolutionRetriever.h 59
2. EvolutionRetriever.C 59
F. TOutException Class 60
1. Exception.h 60
2. Exception.C 61
G. TOutFactory Class 62
1. Factory.h 62
2. Factory.C 63
H. TOutGeneralSpecification Class 64
1. GeneralSpecification.h 64
2. GeneralSpecification.C 65
I. TOutGeneralSpecificationReader Class 67
1. GeneralSpecificationReader.h 67
2. GeneralSpecificationReader.C 69
J. TOutIntersectionObserver Class 72
1. IntersectionObserver.h 72
2. IntersectionObserver.C 73
K. TOutLinkEvolutionObserver Class 74
1. TOutLinkEvolutionObserver.h 74
2. TOutLinkEvolutionObserver.C 74
L. TOutLinkSpaceObserver Class 75
1. LinkSpaceObserver.h 75
2. LinkSpaceObserver.C 76
M. TOutLinkTimeObserver Class 78
1. LinkTimeObserver.h 78
2. LinkTimeObserver.C 79
N. TOutNodeEvolutionObserver Class 80
1. NodeEvolutionObserver.h 80
2. NodeEvolutionObserver.C 80
O. TOutObserver Class 81
1. Observer.h 81
2. Observer.C 82
P. TOutProcessor Class 83
1. Processor.h 83
2. Processor.C 84
Q. TOutRecord Class 85
1. Record.h 85
2. Record.C 87
R. TOutRetriever Class 90
1. Retriever.h 90

2. Retriever.C 91
S. TOutSignalCoordinatorEvolutionObserver Class 93
1. SignalCoordinatorEvolutionObserver.h 93
2. SignalCoordinatorEvolutionObserver.C 93
T. TOutSignalizedControlObserver Class 94
1. SignalizedControlObserver.h 94
2. SignalizedControlObserver.C 95
U. TOutSpecificationReader Class 95
1. SpecificationReader.h 95
2. SpecificationReader.C 96
V. TOutStorage Class 97
1. Storage.h 97
2. Storage.C 99
W. TOutSummaryProcessor Class 103
1. SummaryProcessor.h 103
2. SummaryProcessor.C 105
X. TOutSummaryRetriever Class 107
1. SummaryRetriever.h 107
2. SummaryRetriever.C 107
Y. TOutTextWriter Class 108
1. TextWriter.h 108
2. TextWriter.C 109
Z. TOutVehicleObserver Class 110
1. VehicleObserver.h 110
2. VehicleObserver.C 111
AA. TOutWriter Class 112
1. Writer.h 112
BB. Type Definitions 113
1. Id.h 113
2. Names.h 113
IX. APPENDIX: Test Program 114
A. Test.C 114
B. TestEvolutionRetriever.C 115
C. TestGeneralSpecification.C 117
D. TestRecord.C 119
E. TestStorage.C 121
F. TestTextWriter.C 123
G. TestSpecification.import 125
X. APPENDIX: Source Creation Utility 125
A. CreateSources.C 125
XI. APPENDIX: Storage Dump Utility 126
A. DumpStorage.C 126

I. Introduction

The output subsystem collects data from a running microsimulation, stores the data for future use, and manages the subsequent retrieval of the data. It forms a layer separating the other subsystems from the actual data files so that the other subsystems do not need to access the data files at the physical level or deal with the physical location and organization of the files. Figure 1 shows the position of the simulation output subsystem within the TRANSIMS software architecture. This subsystem only depends on the database subsystem (strongly) and the network subsystem (weakly) and is not tied to the specific design used for the IOC-1 microsimulation; this opens the possibility to reuse it in other TRANSIMS traffic simulations.

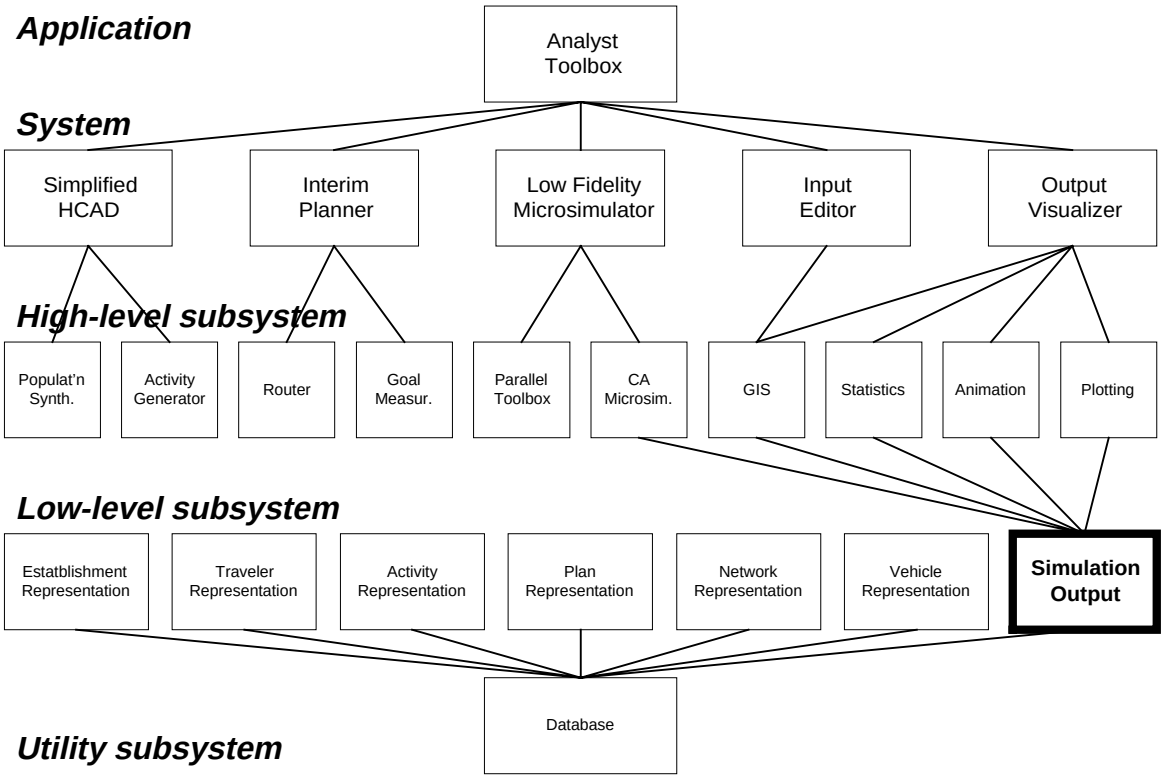

Figure 1. Location of the simulation output subsystem in the TRANSIMS software architecture.

This subsystem also allows the user to specify what data is collected and retrieved, and to filter it by space and time. Users can configure the subsystem to collect a wide variety of trajectory, event, and summary data from the simulation. Figure 2 shows an example of how data collection can be configured. The data can be accessed in binary format via a direct connection to the subsystem or in a delimited-text format for off-line postprocessing.

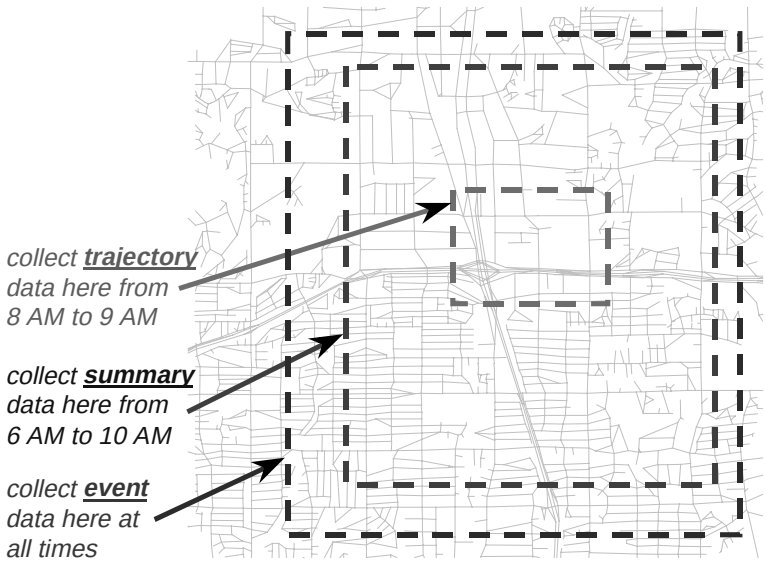

Figure 2. Example of how data collection can be filtered by space and by time.

The subsystem has been put to a wide variety of uses in the first TRANSIMS case study. Evolution data was used for animating vehicle movement, making periodic snapshots of the traffic, understanding the traffic behavior induced by CA (cellular automaton) microsimulation rules, refining the driving logic, and deriving fundamental diagrams. Event data has helped to locate problems with network data, driver logic, and plans, and to record the entry and exit times of vehicles in and out of the study area. Summary data provided a means to animate vehicle densities, identify congestion and deadlocks, and replan trips using observed link travel times.

The collection occurs in a distributed manner such that the impact of the subsystem on the microsimulation performance is minimized: Although the simulation output subsystem runs on multiple computational nodes (CPNs) during data collection, it does not require any communication between the CPNs. This leaves the full communication bandwidth available for use by the simulation proper. The retrieval, on the other hand, provides a unified view of the distributed data by coordinating the retrieval of data from remote file systems. Figure 3 illustrates the dual uses of the subsystem.

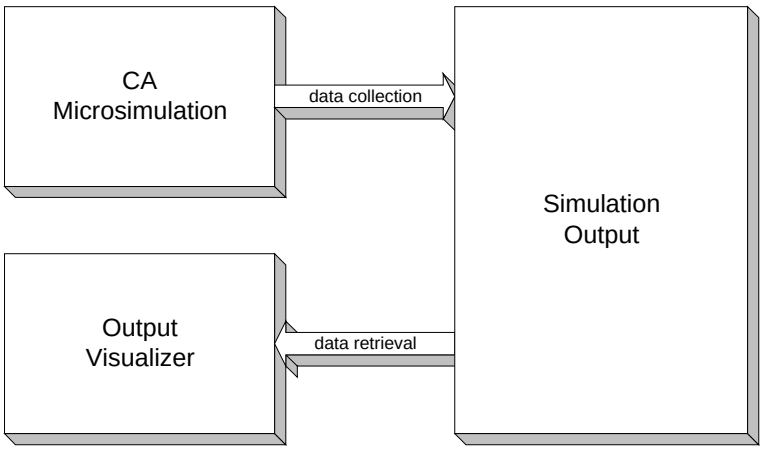

Figure 3. Dual uses of the simulation output subsystem.

The body of this document outlines the design, implementation, and usage of the subsystem. The appendices contain the complete C++ source code for the subsystem.

II. Design

A. Concepts

1. Types of Data Collection

The output subsystem currently can collect three types of data: evolution (trajectory) data, event data, and summary data. Any number of each of these may be collected simultaneously in a simulation.

Evolution data provides the most detailed information about how the state of the microsimulation evolves in time. The vehicle data for links consists of the location, velocity, and status of each vehicle; this provides a complete "trajectory" for each vehicle in the simulation. The vehicle data for intersections consists of the location of the vehicle within the intersection buffer. The traffic control data simply reports the current phase and allowed movements at the traffic control. Evolution data may be collected for each time step; the data is not summarized (i.e., totaled or averaged) in any way.

Event data supplies information on exceptional conditions of vehicle status. Examples include when a vehicle becomes lost (unable to follow its plan), when the plan for a vehicle is invalid, and when the vehicle enters or exits the study area. Event data is collected only when an event occurs.

Summary data reports aggregate data about the simulation. The link travel time data consists of counts of vehicles exiting links and means and variances of the vehicle traversal times for those links. Link density data provides counts and mean velocities of vehicles in variably-sized boxes that partition links. Summary data is sampled and reported periodically throughout the simulation.

2. Filtering

The simulation output subsystem has the capability to collect data on any subset of nodes and links in the road network (Figure 2). It is also possible to set the starting and ending times within which data collection occurs (also Figure 2). A user can also specify the frequency of reporting for evolution and summary data and the sampling frequency (i.e., the frequency at which the data is observed) for summary data. The space and time filtering can be different for each type of data.

B. Classes

The simulation output subsystem has classes containing domain knowledge and classes forming a domain-independent data management layer (Figure 4). Figure 5, Figure 6, and Figure 7, show the relationships between classes used for evolution, event, and summary data collection, respectively; Figure 8 shows the relationships between classes used for data retrieval.

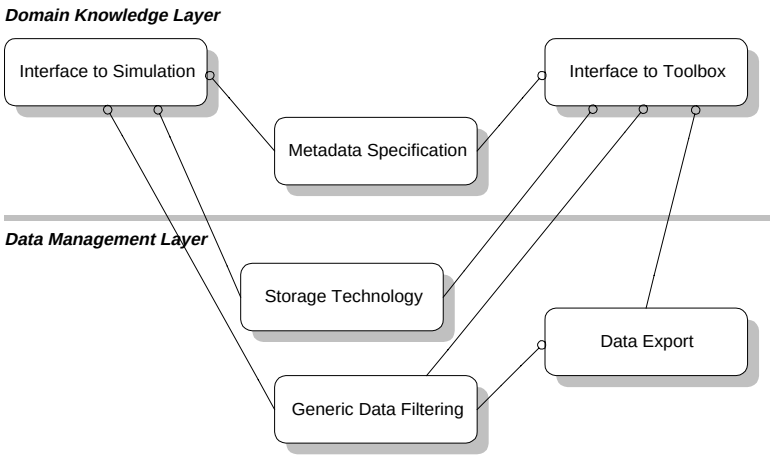

Figure 4. Categories of classes in the TRANSIMS simulation output subsystem.

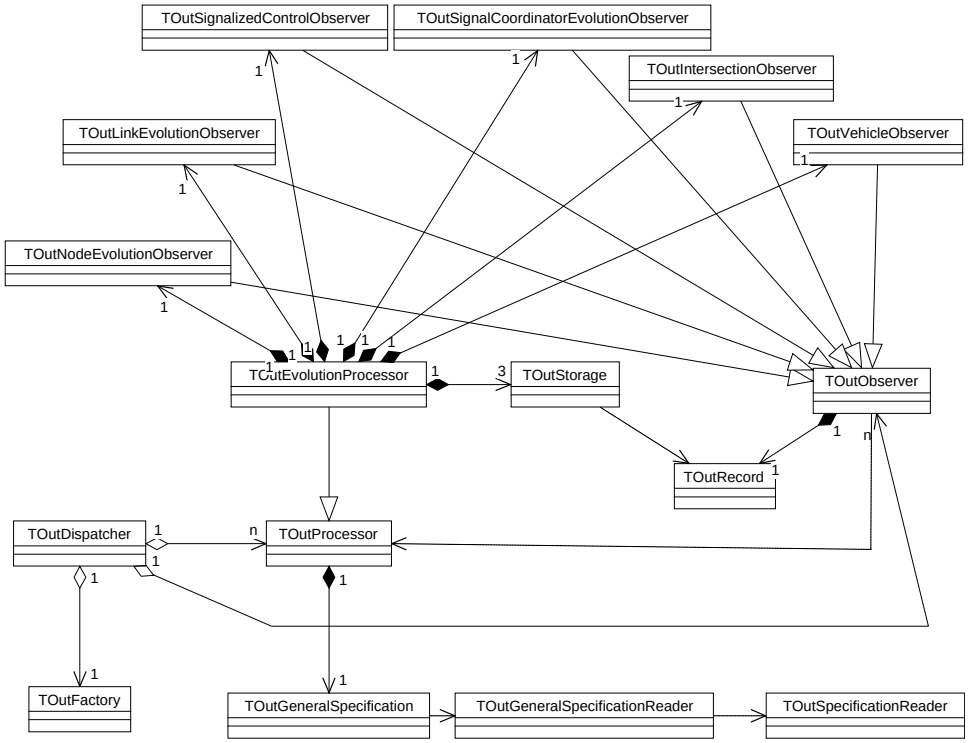

Figure 5. Class diagram for the TRANSIMS simulation output subsystem classes involved in evolution data collection (unified notation).

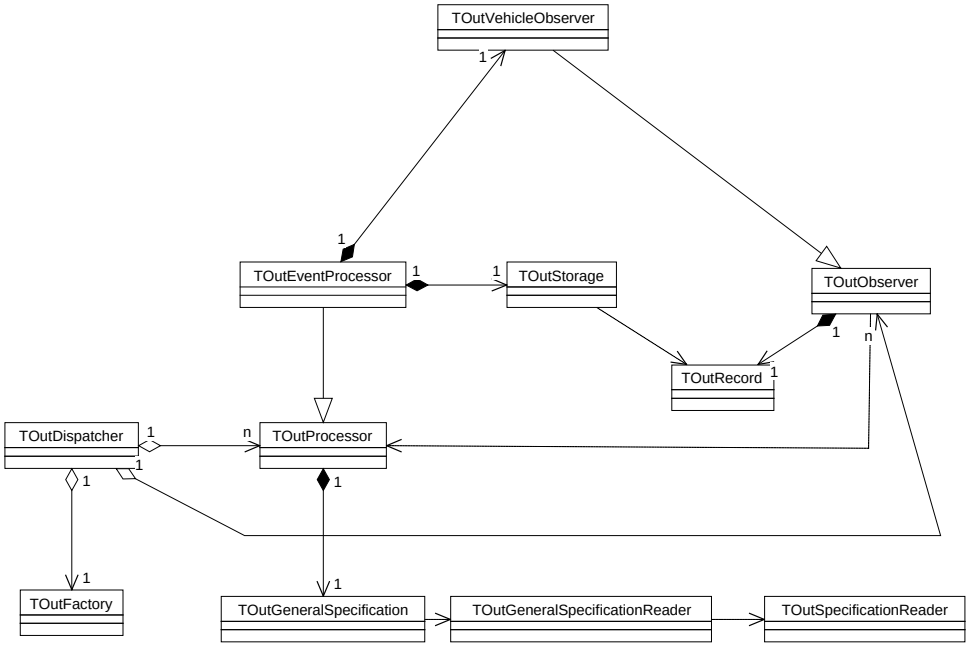

Figure 6. Class diagram for the TRANSIMS simulation output subsystem classes involved in event data collection (unified notation).

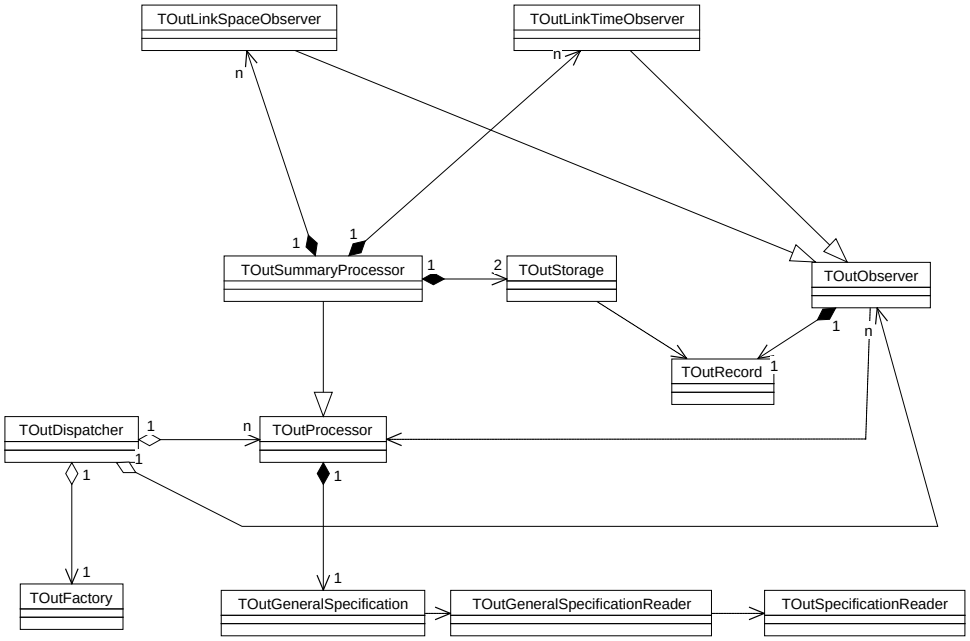

Figure 7. Class diagram for the TRANSIMS simulation output subsystem classes involved in summary data collection (unified notation).

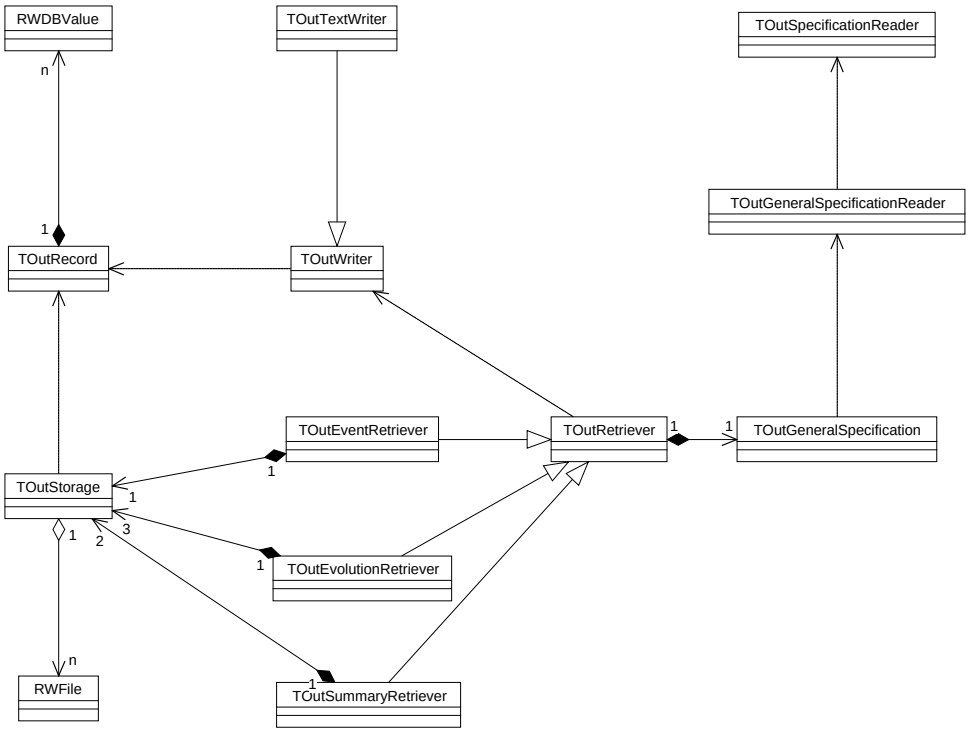

Figure 8. Class diagram for the TRANSIMS simulation output subsystem classes involved in data retrieval (unified notation).

1. TOutDispatcher

An output dispatcher coordinates the construction of simulation output objects and supervises the transfer of data. Each dispatcher has processors and observers; it also remembers the factory it uses to construct objects.

```
enum EObserverType {kNodeEvolutionObserver,
           kLinkEvolutionObserver, kVehicleObserver,
           kIntersectionObserver,
           kSignalCoordinatorEvolutionObserver,
           kSignalizedControlObserver, kLinkSpaceObserver,
           kLinkTimeObserver}
```

Observer types.

```
TOutDispatcher(TOutSpecificationReader& reader, TOutFactory&
           factory)
```

Construct an output dispatcher.

```
bool operator==(const TOutDispatcher& dispatcher) const
bool operator!=(const TOutDispatcher& dispatcher) const
```

Return whether two output dispatchers are the same.

```
void RecordOutput(REAL time)
```

Begin output recording for this time step.

```
ProcessorMap& GetProcessors()
const ProcessorMap& GetProcessors() const
```

Return the processors.

```
ObserverMap& GetObservers()
const ObserverMap& GetObservers() const
```

Return the observers.

```
EvolutionProcessorSet& GetEvolutionProcessors()
const EvolutionProcessorSet& GetEvolutionProcessors() const
```

Return the evolution processors.

```
EventProcessorSet& GetEventProcessors()
const EventProcessorSet& GetEventProcessors() const
```

Return the event processors.

```
SummaryProcessorSet& GetSummaryProcessors()
const SummaryProcessorSet& GetSummaryProcessors() const
```

Return the summary processors.

```
void SetNodeObservers(TOutObserver::ObserverMap& observers)
```

Define the node observers.

```
void SetLinkObservers(TOutObserver::ObserverMap& observers)
```

Define the link observers.

```
void SetVehicleObservers(TOutObserver::ObserverMap& observers)
```
Define the vehicle observers.

```
void SetIntersectionObservers(TOutObserver::ObserverMap&
        observers)
```
Define the intersection observers.

```
void SetSignalCoordinatorObservers(TOutObserver::ObserverMap&
        observers)
```
Define the signal coordinator observers.

```
void SetSignalizedControlObservers(TOutObserver::ObserverMap&
        observers)
```
Define the signalized control observers.

```
void SetLinkSpaceObservers(TOutObserver::ObserverMap& observers)
```
Define the link space observers.

```
void SetLinkTimeObservers(TOutObserver::ObserverMap& observers)
```
Define the link time observers.

```
void ClearLinkSpaceObservers(TOutObserver::ObserverMap&
        observers)
```
Undefine the link space observers.

```
void ClearLinkTimeObservers(TOutObserver::ObserverMap& observers)
```
Undefine the link time observers.

2. TOutFactory

An output factory allocates and constructs new simulation output objects.

```
TOutFactory()
```
Construct a factory.

```
virtual TOutEvolutionProcessor*
        NewEvolutionProcessor(OutProcessorId id, const
        TOutGeneralSpecification& specification)
```
Return a new evolution processor from the specification.

```
virtual TOutEventProcessor* NewEventProcessor(OutProcessorId id,
        const TOutGeneralSpecification& specification)
```
Return a new event processor from the specification.

```
virtual TOutSummaryProcessor* NewSummaryProcessor(OutProcessorId
        id, const TOutGeneralSpecification& specification)
```
Return a new summary processor from the specification.

```
virtual TOutLinkEvolutionObserver*
            NewLinkEvolutionObserver(OutObserverId id)
```

Return a new link evolution observer.

```
virtual TOutVehicleObserver* NewVehicleObserver(OutObserverId id)
```

Return a new vehicle observer.

```
virtual TOutNodeEvolutionObserver*
            NewNodeEvolutionObserver(OutObserverId id)
```

Return a new node evolution observer.

```
virtual TOutIntersectionObserver*
            NewIntersectionObserver(OutObserverId id)
```

Return a new intersection observer.

```
virtual TOutSignalCoordinatorEvolutionObserver*
            NewSignalCoordinatorEvolutionObserver(OutObserverId
            id)
```

Return a new signal coordinator evolution observer.

```
virtual TOutSignalizedControlObserver*
            NewSignalizedControlObserver(OutObserverId id)
```

Return a new signalized control observer.

```
virtual TOutLinkSpaceObserver* NewLinkSpaceObserver(OutObserverId
            id)
```

Return a new link space observer.

```
virtual TOutLinkTimeObserver* NewLinkTimeObserver(OutObserverId
            id);
```

Return a new link time observer.

3. TOutGeneralSpecification

The general specification defines the frequency and extent of data to be collected or retrieved in both space and time. Each specification has a root, a name, a minimum time, a maximum time, a time step, a time sample, a box length, a collection region, a set of node ids, and a set of link ids.

```
static const REAL kMinusInfinity
static const REAL kPlusInfinity
```

Time constants.

```
TOutGeneralSpecification(TOutGeneralSpecificationReader& reader)
```

Construct a general specification from a reader.

```
const string& GetRoot() const
```

Return the root for the specification.

`const string& GetName() const`
Return the name for the specification.

`bool CollectForTime(REAL time) const`
Return whether data should be collected for the specified time.

`bool SampleForTime(REAL time) const`
Return whether data should be sampled at the specified time.

`bool IsAtStartTime(REAL time) const`
Return whether the specified time is the start time.

`REAL GetBoxLength() const`
Return the box length.

`bool CollectForPoint(const TGeoPoint& point) const`
Return whether data should be collected for the specified point in space.

`bool CollectForNode(NetNodeId id) const`
Return whether data should be collected for the specified node.

`bool CollectForLink(NetLinkId id) const`
Return whether data should be collected for the specified link.

4. TOutGeneralSpecificationReader

A general specification reader reads specifications from the database. Each general specification reader has database table accessors for the specification and for node and link tables.

```
enum EProcessorType {kEvolutionProcessor, kEventProcessor,
          kSummaryProcessor}
```
Processor types.

`TOutGeneralSpecificationReader(TOutSpecificationReader& reader)`
Construct a general specification reader.

```
TNetGeneralSpecificationReader(const
          TNetGeneralSpecificationReader& reader)
```
Construct a copy of the given general specification reader.

```
TNetGeneralSpecificationReader& operator=(const
          TNetGeneralSpecificationReader& reader)
```
Make the reader a copy of the given general specification reader.

`void Reset()`
Reset the iteration over the table.

`void GetNextSpecification()`
Get the next specification in the table.

`bool MoreSpecifications() const`
Return whether there are any more specifications in the table.

`string GetRoot() const`
Return the root of the specification.

`string GetName() const`
Return the name of the specification.

`REAL GetMinimumTime() const`
Return the minimum time of the specification.

`REAL GetMaximumTime() const`
Return the maximum time of the specification.

`REAL GetTimeStep() const`
Return the time step of the specification.

`REAL GetTimeSample() const`
Return the time sampling of the specification.

`REAL GetBoxLength() const`
Return the box length of the specification.

`TGeoRectangle GetRegion() const`
Return the geographic region of the specification.

`NodeIdSet GetNodes() const`
Return the nodes in the specification.

`LinkIdSet GetLinks() const`
Return the links in the specification.

`EProcessorType GetProcessorType() const`
Return the processor type in the specification. A `TOutInvalidProcessor` exception is thrown if the processor is not a valid type.

5. TOutSpecificationReader

A specification reader reads an output specification from the database. Each reader has a general specification table, a node specification table, and a link specification table.

```
TOutSpecificationReader(TDbTable generalTable, TDbTable
            nodeTable, TDbTable linkTable)
```
Construct a reader for the specified tables.

`TDbTable& GetGeneralTable()`
Return the general specification table.

`TDbTable& GetNodeTable()`
Return the node specification table.

`TDbTable& GetLinkTable()`
Return the link specification table.

6. TOutProcessor

An output processor coordinates the processing of domain information into a domain-independent representation that is filtered and summarized before storage. A processor has an id and a general output specification; the class keeps track of next available processor id.

```
enum EProcessorType {kEvolutionProcessor, kEventProcessor,
          kSummaryProcessor}
```
Processor types.

```
TOutProcessor(OutProcessorId id, const TOutGeneralSpecification&
          specification)
```
Construct a processor.

`virtual EProcessorType GetProcessorType() const`
Return the processor type.

`virtual void RecordOutput(REAL time)`
Begin recording output for this time.

```
OutProcessorId GetId()
const OutProcessorId GetId() const
```
Return the processor's id.

`static OutProcessorId GetNextId()`
Return the next unused processor id.

```
TOutGeneralSpecification& GetGeneralSpecification ()
const TOutGeneralSpecification& GetGeneralSpecification() const
```
Return the general specification.

7. TOutEvolutionProcessor

An output evolution processor deals with evolving data such as that needed for animation, waterfall plots, etc. An evolution processor has a node observer, a link observer, a vehicle observer, an intersection observer, a signal coordinator observer, and a signalized control observer, and is connected to corresponding storage objects.

```
TOutEvolutionProcessor(OutProcessorId, const
           TOutGeneralSpecification& specification)
```

Construct an evolution processor.

```
EProcessorType GetProcessorType() const
```

Return the processor type.

```
TOutEvolutionSpecification& GetEvolutionSpecification()
const TOutEvolutionSpecification& GetEvolutionSpecification()
           const
```

Return the evolution specification.

```
virtual void RecordOutput(REAL time)
```

Begin recording output for this time step.

```
virtual void RecordNode()
```

Finish recording output for a node.

```
virtual void RecordLink()
```

Finish recording output for a link.

```
virtual void RecordVehicle()
```

Finish recording output for a vehicle.

```
virtual void RecordIntersection()
```

Finish recording output for an intersection.

```
virtual void RecordSignalCoordinator()
```

Finish recording output for a signal coordinator.

```
virtual void RecordSignalizedControl()
```

Finish recording output for a signalized control.

```
TOutObserver& GetNodeObserver()
const TOutObserver& GetNodeObserver() const
```

Return the node observer.

```
void SetNodeObserver(TOutObserver& observer)
```

Define the node observer.

```
TOutObserver& GetLinkObserver()
const TOutObserver& GetLinkObserver() const
```

Return the link observer.

```
void SetLinkObserver(TOutObserver& observer)
```

Define the link observer.

```
TOutObserver& GetVehicleObserver()
const TOutObserver& GetVehicleObserver() const
```

Return the vehicle observer.

```
void SetVehicleObserver(TOutObserver& observer)
```

Define the vehicle observer.

```
TOutObserver& GetIntersectionObserver()
const TOutObserver& GetIntersectionObserver() const
```

Return the intersection observer.

```
void SetIntersectionObserver(TOutObserver& observer)
```

Define the intersection observer.

```
TOutObserver& GetSignalCoordinatorObserver()
const TOutObserver& GetSignalCoordinatorObserver() const
```

Return the signal coordinator observer.

```
void SetSignalCoordinatorObserver(TOutObserver& observer)
```

Define the signal coordinator observer.

```
TOutObserver& GetSignalizedControlObserver()
const TOutObserver& GetSignalizedControlObserver() const
```

Return the signalized control observer.

```
void SetSignalizedControlObserver(TOutObserver& observer)
```

Define the signalized control observer.

8. TOutEventProcessor

An output event processor deals with conditions occurring in the simulation such as vehicle entry, vehicle exit, and lost vehicles. An event processor has a vehicle observer and a vehicle status mask and is connected to a vehicle storage.

```
TOutEventProcessor(OutProcessorId id, const
            TOutGeneralSpecification& specification, UINT
            vehicleMask)
```

Construct an event processor.

```
EProcessorType GetProcessorType() const
```

Return the processor type.

```
virtual void RecordOutput(REAL time)
```

Begin recording output for this time step.

```
virtual void RecordVehicle()
```

Finish recording output for a vehicle.

```
TOutObserver& GetVehicleObserver()
const TOutObserver& GetVehicleObserver() const
```
Return the vehicle observer.

```
void SetVehicleObserver(TOutObserver& observer)
```
Define the vehicle observer.

```
UINT GetVehicleMask() const
```
Return the vehicle status mask.

9. TOutSummaryProcessor

An output summary processor collects statistics on the simulation. A summary processor has space and time observers and is connected to corresponding storages.

```
TOutSummaryProcessor(OutProcessorId id, const
            TOutGeneralSpecification& specification)
```
Construct a summary processor.

```
EProcessorType GetProcessorType() const
```
Return the processor type.

```
virtual void RecordOutput(REAL time)
```
Begin recording output for this time step.

```
virtual void RecordSpace(const TOutObserver& observer)
```
Finish recording output for link space data.

```
virtual void RecordTime(const TOutObserver& observer)
```
Finish recording output for link time data.

```
ObserverSet& GetSpaceObservers()
const ObserverSet& GetSpaceObservers() const
```
Return the space observers.

```
ObserverSet& GetTimeObservers()
const ObserverSet& GetTimeObservers() const
```
Return the time observers.

```
void AddSpaceObserver(TOutObserver& observer)
```
Define a space observer.

```
void AddTimeObserver(TOutObserver& observer)
```
Define a time observer.

```
void RemoveSpaceObserver(TOutObserver& observer)
```
Undefine a space observer.

```
void RemoveTimeObserver(TOutObserver& observer)
```
Undefine a time observer.

10. TOutObserver

An observer converts data from the object to which it is attached into the generic form understood by the output subsystem. An observer has an id and an output record. The class keeps track of the next available observer id.

```
TOutObserver(OutObserverId)
```
Construct an observer.

```
virtual void Observe(const void* object, const TOutProcessor&
            processor)
```
An observer has an observe function for noting the values of data members of interest in the observed object. This virtual function in the basic representation must be overridden in the view.

```
OutObserverId GetId()
const OutObserverId GetId() const
```
Return the observer's id.

```
TOutRecord& GetRecord()
const TOutRecord& GetRecord() const
```
Return the associated record.

```
static OutObserverId GetNextId()
```
Return the next unused observer id.

```
void SetTime(REAL time)
```
Define the record's time.

11. TOutVehicleObserver

A vehicle observer observes data related to vehicles.

```
TOutVehicleObserver(OutObserverId id)
```
Construct a vehicle observer.

```
void SetId(UINT vehicle)
```
Define the vehicle's id.

```
void SetLink(NetLinkId link)
```
Define the id of the link the vehicle is on.

```
void SetLane(NetLaneNumber lane)
```
Define the lane number the vehicle is on.

```
void SetDistance(REAL distance)
```
Define the vehicle's distance from the node.

`void SetNode(NetNodeId node)`
Define the id of the node from which the distance is measured from.

`void SetVelocity(REAL velocity)`
Define the vehicle's velocity.

`void SetStatus(BYTE status)`
Define the vehicle's status.

12. TOutNodeEvolutionObserver

A node evolution observer observes evolving data related to nodes.

`TOutNodeEvolutionObserver(OutObserverId id)`
Construct a node evolution observer.

13. TOutLinkEvolutionObserver

A link evolution observer observes evolving data related to links.

`TOutLinkEvolutionObserver(OutObserverId id)`
Construct a link evolution observer.

14. TOutSignalCoordinatorEvolutionObserver

A signal coordinator evolution observer observes evolving data related to signal coordinators.

`TOutSignalCoordinatorEvolutionObserver(OutObserverId id)`
Construct a signal coordinator evolution observer.

15. TOutSignalizedControlObserver

A signalized control observer observes data related to signals.

`TOutSignalizedControlObserver(OutObserverId id)`
Construct a signalized control observer.

`void SetLink(NetLinkId link)`
Define the id of the link entering the intersection.

`void SetLane(NetLaneNumber lane)`
Define the lane number of the lane entering the intersection.

`void SetNode(NetNodeId node)`
Define the id of the node associated with the intersection.

`void SetSignal(TNetTrafficControl::ETrafficControl state)`
Define the current signal state.

16. TOutIntersectionObserver

An intersection observer observes data related to intersections.

`TOutIntersectionObserver(OutObserverId id)`
Construct an intersection observer.

`void SetId(UINT vehicle)`
Define the vehicle's id.

`void SetLink(NetLinkId link)`
Define the id of the link the vehicle entered from.

`void SetLane(NetLaneNumber lane)`
Define the lane number the vehicle entered from.

`void SetNode(NetNodeId node)`
Define the id of the node the intersection is associated with.

`void SetQIndex(BYTE index)`
Define the index of the vehicle's position in the queue.

17. TOutLinkSpaceObserver

A link space observer summarizes vehicle spatial data on a link. Each observer has a link length, a box length, and box data.

`TOutLinkSpaceObserver(OutObserverId id)`
Construct a link space observer.

`bool IsInitialized() const`
Return whether the observer has been initialized.

`void ClearBoxData()`
Clear the box data for the observer.

`void SetLink(NetLinkId id)`
Set the link id.

`void SetNode(NetNodeId id)`
Set the departure node id.

`void SetLengths(REAL linkLength, REAL boxLength, REAL cellLength)`
Set the link, box, and cell lengths.

`void AddVehicle(REAL distance, REAL velocity)`
Add a vehicle.

```
void ReportObservations(TOutProcessor& processor)
```
Report the observations to the specified processor.

18. TOutLinkTimeObserver

A link time observer records vehicle travel times on a link. Each link time observer has a vehicle count, a total of travel times, and a total of travel time squares.

```
TOutLinkTimeObserver(OutObserverId id)
```
Construct a link time observer.

```
void AddVehicle(REAL time)
```
Add a vehicle.

```
void ClearData()
```
Clear the data for the observer.

```
void SetLink(NetLinkId id)
```
Set the link id.

```
void SetNode(NetNodeId id)
```
Set the departure node id.

```
void ReportObservations(TOutProcessor& processor)
```
Report the observations to the specified processor.

19. TOutRetriever

An output retriever acts as an interface to coordinate the retrieval of data stored in a simulation. Each retriever has a general specification and may refer to a network.

```
virtual void Retrieve()
```
Perform the retrieval.

```
TOutRetriever(const TOutGeneralSpecification& specification,
            const TNetNetwork* network = NULL)
```
Construct a retriever based on the given specification and network.

```
TOutGeneralSpecification& GetGeneralSpecification()
const TOutGeneralSpecification& GetGeneralSpecification() const
```
Return the general specification.

```
const TNetNetwork* GetNetwork() const
```
Return the network, if any.

```
void BasicRetrieve(TOutStorage& storage, TOutWriter& writer, bool
            sort, bool filter = TRUE)
```
Retrieve data from the specified storage and put it in the specified writer, sorting it if indicated, and performing time and space filtering if indicated.

20. TOutEvolutionRetriever

An evolution retriever gets specific trajectory data from storage and coordinates its conversion and filtering. The retriever is connected to a vehicle storage, an intersection storage, and a signal storage.

```
TOutEvolutionRetriever(const TOutStorage::HostSet& hosts, const
            TOutGeneralSpecification& specification, const
            TNetNetwork* network = NULL)
```

Construct a reader for the specified hosts, given specification, and network.

```
void Retrieve(TOutWriter& vehicleWriter, TOutWriter&
            intersectionWriter, TOutWriter& signalWriter, bool
            sort = TRUE)
```

Perform the retrieval on the specified writers.

21. TOutEventRetriever

An event retriever gets specific event data from storage and coordinates its conversion and filtering. The retriever is connected to a vehicle storage.

```
TOutEventRetriever(const TOutStorage::HostSet& hosts, const
            TOutGeneralSpecification& specification, const
            TNetNetwork* network = NULL)
```

Construct a reader for the specified hosts, given specification, and network.

```
void Retrieve(TOutWriter& vehicleWriter, bool sort = TRUE)
```

Perform the retrieval on the specified writers.

22. TOutSummaryRetriever

An event retriever gets specific summary data from storage and coordinates its conversion and filtering. The retriever is connected to a space storage and a time storage.

```
TOutSummaryRetriever(const TOutStorage::HostSet& hosts, const
            TOutGeneralSpecification& specification, const
            TNetNetwork* network = NULL)
```

Construct a reader for the specified hosts, given specification, and network.

```
void Retrieve(TOutWriter& spaceWriter, TOutWriter& timeWriter,
            bool sort = TRUE)
```

Perform the retrieval on the specified writers.

23. TOutWriter

An output writer provides an interface for the external writing of data from simulation output. The exception `TOutWriterFailure` is thrown if an operation fails.

```
TOutWriter()
```

Construct a writer.

```
virtual void Write(const TOutRecord& record)
virtual void Write(const TOutRecord& record, const
            FieldCollection& fields)
```

Write the specified record.

24. TOutTextWriter

A text writer puts simulation output data into a formatted text file. The exception `TOutWriterFailure` is thrown if an operation fails. Each text writer is connected to a file and has a delimiter. A text writer may have to include header information in its output.

```
TOutTextWriter(const string& file, const string& delimiter =
            "\t", bool includeHeader = FALSE)
```

Open the specified file for writing, including the record header.

```
void Write(const TOutRecord& record)
void Write(const TOutRecord& record, const FieldCollection&
            fields)
```

Write the specified record with fields in the given order.

25. TOutStorage

An output storage manages the distributed file system and isolates the rest of the simulation output objects from the details of the physical storage. Member functions throw the exception `TOutStorageFailure` when errors occur. The class has a local host name. Each instance has a file suffix, a root location, a basic name, and a file on each host.

```
static const long kBegin
static const long kEnd
```

Constants for seek positions.

```
enum Mode {kRead, kWrite, kDelete}
```

Storage modes: Use read mode for opening existing files, write mode for creating a new file, and delete mode for deleting existing files.

```
TOutStorage(const string& root, const string& name, Mode mode =
            kRead)
TOutStorage(const HostSet& hosts, const string& root, const
            string& name, Mode mode = kRead)
```

Connect the storage with the given root and basic name to the specified hosts.

```
const string& GetRoot() const
```

Return the root name.

```
const string& GetName() const
```

Return the basic name.

```
HostSet GetHosts() const
```
Return the host names.

```
HostHandle GetHostHandle(const string& host) const
```
Return the host handle.

```
long GetOffset() const
long GetOffset(HostHandle host) const
long GetOffset(const string& host) const
```
Return the current offset for the specified host file.

```
bool AtEnd() const
bool AtEnd(HostHandle host) const
bool AtEnd(const string& host) const
```
Return whether an end-of-file has occurred for the specified host file.

```
void Seek(long position = kBegin)
void Seek(HostHandle host, long position = kBegin)
void Seek(const string& host, long position = kBegin)
```
Position the specified host file to the given location.

```
void Write(const TOutRecord& record)
void Write(HostHandle host, const TOutRecord& record)
void Write(const string& host, const TOutRecord& record)
```
Write the given record on the specified host.

```
bool Read(TOutRecord& record)
bool Read(const string& host, TOutRecord& record)
bool Read(HostHandle host, TOutRecord& record)
```
Read the given record on the specified host. Return whether a record was available for reading.

```
void WriteHeader(const TOutRecord& record)
void WriteHeader(const string& host, const TOutRecord& record)
void WriteHeader(HostHandle host, const TOutRecord& record)
```
Write the given record header on the specified host.

```
bool ReadHeader(TOutRecord& record)
bool ReadHeader(const string& host, TOutRecord& record)
bool ReadHeader(HostHandle host, TOutRecord& record)
```
Read the given record header on the specified host. Return whether a record was available for reading.

```
void Flush()
```
Flush any pending operations.

26. TOutRecord

This class is used for storing the values of a collection of fields. Each record has a field map holding the current values of the fields.

```
enum Type {kNoType, kChar, kUnsignedChar, kShort, kUnsignedShort,
           kInt, kUnsignedInt, kLong, kUnsignedLong, kFloat,
           kDouble, kString}
```

Field types.

```
TOutRecord()
```

Construct a record.

```
TOutRecord(const TOutRecord& record)
```

Make a copy of the given record.

```
TOutRecord& operator=(const TOutRecord& record)
```

Make the record a copy of the given record.

```
void SetField(const string& field, char value)
void SetField(const string& field, unsigned char value)
void SetField(const string& field, short value)
void SetField(const string& field, unsigned short value)
void SetField(const string& field, int value)
void SetField(const string& field, unsigned int value)
void SetField(const string& field, long value)
void SetField(const string& field, unsigned long value)
void SetField(const string& field, float value)
void SetField(const string& field, double value)
void SetField(const string& field, const string& value)
void SetField(const string& field)
```

Set the value of the specified field.

```
void GetField(const string& field, char& value) const
void GetField(const string& field, unsigned char& value) const
void GetField(const string& field, short& value) const
void GetField(const string& field, unsigned short& value) const
void GetField(const string& field, int& value) const
void GetField(const string& field, unsigned int& value) const
void GetField(const string& field, long& value) const
void GetField(const string& field, unsigned long& value) const
void GetField(const string& field, float& value) const
void GetField(const string& field, double& value) const
void GetField(const string& field, string& value) const
```

Get the value of the specified field.

```
Type GetType(const string& field) const
```

Return the type of the specified field.

```
FieldMap& GetMap()
const FieldMap& GetMap() const
```

Return the map for the record.

```
FieldMapIterator GetIterator() const
```

Return an iterator for the record's map.

27. TOutException

An output exception signals the failure of a simulation output subsystem function. Each exception has a message. Figure 9 shows the hierarchy of exception classes.

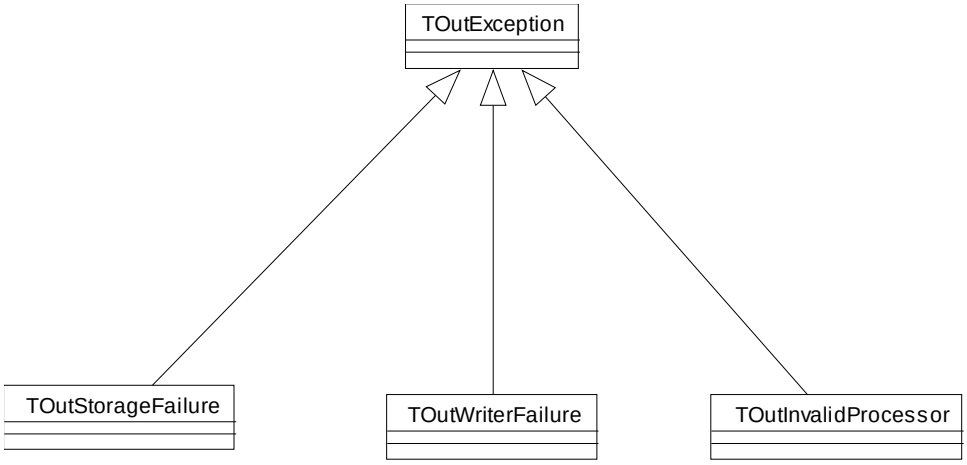

Figure 9. Exception hierarchy for the TRANSIMS simulation output subsystem (unified notation).

`TOutException(const string& message = "Simulation output error.")`
Construct an exception with the specified message text.

`TOutException(const TOutException& exception)`
Construct a copy of the specified exception.

`TOutException& operator=(const TOutException& exception)`
Make the exception a copy of the specified exception.

`const string& GetMessage() const`
Return the message text for the exception.

`class TOutStorageFailure`
This exception is thrown when a storage operation fails.

`class TOutWriterFailure`
This exception is thrown when a writer operation fails.

`class TOutInvalidProcessor`
This exception is thrown when the processor type is invalid.

III. Implementation

A. C++ Libraries

The Booch Components [RW 94] provide C++ container classes that the simulation output subsystem uses extensively. The DBtools.h++ [SL 95; Su 95] and Tools.h++ [Ke

94] libraries provide platform-independent data type and file system support, respectively. The subsystem also uses the standard C++ library [Pl 95], the standard C library [Pl 92], and the POSIX library [Ga 95]. All of these libraries compile on a wide variety of platforms (UNIX and otherwise).

B. File System

Although the simulation output subsystem runs on multiple computational nodes (CPNs), it stores output data locally (Figure 10) and thus does not require any communication between the CPNs. A unified view of the data is provided during data retrieval by accessing and collating the data on multiple remote disks (Figure 11).

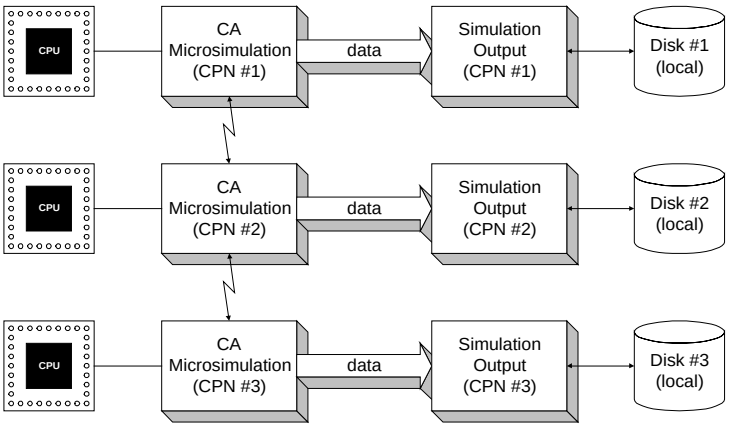

Figure 10. Simulation output subsystem configuration for simultaneous data collection on multiple CPNs.

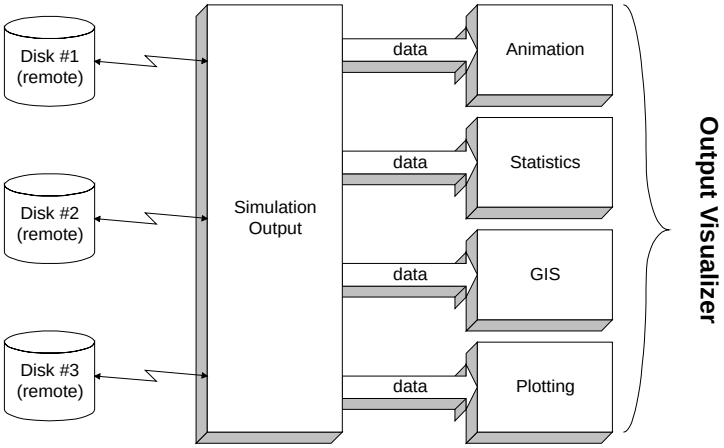

Figure 11. Simulation output subsystem configuration for data retrieval from multiple disks.

The local-storage/remote-retrieval paradigm requires coordination between the microsimulation software and the postprocessing software. The simplest way to accomplish this is to have each CPN store data locally into a directory named “local” on

each CPN—this will, of course, be a different physical disk for each CPN. Each of these local directories is given a different NFS (network file system) name (typically the name of the CPN) so that it can be accessed remotely. Figure 12 illustrates this scheme. Several other workable arrangements are possible, however.

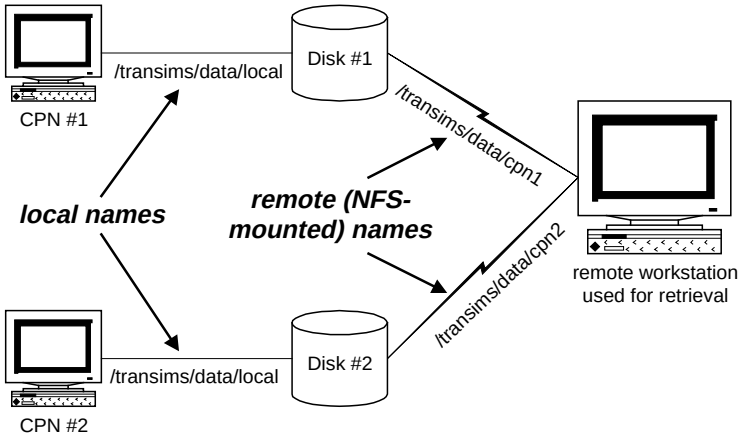

Figure 12. Typical naming and NFS-mounting scheme for simulation output directories.

C. Integration into the Microsimulation

The simulation output subsystem must be integrated into the microsimulation by subclassing the appropriate processor (`TOut…Processor`) and observer (`TOut…Observer`) classes and by providing a subclass of `TOutFactory` for creating instances of these subclasses. This mechanism allows a flexible, yet tight, coupling between the two subsystems without requiring the simulation output subsystem to be tailored to the specifics of the microsimulation. Figure 13 illustrates an example of how this subclassing can be implemented.

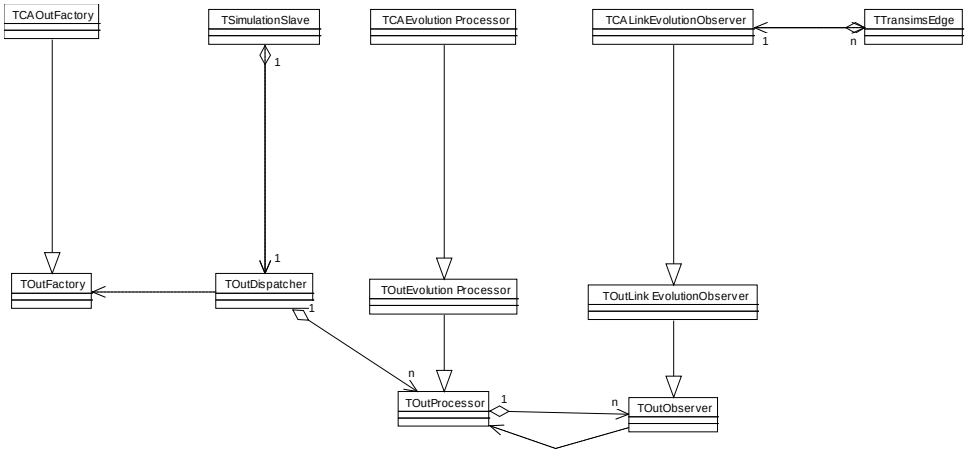

Figure 13. Example of subclassing simulation output classes in a simulation (unified notation).

IV. Usage

A. Specification Formats

This section describes the files that a user must currently prepare in order to use the simulation output subsystem in conjunction with the CA microsimulation.

The general specification output tables describe the general characteristics of the output that is to be collected during a CA run. These specifications are stored in the Oracle database management system prior to running the CA, retrieved by the CA during execution, and used to guide the simulation output subsystem's collection and storage of data.

Tables may be created in Oracle by using the database functions of the Input Editor. Tables may also be created by using the database subsystem import utility, for which the usage is:

```
Import file
```

where `file` contains information and commands for the construction of one or more data tables. Import prints a brief help message if invoked without a file. When constructing new tables, it is recommended that, when possible, an existing table description file that is similar to the desired new table description be copied and edited. This reduces the chances for errors in the SQL statements in the file.

Three tables are currently used to provide the general specification for the simulation output subsystem. Preparation of the files for each of these tables is described below.

1. Output Specification

The output specification table provides information about the time frame for collecting data and where the data should be stored. Table 1 defines the format for this table. The combination of `ROOT` and `NAME` must be unique among multiple simultaneous users of the subsystem, or else the output files will be overwritten. For example, evolutionary output might be collected on each CPN for vehicle, intersection, and signal data; the files created would be:

`ROOT + "/local" + NAME + ".veh" + ".stg"`	for vehicle data
`ROOT + "/local" + NAME + ".int" + ".stg"`	for intersection data
`ROOT + "/local" + NAME + ".sig" + ".stg"`	for signal data

Typically, the directory `ROOT + "/local" + NAME` on each host is just a NFS (network file system) link to the physical directory `ROOT + hostname + NAME`; this allows both local and global views of the output directories (see Figure 12). Note that the output files must be deleted manually when they are no longer needed.

Table 1. Format for the output specification data table (an "Output Specification" data source in the database subsystem).

Field	*Interpretation*
ROOT	The directory where the output should be written.
NAME	The output file name.
PROCESSOR	The type of processor: • "Evolution": an evolution processor. • "Event": an event processor. • "Summary": a summary processor.
TIMEMIN	The first time (in seconds from simulation start) at which to collect data.
TIMEMAX	The last time (in seconds from simulation start) at which to collect data.
TIMESTP	The frequency (in seconds) at which to report data (i.e., write it to disk).
TIMESMP	The frequency (in seconds) at which to accumulate sample data.
BOXLEN	The length of the boxes used for summary data.
ABSCISSAMN	The minimum abscissa for which to collect data (currently ignored).
ABSCISSAMX	The maximum abscissa for which to collect data (currently ignored).
ORDINATEMN	The minimum ordinate for which to collect data (currently ignored).
ORDINATEMX	The maximum ordinate for which to collect data (currently ignored).

An example output specification file is reproduced here:

```
Benchmark 1 (1 sq. mi.) Output Spec. II                          1
This is a sample output specification table.                     2
OUTSPECBENCH11II                                                 3
Output Specification                                             4
CREATE TABLE OUTSPECBENCH11II (                                  5
  PROCESSOR VARCHAR(25),                                         6
  ROOT VARCHAR(50),                                              7
  NAME VARCHAR(50),                                              8
  TIMEMIN NUMBER(10),                                            9
  TIMEMAX NUMBER(10),                                           10
  TIMESTP NUMBER(10),                                           11
  TIMESMP NUMBER(10),                                           12
  BOXLEN NUMBER(10),                                            13
  ABSCISSAMIN FLOAT,                                            14
  ABSCISSAMAX FLOAT,                                            15
  ORDINATEMIN FLOAT,                                            16
  ORDINATEMAX FLOAT,                                            17
  PRIMARY KEY (NAME)                                            18
);                                                              19
PROCESSOR, ROOT, NAME, TIMEMIN, TIMEMAX, TIMESTP, TIMESMP, BOXLEN,
ABSCISSAMIN, ABSCISSAMAX, ORDINATEMIN, ORDINATEMAX              20
'Evolution', '/transims/output4', 'bn1_evolution_1', 0, 36000,
300, 0, 0, 0, 3000, 0, 3000                                     21
'Summary', '/transims/output4', 'bn1_summary_1', 0, 36000, 180,
10, 150, 0, 3000, 0, 3000                                       22
'Summary', '/transims/output4', 'bn1_summary_2', 0, 36000, 300,
60, 50, 0, 3000, 0, 3000                                        23
'Event', '/transims/output4', 'bn1_event_1', 0, 36000, 1, 1, 1, 0,
3000, 0, 3000                                                   24
```

The first line of the file is the unique table name, with no more than 50 characters. The second line is a comment describing the table. Line 3 is a unique SQL table name. Line 4 is the data source name. Line 5 begins the SQL command for creating the table with the name specified in line 3. This command continues for fourteen additional lines and is

terminated with a closing parenthesis and a semicolon. Line 20 names the columns whose values are specified in the same order and delimited by commas on lines 21 through 24. One type of evolution data, two types of summary data, and one type of event data are specified for collection.

2. Output Node Specification

The output node specification table is used to specify the nodes at which data should be collected. Table 2 defines the format for this table. Until geographic filtering is supported this is the only way to indicate the nodes for which data collection occurs.

Table 2. Format for the output node specification data table (an "Output Node Specification" data source in the database subsystem).

Field	*Interpretation*
NAME	The output file name.
NODE	The node id.

An example output node specification file is reproduced here:

```
Benchmark 1 (1 sq. mi.) Output Node Spec. II                    1
This is a sample output node specification table.               2
OUTNODEBENCH11II                                                3
Output Node Specification                                       4
CREATE TABLE OUTNODEBENCH11II (                                 5
  NAME VARCHAR(50),                                             6
  NODE NUMBER(10),                                              7
  PRIMARY KEY (NAME,NODE)                                       8
);                                                              9
NAME, NODE                                                     10
'bn1_evolution_1',40006                                        11
'bn1_evolution_1',1793                                         12
'bn1_evolution_1',1808                                         13
'bn1_evolution_1',2413                                         14
'bn1_evolution_1',2423                                         15
'bn1_summary_2',2423                                           16
'bn1_summary_2',2424                                           17
```

The first line of the file is the unique table name, with no more than 50 characters. The second line is a comment describing the table. Line 3 is a unique SQL table name. Line 4 is the data source name. Line 5 begins the SQL command for creating the table with the name specified in line 3. This command continues for four additional lines. Line 10 names the columns whose values are specified on lines 11 and following. Evolution data is collected on five nodes and summary data on two nodes.

3. Output Link Specification

The output link specification table is used to specify the links at which data should be collected. Table 3 defines the format for this table. Until geographic filtering is supported this is the only way to indicate the links for which data collection occurs.

Table 3. Format for the output link specification data table (an "Output Link Specification" data source in the database subsystem).

Field	*Interpretation*
NAME	The output file name.
LINK	The link id.

An example output link specification file is reproduced here:

```
Benchmark 1 (1 sq. mi.) Output Link Spec. II                    1
This is a sample output link specification table.               2
OUTLINKBENCH11II                                                3
Output Link Specification                                       4
CREATE TABLE OUTLINKBENCH11II (                                 5
  NAME VARCHAR(50),                                             6
  LINK NUMBER(10),                                              7
  PRIMARY KEY (NAME,LINK)                                       8
);                                                              9
NAME, LINK                                                     10
'bn1_summary_1',11150000                                       11
'bn1_summary_1',11140001                                       12
'bn1_summary_1',8830507                                        13
'bn1_event_1',8830507                                          14
'bn1_event_1',8810509                                          15
'bn1_event_1',8811002                                          16
```

The first line of the file is the unique table name, with no more than 50 characters. The second line is a comment describing the table. Line 3 is a unique SQL table name. Line 4 is the data source name. Line 5 begins the SQL command for creating the table with the name specified in line 3. This command continues for four additional lines. Line 10 names the variables whose values are specified on lines 11 and following. Summary data is collected on three links and event data on three links.

B. Data Retrieval

The binary output stored in a distributed manner by the microsimulation is generally postprocessed to collect it into a single location. The DumpStorage utility may be used to postprocess the output into a text format for display or analysis. The usage is:

```
DumpStorage outname root name hosts…
```

where `outname` is the destination file name, `root` is the root directory for the host subdirectories, `name` is the name of the storage file, and `hosts…` are the host subdirectory names. For example,

```
DumpStorage bn1_summary_1.txt /transims/output4 bn1_summary_1.tim
bach faure sousa
```

collates the data from the files

```
/transims/output4/bach/bn1_summary_1.tim.stg
/transims/output4/faure/bn1_summary_1.tim.stg
/transims/output4/sousa/bn1_summary_1.tim.stg
```

into the file

```
./bn1_summary_1.txt
```

in a tab-delimited ASCII text format.

C. Output Formats

1. Evolution Data

Vehicle evolution data files have the storage file suffix `.veh.stg`. Table 4 lists the fields present in such files; each record in the file represent a single vehicle. The data reporting start time, finish time, and reporting frequency are given by the output specification. The output specification also determines on which links the data is collected.

Table 4. Data format for vehicle evolution storage (`.veh.stg`) files.

Field	*Interpretation*
VEHICLE	The vehicle id.
NODE	The node the vehicle was traveling away from.
LINK	The link the vehicle was traveling on.
LANE	The lane the vehicle was traveling on.
TIME	The time the data was taken (in seconds from simulation start).
DISTANCE	The distance (in meters) the vehicle was away from the node setback.
VELOCITY	The velocity (in meters per second) the vehicle was traveling.
STATUS	The vehicle status bits. • 1: The vehicle is lost. • 2: The vehicle is at a dead end. • 4: The vehicle has just entered the study area. • 8: The vehicle has just exited the study area. • 16: The vehicle is in the study area. • 32: The vehicle has an invalid plan.

Intersection evolution data files have the storage file suffix `.int.stg`. Table 5 lists the fields present in such files; each record in the file represents a single vehicle. The data reporting start time, finish time, and reporting frequency are given by the output specification. The output specification also determines on which nodes the data is collected.

Table 5. Data format for intersection evolution storage (`.int.stg`) files.

Field	*Interpretation*
VEHICLE	The vehicle id.
NODE	The node where the vehicle is located.
TIME	The time the data was taken (in seconds from simulation start).
LINK	The link the vehicle entered from.
LANE	The lane the vehicle entered from.
QINDEX	The vehicle position in the queue.

Signal evolution data files have the storage files suffix `.sig.stg`. Table 6 lists the fields present in such files; each record in the file represents an incoming lane at an intersection. The data reporting start time, finish time, and reporting frequency are given by the output

specification. The output specification also determines on which nodes the data is collected.

Table 6. Data format for signal evolution storage (`.sig.stg`) files.

Field	*Interpretation*
NODE	The node where the signal is located.
TIME	The time the data was taken (in seconds from simulation start).
LINK	The link entering the signal.
LANE	The lane entering the signal.
SIGNAL	The type of control present: • 0: None. • 1: Stop. • 2: Yield. • 3: Wait. • 4: Caution. • 5: Permitted. • 6: Protected.

2. Event Data

Vehicle event data files have the storage file suffix `.veh.stg`. Table 7 lists the fields present in such files; each record in the file represents a single vehicle event. The data reporting start time, finish time, and reporting frequency are given by the output specification. This data is collected for all links—i.e., the output link specification is ignored.

Table 7. Data format for vehicle event storage (`.veh.stg`) files.

Field	*Interpretation*
VEHICLE	The vehicle id.
NODE	The node the vehicle was traveling away from.
LINK	The link the vehicle was traveling on.
LANE	The lane the vehicle was traveling on.
TIME	The time the data was taken (in seconds from simulation start).
DISTANCE	The distance (in meters) the vehicle was away from the node setback.
VELOCITY	The velocity (in meters per second) the vehicle was traveling.
STATUS	The vehicle status bits. • 1: The vehicle is lost. • 2: The vehicle is at a dead end. • 4: The vehicle has just entered the study area. • 8: The vehicle has just exited the study area. • 16: The vehicle is in the study area. • 32: The vehicle has an invalid plan.

3. Summary Data

Link space summary data files have the storage file suffix `.spa.stg`. Table 8 lists the fields present in such files; each record in the file represents the summary for a single box on a link. If there is a short box, it is at the beginning of the link. The beginning distance of the box, which is not written in the file, is the ending distance of the box minus the box

size. The data reporting start time, finish time, the sampling frequency, the data reporting frequency, and the box size are given by the output specification. The output specification also determines on which links the data is collected. Note that no data is reported at the reporting start time. Also, there may be two entries for links that are split by a CPN boundary; when this happens, it is necessary to add the respective `COUNT` and `SUM` entries for the duplicate box records.

Table 8. Data format for link space summary storage (`.spa.stg`) files.

Field	*Interpretation*
LINK	The link being reported.
NODE	The node from which vehicles were traveling.
DISTANCE	The ending distance of the box (in meters) from the node setback.
TIME	The time the data was taken (in seconds from simulation start).
COUNT	The number of vehicles in the box.
SUM	The sum of the vehicle velocities (in meters per second) in the box.

Link time summary data files have the storage file suffix `.tim.stg`. Table 9 lists the fields present in such files; each record in the file represents the summary for a single direction of a link. The data reporting start time, finish time, and reporting frequency are given by the output specification. The output specification also determines on which links the data is collected. Note that no data is reported at the reporting start time. Also, there may be two entries for links that are split by a CPN boundary; when this happens, it is necessary to add the respective `COUNT`, `SUM`, and `SUMSQUARES` entries for the duplicate link records.

Table 9. Data format for link time summary storage (`.tim.stg`) files.

Field	*Interpretation*
LINK	The link being reported.
NODE	The node from which vehicles were traveling.
TIME	The time the data was taken (in seconds from simulation start).
COUNT	The number of vehicles leaving the link.
SUM	The sum of the vehicle travel times (in seconds) for vehicles leaving the link. The time spent in the previous intersection is included in this value.
SUMSQUARES	The sum of the vehicle travel time squares (in seconds squared) for vehicles leaving the link. The time spent in the previous intersection is included in this value.

D. Example of Retrieval Using C++

The following example shows how to access and retrieve evolution data using C++ calls to the simulation output subsystem.

```
//  Create a character buffer for lines from standard input.
char line[1000];

//  Get the file name for the vehicle text output file.
cin.getline(line, 1000);
TOutWriter* vehicleWriter = new TOutTextWriter(line, "\t", TRUE);
```

```
//  Get the file name for the output text files.
cin.getline(line, 1000);
TOutWriter* intersectionWriter = new TOutTextWriter(line, "\t",
        TRUE);
cin.getline(line, 1000);
TOutWriter* signalWriter = new TOutTextWriter(line, "\t", TRUE);
TDbDirectory directory(TDbDirectoryDescription("IOC-1"));

//  Open the data sources.
TDbSource generalSource(directory,
        directory.GetSource("Output Specification"));
TDbSource nodeSource(directory,
            directory.GetSource("Output Node Specification"));
TDbSource linkSource(directory,
            directory.GetSource("Output Link Specification"));

//  Get the data table names and open the data tables.
cin.getline(line, 1000);
TDbTable generalTable(generalSource,
        generalSource.GetTable(line));
cin.getline(line, 1000);
TDbTable nodeTable(nodeSource, nodeSource.GetTable(line));
cin.getline(line, 1000);
TDbTable linkTable(linkSource, linkSource.GetTable(line));

//  Create the specification.
TOutGeneralSpecificationReader
        reader(TOutSpecificationReader(generalTable,
        nodeTable, linkTable));
reader.Reset();

//  Get the set of hosts.
TOutStorage::HostSet hosts(HashValue);
for (cin.getline(line, 1000); !cin.eof(); cin.getline(line, 1000))
    hosts.Add(line);

//  Create the retriever.
TOutEvolutionRetriever retriever(hosts,
        TOutGeneralSpecification(reader));

//  Retrieve the data.
retriever.Retrieve(*vehicleWriter, *intersectionWriter,
        *signalWriter, hosts.Extent() > 1);

//  Destroy the writers.
delete vehicleWriter;
delete intersectionWriter;
delete signalWriter;
```

E. Notes

1. Database Setup

The CreateSources utility must be executed before the first use of the simulation output subsystem. This application registers the data sources in the database subsystem.

2. Empty Storage Files

The present implementation of the `TOut…Retriever` classes cannot handle input files of zero length. If no data is collected on a CPN, it is advisable to delete the empty storage file(s) created for that CPN.

V. Future Work

Future work planned for the TRANSIMS simulation output subsystem will focus on several areas:

- collecting new types of data such as turn counts, fundamental diagrams, and velocity-acceleration histograms,
- enhancing the performance of the subsystem by supporting data compression, indexing, sorting, and filtering,
- improving the DumpStorage utility to provide more flexibility in exporting data, and
- eliminating the dependence on commercial products such as DBtools.h++ and Tools.h++.

VI. References

[Ga 95] B. O. Gallmeister, *POSIX.4: Programming for the Real World*, (Sebastopol, California: O'Reilly & Associates, 1995).

[Ke 94] T. Keffer, *Tools.h++ Introduction and Reference Manual*, Version 6, (Corvallis, Oregon: Rogue Wave Software, 1994).

[Pl 92] P. J. Plauger, *The Standard C Library*, (Englewood Cliffs, New Jersey: Prentice Hall, 1992).

[Pl 95] P. J. Plauger, *The Draft Standard C++ Library*, (Englewood Cliffs, New Jersey: Prentice Hall, 1995).

[RW 94] Rogue Wave Software, *The C++ Booch Components*, Version 2.3, (Corvallis, Oregon: Rogue Wave Software, 1994).

[SL 95] S. Sulsky and K. L. Lohn, *DBtools.h++ User's Guide and Tutorial*, Version 1, (Corvallis, Oregon: Rogue Wave Software, 1995).

[Su 95] S. Sulsky, *DBtools.h++ Class Reference*, Version 1, (Corvallis, Oregon: Rogue Wave Software, 1995).

VII. APPENDIX: Booch Notation Diagrams

Figure 14, Figure 15, Figure 16, and Figure 17 repeat the class diagrams of Figure 5, Figure 6, Figure 7, and Figure 8; Figure 18 repeats the exception hierarchy of Figure 9; and Figure 19 repeats the example of Figure 13 using Booch notation.

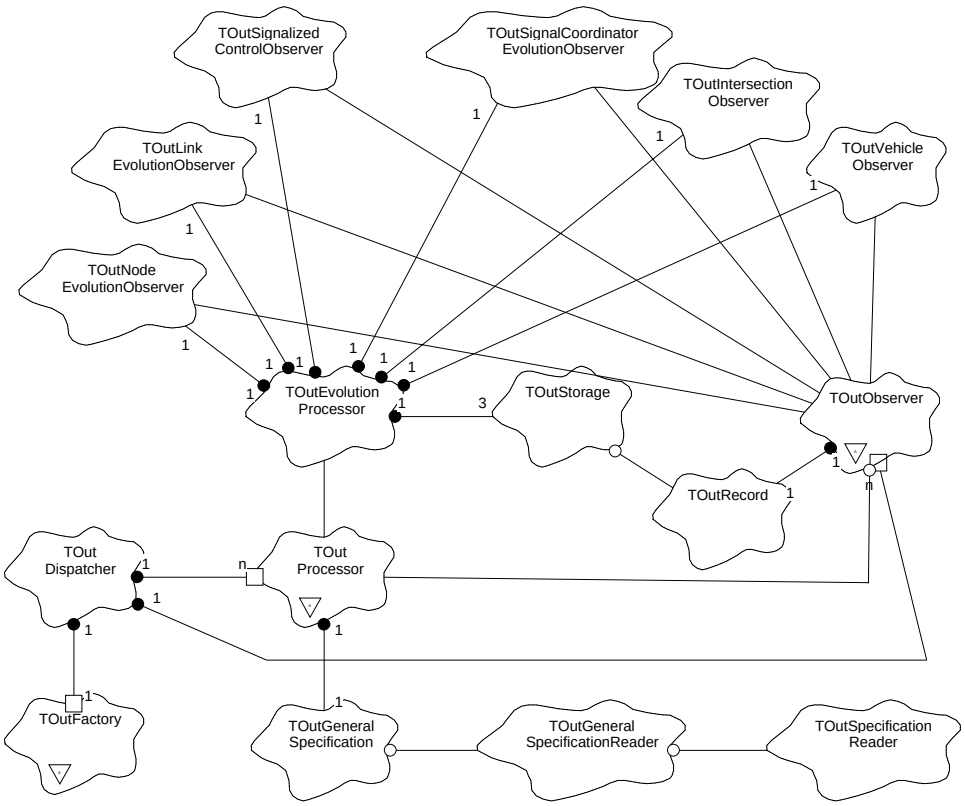

Figure 14. Class diagram for the TRANSIMS simulation output subsystem classes involved in evolution data collection (Booch notation).

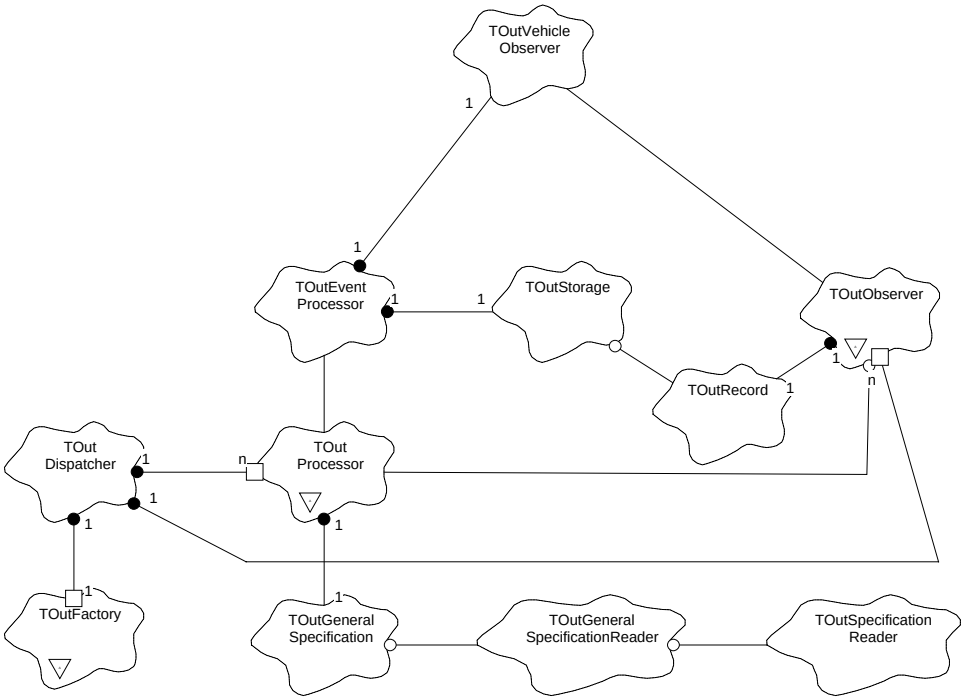

Figure 15. Class diagram for the TRANSIMS simulation output subsystem classes involved in event data collection (Booch notation).

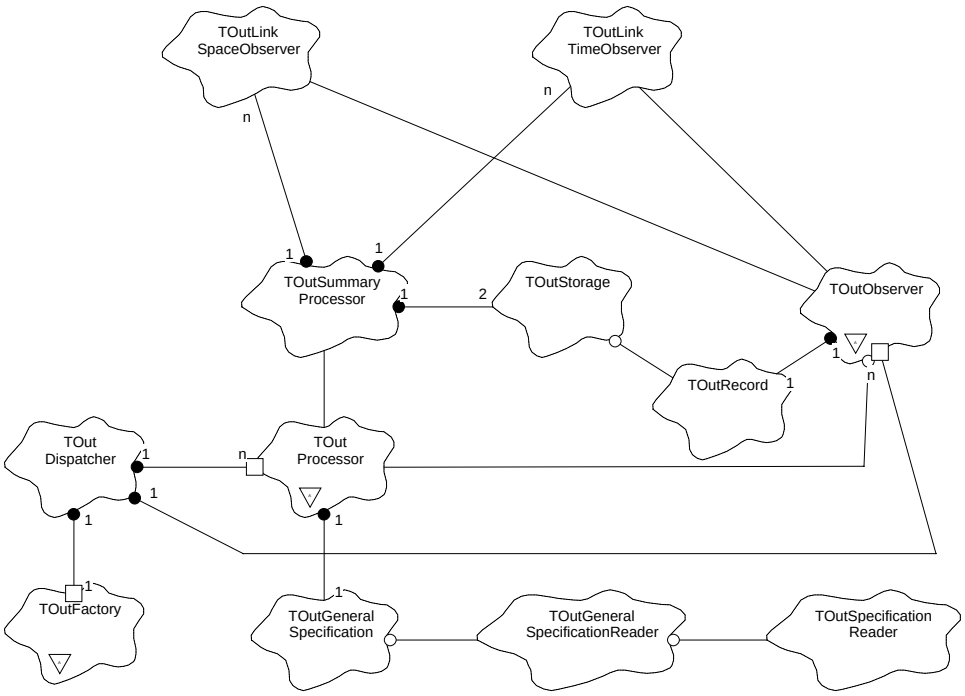

Figure 16. Class diagram for the TRANSIMS simulation output subsystem classes involved in summary data collection (Booch notation).

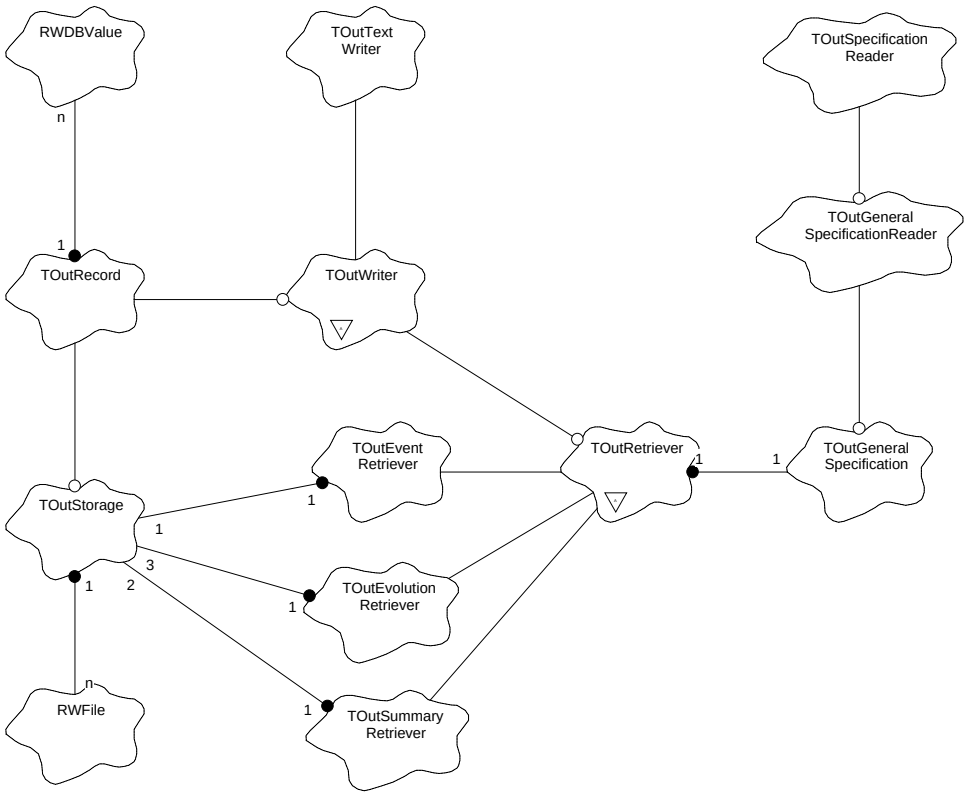

Figure 17. Class diagram for the TRANSIMS simulation output subsystem classes involved in data retrieval (Booch notation).

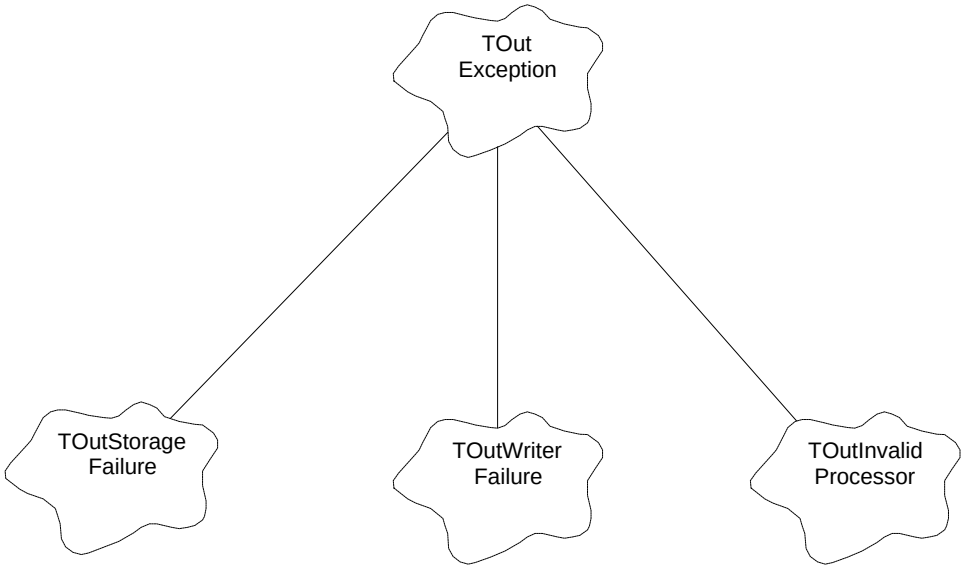

Figure 18. Exception hierarchy for the TRANSIMS simulation output subsystem (Booch notation).

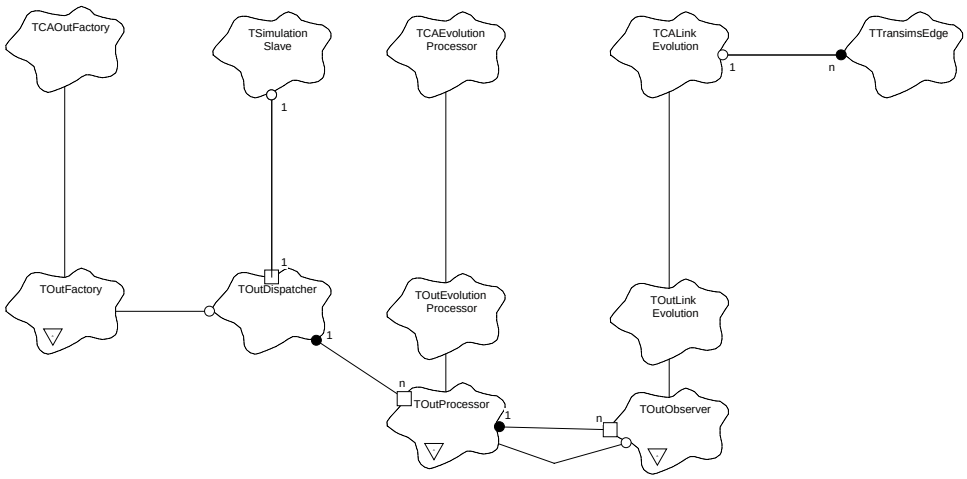

Figure 19. Example of subclassing simulation output classes in a simulation (Booch notation).

VIII. APPENDIX: Source Code

This appendix contains the complete C++ source code for the simulation output subsystem classes.

A. TOutDispatcher Class

1. Dispatcher.h

```
//  Project: TRANSIMS
//  Subsystem: Simulation Output
//  $RCSfile: Dispatcher.h,v $
//  $Revision: 0.5 $
//  $Date: 1996/06/19 17:07:17 $
//  $State: Stab $
//  $Author: bwb $
//  U.S. Government Copyright 1995
//  All rights reserved


#ifndef TRANSIMS_OUT_DISPATCHER
#define TRANSIMS_OUT_DISPATCHER


//  Include TRANSIMS header files.
#include "GBL/Globals.h"
#include "OUT/Id.h"
#include "OUT/Observer.h"


//  Include Booch Components header files.
#include "BCStoreM.h"
#include "BCMapU.h"
#include "BCSetU.h"


//  Forward declarations.
class TOutProcessor;
class TOutFactory;
class TOutSpecificationReader;
class TOutEvolutionProcessor;
class TOutEventProcessor;
class TOutSummaryProcessor;


//  An output dispatcher coordinates the construction of simulation output
```

```
//  objects and supervises the transfer of data.
class TOutDispatcher
{
  public:

    //  Type definitions.
    typedef BC_TUnboundedMap<OutProcessorId, TOutProcessor*, 10U, BC_CManaged>
            ProcessorMap;
    typedef BC_TMapActiveIterator<OutProcessorId, TOutProcessor*>
            ProcessorMapIterator;
    typedef BC_TUnboundedMap<OutObserverId, TOutObserver*, 10U, BC_CManaged>
            ObserverMap;
    typedef BC_TMapActiveIterator<OutObserverId, TOutObserver*>
            ObserverMapIterator;
    typedef BC_TUnboundedSet<TOutEvolutionProcessor*, 10U, BC_CManaged>
            EvolutionProcessorSet;
    typedef BC_TSetActiveIterator<TOutEvolutionProcessor*>
            EvolutionProcessorSetIterator;
    typedef BC_TUnboundedSet<TOutEventProcessor*, 10U, BC_CManaged>
            EventProcessorSet;
    typedef BC_TSetActiveIterator<TOutEventProcessor*>
            EventProcessorSetIterator;
    typedef BC_TUnboundedSet<TOutSummaryProcessor*, 10U, BC_CManaged>
            SummaryProcessorSet;
    typedef BC_TSetActiveIterator<TOutSummaryProcessor*>
            SummaryProcessorSetIterator;

    //  Observer types.
    enum EObserverType {kNodeEvolutionObserver, kLinkEvolutionObserver,
            kVehicleObserver, kIntersectionObserver,
            kSignalCoordinatorEvolutionObserver, kSignalizedControlObserver,
            kLinkSpaceObserver, kLinkTimeObserver};

    //  Construct an output dispatcher.
    TOutDispatcher(TOutSpecificationReader& reader, TOutFactory& factory);

    //  Destruct an output dispatcher.
    ~TOutDispatcher();

    //  Return whether two output dispatchers are the same.
    bool operator==(const TOutDispatcher& dispatcher) const;
    bool operator!=(const TOutDispatcher& dispatcher) const;

    //  Begin output recording for this time step.
    void RecordOutput(REAL time);

    //  Return the processors.
    ProcessorMap& GetProcessors();
    const ProcessorMap& GetProcessors() const;

    //  Return the observers.
    ObserverMap& GetObservers();
    const ObserverMap& GetObservers() const;

    //  Return the evolution processors.
    EvolutionProcessorSet& GetEvolutionProcessors();
    const EvolutionProcessorSet& GetEvolutionProcessors() const;

    //  Return the event processors.
    EventProcessorSet& GetEventProcessors();
    const EventProcessorSet& GetEventProcessors() const;

    //  Return the summary processors.
    SummaryProcessorSet& GetSummaryProcessors();
    const SummaryProcessorSet& GetSummaryProcessors() const;

    //  Define the node observers.
    void SetNodeObservers(TOutObserver::ObserverMap& observers);

    //  Define the link observers.
    void SetLinkObservers(TOutObserver::ObserverMap& observers);

    //  Define the vehicle observers.
    void SetVehicleObservers(TOutObserver::ObserverMap& observers);

    //  Define the intersection observers.
```

```
    void SetIntersectionObservers(TOutObserver::ObserverMap& observers);

    //  Define the signal coordinator observers.
    void SetSignalCoordinatorObservers(TOutObserver::ObserverMap& observers);

    //  Define the signalized control observers.
    void SetSignalizedControlObservers(TOutObserver::ObserverMap& observers);

    //  Define the link space observers.
    void SetLinkSpaceObservers(TOutObserver::ObserverMap& observers);

    //  Define the link time observers.
    void SetLinkTimeObservers(TOutObserver::ObserverMap& observers);

    //  Undefine the link space observers.
    void ClearLinkSpaceObservers(TOutObserver::ObserverMap& observers);

    //  Undefine the link time observers.
    void ClearLinkTimeObservers(TOutObserver::ObserverMap& observers);

  private:

    //  Do not allow dispatchers to be copied.
    TOutDispatcher(const TOutDispatcher&) {}

    //  Do not allow dispatchers to be assigned.
    TOutDispatcher& operator=(const TOutDispatcher&) {return *this;}

    //  Each dispatcher has a processor map.
    ProcessorMap fProcessors;

    //  Each dispatcher has an observer map.
    ObserverMap fObservers;

    //  Each dispatcher has evolution processors.
    EvolutionProcessorSet fEvolutionProcessors;

    //  Each dispatcher has event processors.
    EventProcessorSet fEventProcessors;

    //  Each dispatcher has summary processors.
    SummaryProcessorSet fSummaryProcessors;

    //  Each dispatcher remembers the factor it uses.
    TOutFactory* fFactory;
};


#endif // TRANSIMS_OUT_DISPATCHER
```

2. Dispatcher.C

```
//  Project: TRANSIMS
//  Subsystem: Simulation Output
//  $RCSfile: Dispatcher.C,v $
//  $Revision: 0.5 $
//  $Date: 1996/06/19 17:06:25 $
//  $State: Stab $
//  $Author: bwb $
//  U.S. Government Copyright 1995
//  All rights reserved


//  Include TRANSIMS header files.
#include "OUT/Dispatcher.h"
#include "OUT/Processor.h"
#include "OUT/Factory.h"
#include "OUT/Exception.h"
#include "OUT/EvolutionProcessor.h"
#include "OUT/EventProcessor.h"
#include "OUT/SummaryProcessor.h"
#include "OUT/LinkEvolutionObserver.h"
#include "OUT/VehicleObserver.h"
#include "OUT/NodeEvolutionObserver.h"
```

```
#include "OUT/IntersectionObserver.h"
#include "OUT/SignalCoordinatorEvolutionObserver.h"
#include "OUT/SignalizedControlObserver.h"
#include "OUT/LinkSpaceObserver.h"
#include "OUT/LinkTimeObserver.h"
#include "CA/CAOutFactory.h"


//  Define the hash function for processor maps.
static BC_Index ProcessorIdHashValue(const OutProcessorId& id)
{
    return BC_Index(id);
}


//  Define the hash function for observer maps.
static BC_Index ObserverIdHashValue(const OutObserverId& id)
{
    return BC_Index(id);
}


//  Define the hash function for evolution processor sets.
static BC_Index EvolutionProcessorHashValue(TOutEvolutionProcessor* const& proc)
{
    return BC_Index(proc->GetId());
}


//  Define the hash function for event processor sets.
static BC_Index EventProcessorHashValue(TOutEventProcessor* const& proc)
{
    return BC_Index(proc->GetId());
}


//  Define the hash function for summary processor sets.
static BC_Index SummaryProcessorHashValue(TOutSummaryProcessor* const& proc)
{
    return BC_Index(proc->GetId());
}


//  Construct an output dispatcher.
TOutDispatcher::TOutDispatcher(TOutSpecificationReader& reader, TOutFactory&
        factory)
  : fProcessors(ProcessorIdHashValue),
    fObservers(ObserverIdHashValue),
    fEvolutionProcessors(EvolutionProcessorHashValue),
    fEventProcessors(EventProcessorHashValue),
    fSummaryProcessors(SummaryProcessorHashValue),
    fFactory(&factory)
{
    TOutGeneralSpecificationReader genReader(reader);
    for (genReader.Reset(); genReader.MoreSpecifications();
            genReader.GetNextSpecification()) {

        if (genReader.GetProcessorType() ==
                TOutGeneralSpecificationReader::kEvolutionProcessor) {

            TOutEvolutionProcessor* proc =
                    factory.NewEvolutionProcessor(TOutProcessor::GetNextId(),
                    TOutGeneralSpecification(genReader));
            fProcessors.Bind(proc->GetId(), proc);
            fEvolutionProcessors.Add(proc);

            OutObserverId oid;
            oid = kLinkEvolutionObserver;
            if (fObservers.IsBound(oid) == 0) {
                TOutLinkEvolutionObserver* obs =
                        factory.NewLinkEvolutionObserver(oid);
                fObservers.Bind(obs->GetId(), obs);
            }
            proc->SetLinkObserver(**fObservers.ValueOf(oid));
            oid = kVehicleObserver;
            if (fObservers.IsBound(oid) == 0) {
```

```
                TOutVehicleObserver* obs = factory.NewVehicleObserver(oid);
                fObservers.Bind(obs->GetId(), obs);
            }
            proc->SetVehicleObserver(**fObservers.ValueOf(oid));
            oid = kNodeEvolutionObserver;
            if (fObservers.IsBound(oid) == 0) {
                TOutNodeEvolutionObserver* obs =
                        factory.NewNodeEvolutionObserver(oid);
                fObservers.Bind(obs->GetId(), obs);
            }
            proc->SetNodeObserver(**fObservers.ValueOf(oid));
            oid = kIntersectionObserver;
            if (fObservers.IsBound(oid) == 0) {
                TOutIntersectionObserver* obs =
                        factory.NewIntersectionObserver(oid);
                fObservers.Bind(obs->GetId(), obs);
            }
            proc->SetIntersectionObserver(**fObservers.ValueOf(oid));
            oid = kSignalCoordinatorEvolutionObserver;
            if (fObservers.IsBound(oid) == 0) {
                TOutSignalCoordinatorEvolutionObserver* obs =
                        factory.NewSignalCoordinatorEvolutionObserver(oid);
                fObservers.Bind(obs->GetId(), obs);
            }
            proc->SetSignalCoordinatorObserver(**fObservers.ValueOf(oid));
            oid = kSignalizedControlObserver;
            if (fObservers.IsBound(oid) == 0) {
                TOutSignalizedControlObserver* obs =
                        factory.NewSignalizedControlObserver(oid);
                fObservers.Bind(obs->GetId(), obs);
            }
            proc->SetSignalizedControlObserver(**fObservers.ValueOf(oid));

        } else if (genReader.GetProcessorType() ==
                TOutGeneralSpecificationReader::kEventProcessor) {

            TOutEventProcessor* proc =
                    factory.NewEventProcessor(TOutProcessor::GetNextId(),
                    TOutGeneralSpecification(genReader));
            fProcessors.Bind(proc->GetId(), proc);
            fEventProcessors.Add(proc);

            OutObserverId oid;
            oid = kVehicleObserver;
            if (fObservers.IsBound(oid) == 0) {
                TOutVehicleObserver* obs = factory.NewVehicleObserver(oid);
                fObservers.Bind(obs->GetId(), obs);
            }
            proc->SetVehicleObserver(**fObservers.ValueOf(oid));

        } else if (genReader.GetProcessorType() ==
                TOutGeneralSpecificationReader::kSummaryProcessor) {

            TOutSummaryProcessor* proc =
                    factory.NewSummaryProcessor(TOutProcessor::GetNextId(),
                    TOutGeneralSpecification(genReader));
            fProcessors.Bind(proc->GetId(), proc);
            fSummaryProcessors.Add(proc);

        } else

            throw TOutInvalidProcessor();

    }
}


//  Destruct an output dispatcher
TOutDispatcher::~TOutDispatcher()
{
    for (ObserverMapIterator i(fObservers); !i.IsDone(); i.Next())
        delete *i.CurrentValue();
    for (ProcessorMapIterator j(fProcessors); !j.IsDone(); j.Next())
        delete *j.CurrentValue();
    delete fFactory;
}
```

```
//  Return whether two output dispatchers are the same
bool TOutDispatcher::operator==(const TOutDispatcher& d) const
{
    return (this == &d);
}

bool TOutDispatcher::operator!=(const TOutDispatcher& d) const
{
    return !(this == &d);
}


//  Begin recording output for this time step.
void TOutDispatcher::RecordOutput(REAL time)
{
    for (ProcessorMapIterator it(fProcessors); !it.IsDone(); it.Next())
        (*it.CurrentValue())->RecordOutput(time);
}


//  Return the processors.
TOutDispatcher::ProcessorMap& TOutDispatcher::GetProcessors()
{
    return fProcessors;
}

const TOutDispatcher::ProcessorMap& TOutDispatcher::GetProcessors() const
{
    return fProcessors;
}


//  Return the observers.
TOutDispatcher::ObserverMap& TOutDispatcher::GetObservers()
{
    return fObservers;
}

const TOutDispatcher::ObserverMap& TOutDispatcher::GetObservers() const
{
    return fObservers;
}


//  Return the evolution processors.
TOutDispatcher::EvolutionProcessorSet& TOutDispatcher::GetEvolutionProcessors()
{
    return fEvolutionProcessors;
}

const TOutDispatcher::EvolutionProcessorSet&
        TOutDispatcher::GetEvolutionProcessors() const
{
    return fEvolutionProcessors;
}


//  Return the event processors.
TOutDispatcher::EventProcessorSet& TOutDispatcher::GetEventProcessors()
{
    return fEventProcessors;
}

const TOutDispatcher::EventProcessorSet& TOutDispatcher::GetEventProcessors()
        const
{
    return fEventProcessors;
}


//  Return the summary processors.
TOutDispatcher::SummaryProcessorSet& TOutDispatcher::GetSummaryProcessors()
{
    return fSummaryProcessors;
```

```
}

const TOutDispatcher::SummaryProcessorSet&
        TOutDispatcher::GetSummaryProcessors() const
{
    return fSummaryProcessors;
}


//  Define the node observers.
void TOutDispatcher::SetNodeObservers(TOutObserver::ObserverMap& observers)
{
    for (EvolutionProcessorSetIterator it(fEvolutionProcessors); !it.IsDone();
            it.Next())
        observers.Bind(*it.CurrentItem(),
                &(*it.CurrentItem())->GetNodeObserver());
    //  iterate thru summary processors
}


//  Define the link observers.
void TOutDispatcher::SetLinkObservers(TOutObserver::ObserverMap& observers)
{
    for (EvolutionProcessorSetIterator it(fEvolutionProcessors); !it.IsDone();
            it.Next())
        observers.Bind(*it.CurrentItem(),
                &(*it.CurrentItem())->GetLinkObserver());
    //  iterate thru summary processors
}


//  Define the vehicle observers.
void TOutDispatcher::SetVehicleObservers(TOutObserver::ObserverMap& observers)
{
    for (EvolutionProcessorSetIterator it(fEvolutionProcessors); !it.IsDone();
            it.Next())
         observers.Bind(*it.CurrentItem(),
                &(*it.CurrentItem())->GetVehicleObserver());
    for (EventProcessorSetIterator j(fEventProcessors); !j.IsDone(); j.Next())
        observers.Bind(*j.CurrentItem(),
                &(*j.CurrentItem())->GetVehicleObserver());
}


//  Define the intersection observers.
void TOutDispatcher::SetIntersectionObservers(TOutObserver::ObserverMap&
        observers)
{
    for (EvolutionProcessorSetIterator it(fEvolutionProcessors); !it.IsDone();
            it.Next())
        observers.Bind(*it.CurrentItem(),
                &(*it.CurrentItem())->GetIntersectionObserver());
}


//  Define the signal coordinator observers.
void TOutDispatcher::SetSignalCoordinatorObservers(TOutObserver::ObserverMap&
        observers)
{
    for (EvolutionProcessorSetIterator it(fEvolutionProcessors); !it.IsDone();
            it.Next())
        observers.Bind(*it.CurrentItem(),
                &(*it.CurrentItem())->GetSignalCoordinatorObserver());
}


//  Define the signalized control observers.
void TOutDispatcher::SetSignalizedControlObservers(TOutObserver::ObserverMap&
        observers)
{
    for (EvolutionProcessorSetIterator it(fEvolutionProcessors); !it.IsDone();
            it.Next())
        observers.Bind(*it.CurrentItem(),
                &(*it.CurrentItem())->GetSignalizedControlObserver());
}
```

```
//  Define the link space observers.
void TOutDispatcher::SetLinkSpaceObservers(TOutObserver::ObserverMap& observers)
{
    TOutLinkSpaceObserver* observer =
            fFactory->NewLinkSpaceObserver(TOutObserver::GetNextId());
    for (SummaryProcessorSetIterator i(fSummaryProcessors); !i.IsDone();
            i.Next()) {
        observers.Bind(*i.CurrentItem(), observer);
        (*i.CurrentItem())->AddSpaceObserver(*observer);
    }
    fObservers.Bind(observer->GetId(), observer);
}


//  Define the link time observers.
void TOutDispatcher::SetLinkTimeObservers(TOutObserver::ObserverMap& observers)
{
    TOutLinkTimeObserver* observer =
            fFactory->NewLinkTimeObserver(TOutObserver::GetNextId());
    for (SummaryProcessorSetIterator i(fSummaryProcessors); !i.IsDone();
            i.Next()) {
        observers.Bind(*i.CurrentItem(), observer);
        (*i.CurrentItem())->AddTimeObserver(*observer);
    }
    fObservers.Bind(observer->GetId(), observer);
}


//  Undefine the link space observers.
void TOutDispatcher::ClearLinkSpaceObservers(TOutObserver::ObserverMap&
        observers)
{
    bool deleted = FALSE;
    for (TOutObserver::ObserverMapIterator i(observers); !i.IsDone();
            i.Next()) {
        TOutSummaryProcessor* processor = (TOutSummaryProcessor*)
                *i.CurrentItem();
        TOutObserver* observer = *i.CurrentValue();
        processor->RemoveSpaceObserver(*observer);
        fObservers.Unbind(observer->GetId());
        if (!deleted) { //ISSUE(bwb): We may have to generalized this later.
            delete observer;
            deleted = TRUE;
        }
    }
    observers.Clear();
}


//  Undefine the link time observers.
void TOutDispatcher::ClearLinkTimeObservers(TOutObserver::ObserverMap&
        observers)
{
    bool deleted = FALSE;
    for (TOutObserver::ObserverMapIterator i(observers); !i.IsDone();
            i.Next()) {
        TOutSummaryProcessor* processor = (TOutSummaryProcessor*)
                *i.CurrentItem();
        TOutObserver* observer = *i.CurrentValue();
        processor->RemoveTimeObserver(*observer);
        fObservers.Unbind(observer->GetId());
        if (!deleted) { //ISSUE(bwb): We may have to generalized this later.
            delete observer;
            deleted = TRUE;
        }
    }
    observers.Clear();
}
```

B. TOutEventProcessor Class

1. EventProcessor.h

```
//  Project: TRANSIMS
//  Subsystem: Simulation Output
//  $RCSfile: EventProcessor.h,v $
//  $Revision: 1.2 $
//  $Date: 1996/06/19 17:13:17 $
//  $State: Stab $
//  $Author: bwb $
//  U.S. Government Copyright 1996
//  All rights reserved


#ifndef TRANSIMS_OUT_EVENTPROCESSOR
#define TRANSIMS_OUT_EVENTPROCESSOR


//  Include TRANSIMS header files.
#include "OUT/Processor.h"
#include "OUT/Storage.h"


//  Forward declarations.
class TOutObserver;


//  An output event processor deals with conditions occuring in the simulation
//  such as vehicle entry, vehicle exit, and lost vehicles.
class TOutEventProcessor
  : public TOutProcessor
{
  public:

    //  Construct an event processor.
    TOutEventProcessor(OutProcessorId id, const TOutGeneralSpecification&
            specification, UINT vehicleMask);

    //  Destruct an event processor
    virtual ~TOutEventProcessor();

    //  Return the processor type.
    EProcessorType GetProcessorType() const;

    //  Begin recording output for this time step.
    virtual void RecordOutput(REAL time);

    //  Finish recording output for a vehicle.
    virtual void RecordVehicle();

    //  Return the vehicle observer.
    TOutObserver& GetVehicleObserver();
    const TOutObserver& GetVehicleObserver() const;

    //  Define the vehicle observer.
    void SetVehicleObserver(TOutObserver& observer);

    //  Return the vehicle status mask.
    UINT GetVehicleMask() const;

  private:

    //  An event processor has a vehicle observer.
    TOutObserver* fVehicleObserver;

    //  An event processor is connected to a vehicle storage.
    TOutStorage fVehicleStorage;

    //  An event processor has a vehicle status mask.
    UINT fVehicleMask;
};
```

```
#endif // TRANSIMS_OUT_EVENTPROCESSOR
```

2. EventProcessor.C

```
//  Project: TRANSIMS
//  Subsystem: Simulation Output
//  $RCSfile: EventProcessor.C,v $
//  $Revision: 1.2 $
//  $Date: 1996/06/19 17:12:02 $
//  $State: Stab $
//  $Author: bwb $
//  U.S. Government Copyright 1996
//  All rights reserved


//  Include TRANSIMS header files.
#include "OUT/EventProcessor.h"
#include "OUT/Exception.h"
#include "OUT/Observer.h"
#include "OUT/Names.h"


//  Construct an event processor.
TOutEventProcessor::TOutEventProcessor(OutProcessorId id, const
        TOutGeneralSpecification& specification, UINT vehicleMask)
  : TOutProcessor(id, specification),
    fVehicleObserver((TOutObserver*) NULL),
    fVehicleStorage(specification.GetRoot(), specification.GetName() + "." +
            kVehicleSuffix, TOutStorage::kWrite),
    fVehicleMask(vehicleMask)
{
}


//  Destroy an event processor.
TOutEventProcessor::~TOutEventProcessor()
{
}


//  Return the processor type.
TOutProcessor::EProcessorType TOutEventProcessor::GetProcessorType() const
{
    return kEventProcessor;
}


//  Begin recording output for this time step.
void TOutEventProcessor::RecordOutput(REAL)
{
    return;
}


//  Finish recording output for a vehicle.
void TOutEventProcessor::RecordVehicle()
{
    if (fVehicleStorage.GetOffset() == 0)
        fVehicleStorage.WriteHeader(fVehicleObserver->GetRecord());
    fVehicleStorage.Write(fVehicleObserver->GetRecord());
}


//  Return the vehicle observer.
TOutObserver& TOutEventProcessor::GetVehicleObserver()
{
    return *fVehicleObserver;
}

const TOutObserver& TOutEventProcessor::GetVehicleObserver() const
{
    return *fVehicleObserver;
}
```

```
//  Define the vehicle observer.
void TOutEventProcessor::SetVehicleObserver(TOutObserver& observer)
{
    fVehicleObserver = &observer;
}


//  Return the vehicle status mask.
UINT TOutEventProcessor::GetVehicleMask() const
{
    return fVehicleMask;
}
```

C. TOutEventRetriever Class

1. EventRetriever.h

```
//  Project: TRANSIMS
//  Subsystem: Simulation Output
//  $RCSfile: EventRetriever.h,v $
//  $Revision: 0.1 $
//  $Date: 1996/06/19 17:13:59 $
//  $State: Stab $
//  $Author: bwb $
//  U.S. Government Copyright 1995
//  All rights reserved


#ifndef TRANSIMS_OUT_EVENTRETRIEVER
#define TRANSIMS_OUT_EVENTRETRIEVER


//  Include TRANSIMS header files.
#include <GBL/Globals.h>
#include <OUT/Retriever.h>


//  An event retriever gets specific event data from storage and
//  coordinates its conversion and filtering.
class TOutEventRetriever
  : public TOutRetriever
{
  public:

    //  Construct a reader for the specified hosts and given specification,
    //  and network.
    TOutEventRetriever(const TOutStorage::HostSet& hosts, const
            TOutGeneralSpecification& specification, const TNetNetwork* network
            = NULL);

    //  Perform the retrieval on the specified writers.
    void Retrieve(TOutWriter& vehicleWriter, bool sort = TRUE);

  private:

    //  The retriever is connected to a vehicle storage.
    TOutStorage fVehicleStorage;
};


#endif // TRANSIMS_OUT_EVENTRETRIEVER
```

2. EventRetriever.C

```
//  Project: TRANSIMS
//  Subsystem: Simulation Output
//  $RCSfile: EventRetriever.C,v $
//  $Revision: 0.1 $
//  $Date: 1996/06/19 17:14:37 $
//  $State: Stab $
//  $Author: bwb $
//  U.S. Government Copyright 1996
```

```
//  All rights reserved


//  Include TRANSIMS header files.
#include <OUT/EventRetriever.h>
#include <OUT/Names.h>


//  Construct a reader for the specified hosts and given specification.
TOutEventRetriever::TOutEventRetriever(const TOutStorage::HostSet&
        hosts, const TOutGeneralSpecification& specification, const TNetNetwork*
        network)
  : TOutRetriever(specification, network),
    fVehicleStorage(hosts, specification.GetRoot(), specification.GetName() +
            "." + kVehicleSuffix)
{
}


//  Perform the retrieval on the specified writers.
void TOutEventRetriever::Retrieve(TOutWriter& vehicleWriter, bool sort)
{
    BasicRetrieve(fVehicleStorage, vehicleWriter, sort);
}
```

D. TOutEvolutionProcessor Class

1. EvolutionProcessor.h

```
//  Project: TRANSIMS
//  Subsystem: Simulation Output
//  $RCSfile: EvolutionProcessor.h,v $
//  $Revision: 0.6 $
//  $Date: 1996/06/19 17:16:38 $
//  $State: Stab $
//  $Author: bwb $
//  U.S. Government Copyright 1995
//  All rights reserved


#ifndef TRANSIMS_OUT_EVOLPROC
#define TRANSIMS_OUT_EVOLPROC


//  Include TRANSIMS header files.
#include "OUT/Processor.h"
#include "OUT/Storage.h"


//  Forward declarations.
class TOutEvolutionSpecification;
class TOutObserver;


//  An output evolution processor deals with evolving data such as
//  that needed for animation, waterfall plots, etc.
class TOutEvolutionProcessor
  :  public TOutProcessor
{
  public:

    //  Construct an evolution processor .
    TOutEvolutionProcessor(OutProcessorId, const TOutGeneralSpecification&);

    //  Destruct an evolution processor.
    virtual ~TOutEvolutionProcessor();

    //  Return the processor type.
    EProcessorType GetProcessorType() const;

    //  Begin recording output for this time step.
    virtual void RecordOutput(REAL time);

    //  Finish recording output for a node.
```

```
  virtual void RecordNode();

  //  Finish recording output for a link.
  virtual void RecordLink();

  //  Finish recording output for a vehicle.
  virtual void RecordVehicle();

  //  Finish recording output for an intersection.
  virtual void RecordIntersection();

  //  Finish recording output for a signal coordinator.
  virtual void RecordSignalCoordinator();

  //  Finish recording output for a signalized control.
  virtual void RecordSignalizedControl();

  //  Return the node observer.
  TOutObserver& GetNodeObserver();
  const TOutObserver& GetNodeObserver() const;

  //  Define the node observer.
  void SetNodeObserver(TOutObserver& observer);

  //  Return the link observer.
  TOutObserver& GetLinkObserver();
  const TOutObserver& GetLinkObserver() const;

  //  Define the link observer.
  void SetLinkObserver(TOutObserver& observer);

  //  Return the vehicle observer.
  TOutObserver& GetVehicleObserver();
  const TOutObserver& GetVehicleObserver() const;

  //  Define the vehicle observer.
  void SetVehicleObserver(TOutObserver& observer);

  //  Return the intersection observer.
  TOutObserver& GetIntersectionObserver();
  const TOutObserver& GetIntersectionObserver() const;

  //  Define the intersection observer.
  void SetIntersectionObserver(TOutObserver& observer);

  //  Return the signal coordinator observer.
  TOutObserver& GetSignalCoordinatorObserver();
  const TOutObserver& GetSignalCoordinatorObserver() const;

  //  Define the signal coordinator observer.
  void SetSignalCoordinatorObserver(TOutObserver& observer);

  //  Return the signalized control observer.
  TOutObserver& GetSignalizedControlObserver();
  const TOutObserver& GetSignalizedControlObserver() const;

  //  Define the signalized control observer.
  void SetSignalizedControlObserver(TOutObserver& observer);

private:

  //  An evolution processor has a node observer.
  TOutObserver* fNodeObserver;

  //  An evolution processor has a link observer.
  TOutObserver* fLinkObserver;

  //  An evolution processor has a vehicle observer.
  TOutObserver* fVehicleObserver;

  //  An evolution processor has an intersection observer.
  TOutObserver* fIntersectionObserver;

  //  An evolution processor has a signal coordinator observer.
  TOutObserver* fSignalCoordinatorObserver;
```

```
    //  An evolution processor has a signalized control observer.
    TOutObserver* fSignalizedControlObserver;

    //  An evolution processor is connected to a vehicle storage.
    TOutStorage fVehicleStorage;

    //  An evolution processor is connected to an intersection storage.
    TOutStorage fIntersectionStorage;

    //  An evolution processor is connected to a signalized control storage.
    TOutStorage fSignalStorage;
};



#endif // TRANSIMS_OUT_EVOLPROC
```

2. EvolutionProcessor.C

```
//  Project: TRANSIMS
//  Subsystem: Simulation Output
//  $RCSfile: EvolutionProcessor.C,v $
//  $Revision: 0.6 $
//  $Date: 1996/06/19 17:15:42 $
//  $State: Stab $
//  $Author: bwb $
//  U.S. Government Copyright 1995
//  All rights reserved


//  Include TRANSIMS header files.
#include "OUT/EvolutionProcessor.h"
#include "OUT/Exception.h"
#include "OUT/Observer.h"
#include "OUT/Names.h"


//  Construct an evolution processor.
TOutEvolutionProcessor::TOutEvolutionProcessor(OutProcessorId id, const
        TOutGeneralSpecification& spec)
  : TOutProcessor(id, spec),
    fNodeObserver((TOutObserver*) NULL),
    fLinkObserver((TOutObserver*) NULL),
    fVehicleObserver((TOutObserver*) NULL),
    fIntersectionObserver((TOutObserver*) NULL),
    fSignalCoordinatorObserver((TOutObserver*) NULL),
    fSignalizedControlObserver((TOutObserver*) NULL),
    fVehicleStorage(spec.GetRoot(), spec.GetName() + "." + kVehicleSuffix,
            TOutStorage::kWrite),
    fIntersectionStorage(spec.GetRoot(), spec.GetName() + "." +
            kIntersectionSuffix, TOutStorage::kWrite),
    fSignalStorage(spec.GetRoot(), spec.GetName() + "." + kSignalSuffix,
            TOutStorage::kWrite)
{
}


//  Destruct an evolution processor.
TOutEvolutionProcessor::~TOutEvolutionProcessor()
{
}


//  Return the processor type.
TOutProcessor::EProcessorType TOutEvolutionProcessor::GetProcessorType() const
{
    return kEvolutionProcessor;
}


//  Begin recording output for this time step.
void TOutEvolutionProcessor::RecordOutput(REAL time)
{
    throw TOutException(
            "Client must override TOutEvolutionProcessor::RecordOutput(time).");
```

```
}


//  Finish recording output for a node.
void TOutEvolutionProcessor::RecordNode()
{
}


//  Finish recording output for a link.
void TOutEvolutionProcessor::RecordLink()
{
}


//  Finish recording output for a vehicle.
void TOutEvolutionProcessor::RecordVehicle()
{
    if (fVehicleStorage.GetOffset() == 0)
        fVehicleStorage.WriteHeader(fVehicleObserver->GetRecord());
    fVehicleStorage.Write(fVehicleObserver->GetRecord());
}


//  Finish recording output for an intersection.
void TOutEvolutionProcessor::RecordIntersection()
{
    if (fIntersectionStorage.GetOffset() == 0)
        fIntersectionStorage.WriteHeader(fIntersectionObserver->GetRecord());
    fIntersectionStorage.Write(fIntersectionObserver->GetRecord());
}


//  Finish recording output for a signal coordinator.
void TOutEvolutionProcessor::RecordSignalCoordinator()
{
}


//  Finish recording output for a signalized control.
void TOutEvolutionProcessor::RecordSignalizedControl()
{
    if (fSignalStorage.GetOffset() == 0)
        fSignalStorage.WriteHeader(fSignalizedControlObserver->GetRecord());
    fSignalStorage.Write(fSignalizedControlObserver->GetRecord());
}


//  Return the node observer.
TOutObserver& TOutEvolutionProcessor::GetNodeObserver()
{
    return *fNodeObserver;
}

const TOutObserver& TOutEvolutionProcessor::GetNodeObserver() const
{
    return *fNodeObserver;
}


//  Define the node observer.
void TOutEvolutionProcessor::SetNodeObserver(TOutObserver& o)
{
    fNodeObserver = &o;
}


//  Return the link observer.
TOutObserver& TOutEvolutionProcessor::GetLinkObserver()
{
    return *fLinkObserver;
}

const TOutObserver& TOutEvolutionProcessor::GetLinkObserver() const
{
    return *fLinkObserver;
```

```
}


//  Define the link observer.
void TOutEvolutionProcessor::SetLinkObserver(TOutObserver& o)
{
    fLinkObserver = &o;
}


//  Return the vehicle observer.
TOutObserver& TOutEvolutionProcessor::GetVehicleObserver()
{
    return *fVehicleObserver;
}

const TOutObserver& TOutEvolutionProcessor::GetVehicleObserver() const
{
    return *fVehicleObserver;
}


//  Define the vehicle observer.
void TOutEvolutionProcessor::SetVehicleObserver(TOutObserver& o)
{
    fVehicleObserver = &o;
}


//  Return the intersection observer.
TOutObserver& TOutEvolutionProcessor::GetIntersectionObserver()
{
    return *fIntersectionObserver;
}

const TOutObserver& TOutEvolutionProcessor::GetIntersectionObserver() const
{
    return *fIntersectionObserver;
}


//  Define the intersection observer.
void TOutEvolutionProcessor::SetIntersectionObserver(TOutObserver& o)
{
    fIntersectionObserver = &o;
}


//  Return the signal coordinator observer.
TOutObserver& TOutEvolutionProcessor::GetSignalCoordinatorObserver()
{
    return *fSignalCoordinatorObserver;
}

const TOutObserver& TOutEvolutionProcessor::GetSignalCoordinatorObserver() const
{
    return *fSignalCoordinatorObserver;
}


//  Define the signal coordinator observer.
void TOutEvolutionProcessor::SetSignalCoordinatorObserver (TOutObserver& o)
{
    fSignalCoordinatorObserver = &o;
}


//  Return the signalized control observer.
TOutObserver& TOutEvolutionProcessor::GetSignalizedControlObserver()
{
    return *fSignalizedControlObserver;
}

const TOutObserver& TOutEvolutionProcessor::GetSignalizedControlObserver() const
{
    return *fSignalizedControlObserver;
```

```
}


//  Define the signalized control observer.
void TOutEvolutionProcessor::SetSignalizedControlObserver(TOutObserver& o)
{
    fSignalizedControlObserver = &o;
}
```

E. TOutEvolutionRetriever Class

1. EvolutionRetriever.h

```
//  Project: TRANSIMS
//  Subsystem: Simulation Output
//  $RCSfile: EvolutionRetriever.h,v $
//  $Revision: 0.8 $
//  $Date: 1996/06/19 17:21:54 $
//  $State: Stab $
//  $Author: bwb $
//  U.S. Government Copyright 1995
//  All rights reserved


#ifndef TRANSIMS_OUT_EVOLUTIONRETRIEVER
#define TRANSIMS_OUT_EVOLUTIONRETRIEVER


//  Include TRANSIMS header files.
#include <GBL/Globals.h>
#include <OUT/Retriever.h>


//  An evolution retriever gets specific trajectory data from storage and
//  coordinates its conversion and filtering.
class TOutEvolutionRetriever
  : public TOutRetriever
{
  public:

    //  Construct a reader for the specified hosts and given specification,
    //  and network.
    TOutEvolutionRetriever(const TOutStorage::HostSet& hosts, const
            TOutGeneralSpecification& specification, const TNetNetwork* network
            = NULL);

    //  Perform the retrieval on the specified writers.
    void Retrieve(TOutWriter& vehicleWriter, TOutWriter& intersectionWriter,
                  TOutWriter& signalWriter, bool sort = TRUE);

  private:

    //  The retriever is connected to a vehicle storage.
    TOutStorage fVehicleStorage;

    //  The retriever is connected to an intersection storage.
    TOutStorage fIntersectionStorage;

    //  The retriever is connected to a signal storage.
    TOutStorage fSignalStorage;
};


#endif // TRANSIMS_OUT_EVOLUTIONRETRIEVER
```

2. EvolutionRetriever.C

```
//  Project: TRANSIMS
//  Subsystem: Simulation Output
//  $RCSfile: EvolutionRetriever.C,v $
//  $Revision: 0.9 $
//  $Date: 1996/06/19 17:21:35 $
//  $State: Stab $
```

```
//  $Author: bwb $
//  U.S. Government Copyright 1995
//  All rights reserved


//  Include TRANSIMS header files.
#include <OUT/EvolutionRetriever.h>
#include <OUT/Names.h>


//  Construct a reader for the specified hosts and given specification.
TOutEvolutionRetriever::TOutEvolutionRetriever(const TOutStorage::HostSet&
        hosts, const TOutGeneralSpecification& specification, const TNetNetwork*
        network)
  : TOutRetriever(specification, network),
    fVehicleStorage(hosts, specification.GetRoot(), specification.GetName() +
            "." + kVehicleSuffix),
    fIntersectionStorage(hosts, specification.GetRoot(), specification.GetName()
            + "." + kIntersectionSuffix),
    fSignalStorage(hosts, specification.GetRoot(), specification.GetName() +
            "." + kSignalSuffix)
{
}


//  Perform the retrieval on the specified writers.
void TOutEvolutionRetriever::Retrieve(TOutWriter& vehicleWriter,
        TOutWriter& intersectionWriter, TOutWriter& signalWriter, bool sort)
{
    BasicRetrieve(fVehicleStorage, vehicleWriter, sort);
    BasicRetrieve(fIntersectionStorage, intersectionWriter, sort);
    BasicRetrieve(fSignalStorage, signalWriter, sort);
}
```

F. TOutException Class

1. Exception.h

```
//  Project: TRANSIMS
//  Subsystem: Simulation Output
//  $RCSfile: Exception.h,v $
//  $Revision: 0.6 $
//  $Date: 1996/06/19 17:22:59 $
//  $State: Stab $
//  $Author: bwb $
//  U.S. Government Copyright 1995
//  All rights reserved


#ifndef TRANSIMS_OUT_EXCEPTION
#define TRANSIMS_OUT_EXCEPTION


//  Include TRANSIMS header files.
#include <GBL/Globals.h>


//  An output exception signals the failure of a member function.
class TOutException
{
  public:

    //  Construct an exception with the specified message text.
    TOutException(const string& message = "Simulation output error.");

    //  Construct a copy of the specified exception.
//  TOutException(const TOutException& exception);

    //  Make the exception a copy of the specified exception.
//  TOutException& operator=(const TOutException& exception);

    //  Return the message text for the exception.
    const string& GetMessage() const;
```

```
  private:

    //  Each exception has a message.
    string fMessage;
};


//  This exception is thrown when a storage operation fails.
class TOutStorageFailure
  : public TOutException
{
  public:

    //  Construct an exception with the specified message text.
    TOutStorageFailure(const string& message = "Storage failure.");

    //  Construct a copy of the specified exception.
//  TOutStorageFailure(const TOutStorageFailure& exception);

    //  Make the exception a copy of the specified exception.
//  TOutStorageFailure& operator=(const TOutStorageFailure& exception);
};


//  This exception is thrown when a writer operation fails.
class TOutWriterFailure
  : public TOutException
{
  public:

    //  Construct an exception with the specified message text.
    TOutWriterFailure(const string& message = "Writer failure.");

    //  Construct a copy of the specified exception.
//  TOutWriterFailure(const TOutWriterFailure& exception);

    //  Make the exception a copy of the specified exception.
//  TOutWriterFailure& operator=(const TOutWriterFailure& exception);
};


//  This exception is thrown when the processor type is invalid.
class TOutInvalidProcessor
  : public TOutException
{
  public:

    //  Construct an exception with the specified message text.
    TOutInvalidProcessor(const string& message = "Invalid Processor.");

    //  Construct a copy of the specified exception.
//  TOutInvalidProcessor(const TOutInvalidProcessor& exception);

    //  Make the exception a copy of the specified exception.
//  TOutInvalidProcessor& operator=(const TOutInvalidProcessor& exception);
};


#endif // TRANSIMS_OUT_EXCEPTION
```

2. Exception.C

```
//  Project: TRANSIMS
//  Subsystem: Simulation Output
//  $RCSfile: Exception.C,v $
//  $Revision: 0.6 $
//  $Date: 1996/06/19 17:22:32 $
//  $State: Stab $
//  $Author: bwb $
//  U.S. Government Copyright 1995
//  All rights reserved


//  Include TRANSIMS header files.
#include <OUT/Exception.h>
```

```
//  Construct an exception with the specified message text.
TOutException::TOutException(const string& message)
  : fMessage(message)
{
}


//  Return the message text for the exception.
const string& TOutException::GetMessage() const
{
    return fMessage;
}


//  Construct a "storage failure" exception with the specified message text.
TOutStorageFailure::TOutStorageFailure(const string& message)
  : TOutException(message)
{
}


//  Construct a "writer failure" exception with the specified message text.
TOutWriterFailure::TOutWriterFailure(const string& message)
  : TOutException(message)
{
}


//  Construct an "invalid processor" exception with the specified message text.
TOutInvalidProcessor::TOutInvalidProcessor(const string& message)
  : TOutException(message)
{
}
```

G. TOutFactory Class

1. Factory.h

```
//  Project: TRANSIMS
//  Subsystem: Simulation Output
//  $RCSfile: Factory.h,v $
//  $Revision: 0.4 $
//  $Date: 1996/06/19 17:23:26 $
//  $State: Stab $
//  $Author: bwb $
//  U.S. Government Copyright 1995
//  All rights reserved


#ifndef TRANSIMS_OUT_FACTORY
#define TRANSIMS_OUT_FACTORY


//  Include TRANSIMS header files.
#include "OUT/Id.h"


//  Forward declarations.
class TOutEvolutionProcessor;
class TOutEventProcessor;
class TOutSummaryProcessor;
class TOutLinkEvolutionObserver;
class TOutVehicleObserver;
class TOutNodeEvolutionObserver;
class TOutIntersectionObserver;
class TOutSignalCoordinatorEvolutionObserver;
class TOutSignalizedControlObserver;
class TOutLinkSpaceObserver;
class TOutLinkTimeObserver;
class TOutGeneralSpecification;
```

```
//  An output factory allocates and constructs new output objects.
class TOutFactory
{
  public:

    //  Construct a factory.
    TOutFactory ();

    //  Destruct a factory.
    virtual ~TOutFactory();

    //  Return a new evolution processor from the specification.
    virtual TOutEvolutionProcessor* NewEvolutionProcessor(OutProcessorId id,
            const TOutGeneralSpecification& specification) = 0;

    //  Return a new event processor from the specification.
    virtual TOutEventProcessor* NewEventProcessor(OutProcessorId id,
            const TOutGeneralSpecification& specification) = 0;

    //  Return a new summary processor from the specification.
    virtual TOutSummaryProcessor* NewSummaryProcessor(OutProcessorId id,
            const TOutGeneralSpecification& specification) = 0;

    //  Return a new link evolution observer.
    virtual TOutLinkEvolutionObserver* NewLinkEvolutionObserver(OutObserverId
            id) = 0;

    //  Return a new vehicle observer.
    virtual TOutVehicleObserver* NewVehicleObserver(OutObserverId id) = 0;

    //  Return a new node evolution observer.
    virtual TOutNodeEvolutionObserver* NewNodeEvolutionObserver(OutObserverId
            id) = 0;

    //  Return a new intersection observer.
    virtual TOutIntersectionObserver* NewIntersectionObserver(OutObserverId id)
            = 0;

    //  Return a new signal coordinator evolution observer.
    virtual TOutSignalCoordinatorEvolutionObserver*
            NewSignalCoordinatorEvolutionObserver(OutObserverId id) = 0;

    //  Return a new signalized control observer.
    virtual TOutSignalizedControlObserver*
            NewSignalizedControlObserver(OutObserverId id) = 0;

    //  Return a new link space observer.
    virtual TOutLinkSpaceObserver* NewLinkSpaceObserver(OutObserverId id) = 0;

    //  Return a new link time observer.
    virtual TOutLinkTimeObserver* NewLinkTimeObserver(OutObserverId id) = 0;
};


#endif // TRANSIMS_OUT_FACTORY
```

2. Factory.C

```
//  Project: TRANSIMS
//  Subsystem: Simulation Output
//  $RCSfile: Factory.C,v $
//  $Revision: 0.3 $
//  $Date: 1996/06/19 17:23:15 $
//  $State: Stab $
//  $Author: bwb $
//  U.S. Government Copyright 1995
//  All rights reserved


//  Include TRANSIMS header files.
#include "OUT/Factory.h"


//  Construct a factory.
TOutFactory::TOutFactory()
```

```
{
}


//  Destruct a factory.
TOutFactory::~TOutFactory()
{
}
```

H. TOutGeneralSpecification Class

1. GeneralSpecification.h

```
//  Project: TRANSIMS
//  Subsystem: Simulation Output
//  $RCSfile: GeneralSpecification.h,v $
//  $Revision: 0.6 $
//  $Date: 1996/06/19 17:24:30 $
//  $State: Stab $
//  $Author: bwb $
//  U.S. Government Copyright 1995
//  All rights reserved


#ifndef TRANSIMS_OUT_GENERALSPECIFICATION
#define TRANSIMS_OUT_GENERALSPECIFICATION


//  Include Booch Components header files.
#include <BCStoreM.h>
#include <BCSetU.h>


//  Include TRANSIMS header files.
#include <GBL/Globals.h>
#include <NET/Id.h>
#include <NET/Rectangle.h>
#include <OUT/GeneralSpecificationReader.h>


//  The general specification defines the frequency and extent of data to be
//  collected or retrieved in both space and time.
class TOutGeneralSpecification
{
  public:

    //  Type definitions.
    typedef BC_TUnboundedSet<NetNodeId, 10000U, BC_CManaged> NodeIdSet;
    typedef BC_TSetActiveIterator<NetNodeId> NodeIdSetIterator;
    typedef BC_TUnboundedSet<NetLinkId, 10000U, BC_CManaged> LinkIdSet;
    typedef BC_TSetActiveIterator<NetLinkId> LinkIdSetIterator;

    //  Time constants.
    static const REAL kMinusInfinity;
    static const REAL kPlusInfinity;

    //  Construct a general specification from a reader.
    TOutGeneralSpecification(TOutGeneralSpecificationReader& reader);

    //  Return the root for the specification.
    const string& GetRoot() const;

    //  Return the name for the specification.
    const string& GetName() const;

    //  Return whether data should be collected for the specified time.
    bool CollectForTime(REAL time) const;

    //  Return whether data should be sampled at the specified time.
    bool SampleForTime(REAL time) const;

    //  Return whether the specified time is the start time.
    bool IsAtStartTime(REAL time) const;
```

```
    //  Return the box length.
    REAL GetBoxLength() const;

    //  Return whether data should be collected for the specified point in
    //  space.
    bool CollectForPoint(const TGeoPoint& point) const;

    //  Return whether data should be collected for the specified node.
    bool CollectForNode(NetNodeId id) const;

    //  Return whether data should be collected for the specified link.
    bool CollectForLink(NetLinkId id) const;

  private:

    //  Each specification has a root.
    string fRoot;

    //  Each specification has a name.
    string fName;

    //  Each specification has a minumum time.
    REAL fTimeMinimum;

    //  Each specification has a maximum time.
    REAL fTimeMaximum;

    //  Each specification has a time step.
    REAL fTimeStep;

    //  Each specification has a time sample.
    REAL fTimeSample;

    //  Each specification has a box length.
    REAL fBoxLength;

    //  Each specification has a collection region.
    TGeoRectangle fRegion;

    //  Each specification has a set of node ids.
    NodeIdSet fNodes;

    //  Each specification has a set of link ids.
    LinkIdSet fLinks;
};


#endif // TRANSIMS_OUT_GENERALSPECIFICATION
```

2. GeneralSpecification.C

```
//  Project: TRANSIMS
//  Subsystem: Simulation Output
//  $RCSfile: GeneralSpecification.C,v $
//  $Revision: 0.7 $
//  $Date: 1996/06/19 17:24:13 $
//  $State: Stab $
//  $Author: bwb $
//  U.S. Government Copyright 1995
//  All rights reserved


//  Include TRANSIMS header files.
#include <OUT/GeneralSpecification.h>


//  Define the hash function for node ids.
static BC_Index NodeIdHashValue(const NetNodeId& id)
{
    return BC_Index(id);
}


//  Define the hash function for link ids.
static BC_Index LinkIdHashValue(const NetLinkId& id)
```

```
{
    return BC_Index(id);
}


//  Time constants.
const REAL TOutGeneralSpecification::kMinusInfinity = (UINT) -2;
const REAL TOutGeneralSpecification::kPlusInfinity = (UINT) -1;


//  Construct a general specification from a reader.
TOutGeneralSpecification::TOutGeneralSpecification(
        TOutGeneralSpecificationReader& reader)
  : fRoot(reader.GetRoot()),
    fName(reader.GetName()),
    fTimeMinimum(reader.GetMinimumTime()),
    fTimeMaximum(reader.GetMaximumTime()),
    fTimeStep(reader.GetTimeStep()),
    fTimeSample(reader.GetTimeSample()),
    fBoxLength(reader.GetBoxLength()),
    fRegion(reader.GetRegion()),
    fNodes(reader.GetNodes()),
    fLinks(reader.GetLinks())
{
}


//  Return the root for the specification.
const string& TOutGeneralSpecification::GetRoot() const
{
    return fRoot;
}


//  Return the name for the specification.
const string& TOutGeneralSpecification::GetName() const
{
    return fName;
}


//  Return whether data should be collected for the specified time.
bool TOutGeneralSpecification::CollectForTime(REAL time) const
{
    if (fTimeMinimum != kMinusInfinity && time < fTimeMinimum)
        return FALSE;
    if (fTimeMaximum != kPlusInfinity && time > fTimeMaximum)
        return FALSE;
    const REAL start = fTimeMinimum != kMinusInfinity ? fTimeMinimum : 0;
    return int(100 * (time - start)) % int(100 * fTimeStep) == 0;
}


//  Return whether data should be sampled at the specified time.
bool TOutGeneralSpecification::SampleForTime(REAL time) const
{
    if (fTimeMinimum != kMinusInfinity && time < fTimeMinimum)
        return FALSE;
    if (fTimeMaximum != kPlusInfinity && time > fTimeMaximum)
        return FALSE;
    const REAL start = fTimeMinimum != kMinusInfinity ? fTimeMinimum : 0;
    return int(100 * (time - start)) % int(100 * fTimeSample) == 0;
}


//  Return whether the specified time is the start time.
bool TOutGeneralSpecification::IsAtStartTime(REAL time) const
{
    return fTimeMinimum == time;
}


//  Return the box length.
REAL TOutGeneralSpecification::GetBoxLength() const
{
    return fBoxLength;
```

```
}


//  Return whether data should be collected for the specified point in space.
bool TOutGeneralSpecification::CollectForPoint(const TGeoPoint& point) const
{
    return fRegion.Contains(point);
}


//  Return whether data should be collected for the specified node.
bool TOutGeneralSpecification::CollectForNode(NetNodeId id) const
{
    return fNodes.IsMember(id);
}


//  Return whether data should be collected for the specified link.
bool TOutGeneralSpecification::CollectForLink(NetLinkId id) const
{
    return fLinks.IsMember(id);
}
```

I. TOutGeneralSpecificationReader Class

1. GeneralSpecificationReader.h

```
//  Project: TRANSIMS
//  Subsystem: Simulation Output
//  $RCSfile: GeneralSpecificationReader.h,v $
//  $Revision: 0.6 $
//  $Date: 1996/06/19 17:26:00 $
//  $State: Stab $
//  $Author: bwb $
//  U.S. Government Copyright 1995
//  All rights reserved


#ifndef TRANSIMS_OUT_GENERALSPECIFICATIONREADER
#define TRANSIMS_OUT_GENERALSPECIFICATIONREADER


//  Include Booch Components header files.
#include <BCStoreM.h>
#include <BCSetU.h>


//  Include TRANSIMS header files.
#include <GBL/Globals.h>
#include <DBS/Accessor.h>
#include <NET/Id.h>
#include <NET/Rectangle.h>


//  Forward declarations.
class TOutSpecificationReader;


//  A general specification reader reads specifications from the database.
class TOutGeneralSpecificationReader
{
  public:

    //  Type definitions.
    typedef BC_TUnboundedSet<NetNodeId, 10000U, BC_CManaged> NodeIdSet;
    typedef BC_TSetActiveIterator<NetNodeId> NodeIdSetIterator;
    typedef BC_TUnboundedSet<NetLinkId, 10000U, BC_CManaged> LinkIdSet;
    typedef BC_TSetActiveIterator<NetLinkId> LinkIdSetIterator;

    //  Processor types.
    enum EProcessorType {kEvolutionProcessor, kEventProcessor,
            kSummaryProcessor};

    //  Construct a general specification reader.
```

```
    TOutGeneralSpecificationReader(TOutSpecificationReader& reader);

    //  Construct a copy of the given general specification reader.
//  TNetGeneralSpecificationReader(const TNetGeneralSpecificationReader&
//          reader);

    //  Make the reader a copy of the given general specification reader.
//  TNetGeneralSpecificationReader& operator=(const
//          TNetGeneralSpecificationReader& reader);

    //  Reset the iteration over the table.
    void Reset();

    //  Get the next specification in the table.
    void GetNextSpecification();

    //  Return whether there are any more specifications in the table.
    bool MoreSpecifications() const;

    //  Return the root of the specification.
    string GetRoot() const;

    //  Return the name of the specification.
    string GetName() const;

    //  Return the minimum time of the specification.
    REAL GetMinimumTime() const;

    //  Return the maximum time of the specification.
    REAL GetMaximumTime() const;

    //  Return the time step of the specification.
    REAL GetTimeStep() const;

    //  Return the time sampling of the specification.
    REAL GetTimeSample() const;

    //  Return the box length of the specification.
    REAL GetBoxLength() const;

    //  Return the geographic region of the specification.
    TGeoRectangle GetRegion() const;

    //  Return the nodes in the specification.
    NodeIdSet GetNodes() const;

    //  Return the links in the specificaiton.
    LinkIdSet GetLinks() const;

    //  Return the processor type in the specification.  A TOutInvalidProcessor
    //  exception is thrown if the processor is not a valid type.
    EProcessorType GetProcessorType() const;

  private:

    //  Each general specification reader has a database table accessor.
    TDbAccessor fAccessor;

    //  Each general specification reader accesses a node table.
    TDbAccessor fNodeAccessor;

    //  Each general specification reader accesses a link table.
    TDbAccessor fLinkAccessor;

    //  Each general specification has a processor type.
    const TDbField fProcessorField;

    //  Each general specification has a root.
    const TDbField fRootField;

    //  Each general specification has a name.
    const TDbField fNameField;

    //  Each general specification has a minimum time.
    const TDbField fMinimumTimeField;
```

```
    //  Each general specification has a maximum time.
    const TDbField fMaximumTimeField;

    //  Each general specification has a time step.
    const TDbField fTimeStepField;

    //  Each general specification has a time sample.
    const TDbField fTimeSampleField;

    //  Each general specification has a box length.
    const TDbField fBoxLengthField;

    //  Each general specification has a minimum abscissa.
    const TDbField fMinimumAbscissaField;

    //  Each general specification has a maximum abscissa.
    const TDbField fMaximumAbscissaField;

    //  Each general specification has a minimum ordinate.
    const TDbField fMinimumOrdinateField;

    //  Each general specification has a maximum ordinate.
    const TDbField fMaximumOrdinateField;
};


#endif // TRANSIMS_OUT_GENERALSPECIFICATIONREADER
```

2. GeneralSpecificationReader.C

```
//  Project: TRANSIMS
//  Subsystem: Simulation Output
//  $RCSfile: GeneralSpecificationReader.C,v $
//  $Revision: 0.6 $
//  $Date: 1996/06/19 17:25:26 $
//  $State: Stab $
//  $Author: bwb $
//  U.S. Government Copyright 1995
//  All rights reserved


//  Include TRANSIMS header files.
#include <OUT/Exception.h>
#include <OUT/SpecificationReader.h>
#include <OUT/GeneralSpecificationReader.h>


//  Define the hash function for node ids.
static BC_Index NodeIdHashValue(const NetNodeId& id)
{
    return BC_Index(id);
}


//  Define the hash function for link ids.
static BC_Index LinkIdHashValue(const NetLinkId& id)
{
    return BC_Index(id);
}


//  Construct a general specification reader.
TOutGeneralSpecificationReader::TOutGeneralSpecificationReader(
        TOutSpecificationReader& reader)
  : fAccessor(reader.GetGeneralTable()),
    fNodeAccessor(reader.GetNodeTable()),
    fLinkAccessor(reader.GetLinkTable()),
    fProcessorField(reader.GetGeneralTable().GetField("PROCESSOR")),
    fRootField(reader.GetGeneralTable().GetField("ROOT")),
    fNameField(reader.GetGeneralTable().GetField("NAME")),
    fMinimumTimeField(reader.GetGeneralTable().GetField("TIMEMIN")),
    fMaximumTimeField(reader.GetGeneralTable().GetField("TIMEMAX")),
    fTimeStepField(reader.GetGeneralTable().GetField("TIMESTP")),
    fTimeSampleField(reader.GetGeneralTable().GetField("TIMESMP")),
    fBoxLengthField(reader.GetGeneralTable().GetField("BOXLEN")),
```

```
    fMinimumAbscissaField(reader.GetGeneralTable().GetField("ABSCISSAMN")),
    fMaximumAbscissaField(reader.GetGeneralTable().GetField("ABSCISSAMX")),
    fMinimumOrdinateField(reader.GetGeneralTable().GetField("ORDINATEMN")),
    fMaximumOrdinateField(reader.GetGeneralTable().GetField("ORDINATEMX"))
{
}


//  Reset the iteration over the table.
void TOutGeneralSpecificationReader::Reset()
{
    fAccessor.GotoFirst();
}


//  Get the next specification in the table.
void TOutGeneralSpecificationReader::GetNextSpecification()
{
    fAccessor.GotoNext();
}


//  Return whether there are any more specifications in the table.
bool TOutGeneralSpecificationReader::MoreSpecifications() const
{
    return fAccessor.IsAtRecord();
}


//  Return the root of the specification.
string TOutGeneralSpecificationReader::GetRoot() const
{
    string root;
    fAccessor.GetField(fRootField, root);
    return root;
}


//  Return the name of the specification.
string TOutGeneralSpecificationReader::GetName() const
{
    string name;
    fAccessor.GetField(fNameField, name);
    return name;
}


//  Return the minimum time of the specification.
REAL TOutGeneralSpecificationReader::GetMinimumTime() const
{
    REAL time;
    fAccessor.GetField(fMinimumTimeField, time);
    return time;
}


//  Return the maximum time of the specification.
REAL TOutGeneralSpecificationReader::GetMaximumTime() const
{
    REAL time;
    fAccessor.GetField(fMaximumTimeField, time);
    return time;
}


//  Return the time step of the specification.
REAL TOutGeneralSpecificationReader::GetTimeStep() const
{
    REAL time;
    fAccessor.GetField(fTimeStepField, time);
    return time;
}


//  Return the time sampling of the specification.
REAL TOutGeneralSpecificationReader::GetTimeSample() const
```

```
{
    REAL time;
    fAccessor.GetField(fTimeSampleField, time);
    return time;
}


//  Return the box length of the specification.
REAL TOutGeneralSpecificationReader::GetBoxLength() const
{
    REAL length;
    fAccessor.GetField(fBoxLengthField, length);
    return length;
}


//  Return the geographic region of the specification.
TGeoRectangle TOutGeneralSpecificationReader::GetRegion() const
{
    REAL x0, x1, y0, y1;
    fAccessor.GetField(fMinimumAbscissaField, x0);
    fAccessor.GetField(fMaximumAbscissaField, x1);
    fAccessor.GetField(fMinimumOrdinateField, y0);
    fAccessor.GetField(fMaximumOrdinateField, y1);
    return TGeoRectangle(TGeoPoint(x0, y0), TGeoPoint(x1, y1));
}


//  Return the nodes in the specification.
TOutGeneralSpecificationReader::NodeIdSet
        TOutGeneralSpecificationReader::GetNodes() const
{
    NodeIdSet nodes(NodeIdHashValue);

    string current;
    fAccessor.GetField(fNameField, current);
    const TDbField nameField("NAME");
    const TDbField nodeField("NODE");

    for (((TDbAccessor&) fNodeAccessor).GotoFirst(); ((TDbAccessor&)
            fNodeAccessor).IsAtRecord(); ((TDbAccessor&)
            fNodeAccessor).GotoNext()) {
        string name;
        fNodeAccessor.GetField(nameField, name);
        if (name == current) {
            NetNodeId id;
            fNodeAccessor.GetField(nodeField, id);
            nodes.Add(id);
        }
    }

    return nodes;
}


//  Return the links in the specification.
TOutGeneralSpecificationReader::LinkIdSet
        TOutGeneralSpecificationReader::GetLinks() const
{
    LinkIdSet links(LinkIdHashValue);

    string current;
    fAccessor.GetField(fNameField, current);
    const TDbField nameField("NAME");
    const TDbField linkField("LINK");

    for (((TDbAccessor&) fLinkAccessor).GotoFirst(); ((TDbAccessor&)
            fLinkAccessor).IsAtRecord(); ((TDbAccessor&)
            fLinkAccessor).GotoNext()) {
        string name;
        fLinkAccessor.GetField(nameField, name);
        if (name == current) {
            NetLinkId id;
            fLinkAccessor.GetField(linkField, id);
            links.Add(id);
        }
```

```
    }

    return links;
}


//  Return the processor type in the specification.  A TOutInvalidProcessor
//  exception is thrown if the processor is not a valid type.
TOutGeneralSpecificationReader::EProcessorType
        TOutGeneralSpecificationReader::GetProcessorType() const
{
    string processor;
    fAccessor.GetField(fProcessorField, processor);
    if (processor == "Evolution")
        return kEvolutionProcessor;
    else if (processor == "Event")
        return kEventProcessor;
    else if (processor == "Summary")
        return kSummaryProcessor;
    else
        throw TOutInvalidProcessor();
}
```

J. TOutIntersectionObserver Class

1. IntersectionObserver.h

```
//  Project: TRANSIMS
//  Subsystem: Simulation Output
//  $RCSfile: IntersectionObserver.h,v $
//  $Revision: 0.2 $
//  $Date: 1996/06/19 17:27:07 $
//  $State: Stab $
//  $Author: bwb $
//  U.S. Government Copyright 1995
//  All rights reserved


#ifndef TRANSIMS_OUT_INTOBS
#define TRANSIMS_OUT_INTOBS


//  Include TRANSIMS header files.
#include "OUT/Observer.h"
#include "NET/Id.h"


//  An intersection observer observes data related to intersections.
class TOutIntersectionObserver
  : public TOutObserver
{
  public:

    //  Construct an intersection observer
    TOutIntersectionObserver(OutObserverId id);

    //  Destruct an intersection observer.
    virtual ~TOutIntersectionObserver();

  protected:

  //  Define values for the fields being observed.
  //  The Observe method will use these methods.

    //  Define the vehicle’s id.
    void SetId(UINT id);

    //  Define the id of the link the vehicle entered from.
    void SetLink(NetLinkId link);

    //  Define the lane number the vehicle entered from.
    void SetLane(NetLaneNumber lane);

    //  Define the id of the node the intersection is associated with.
```

```
    void SetNode(NetNodeId node);

    //  Define the index of the vehicle’s position in the queue.
    void SetQIndex(BYTE index);
};


#endif // TRANSIMS_OUT_INTOBS
```

2. IntersectionObserver.C

```
//  Project: TRANSIMS
//  Subsystem: Simulation Output
//  $RCSfile: IntersectionObserver.C,v $
//  $Revision: 0.2 $
//  $Date: 1996/06/19 17:26:43 $
//  $State: Stab $
//  $Author: bwb $
//  U.S. Government Copyright 1995
//  All rights reserved


//  Include TRANSIMS header files.
#include "OUT/IntersectionObserver.h"
#include "OUT/Record.h"
#include "OUT/Names.h"


//  Construct an intersection observer
TOutIntersectionObserver::TOutIntersectionObserver(OutObserverId id)
  : TOutObserver(id)
{
    SetTime(-99.);
    SetId(-99);
    SetLink(-99);
    SetLane(-99);
    SetNode(-99);
    SetQIndex(-99);
}


//  Destruct an intersection observer.
TOutIntersectionObserver::~TOutIntersectionObserver()
{
}


//  Define the vehicle’s id.
void TOutIntersectionObserver::SetId(UINT id)
{
    GetRecord().SetField(kVehicleField, id);
}


//  Define the id of the link the vehicle entered from.
void TOutIntersectionObserver::SetLink(NetLinkId id)
{
    GetRecord().SetField(kLinkField, id);
}


//  Define the lane number the vehicle entered from.
void TOutIntersectionObserver::SetLane(NetLaneNumber n)
{
    GetRecord().SetField(kLaneField, n);
}


//  Define the id of the node the intersection is associated with.
void TOutIntersectionObserver::SetNode(NetNodeId id)
{
    GetRecord().SetField(kNodeField, id);
}
```

```
//  Define the index of the vehicle's position in the queue.
void TOutIntersectionObserver::SetQIndex(BYTE i)
{
    GetRecord().SetField(kQIndexField, i);
}
```

K. TOutLinkEvolutionObserver Class

1. TOutLinkEvolutionObserver.h

```
//  Project: TRANSIMS
//  Subsystem: Simulation Output
//  $RCSfile: LinkEvolutionObserver.h,v $
//  $Revision: 0.4 $
//  $Date: 1996/06/19 17:28:06 $
//  $State: Stab $
//  $Author: bwb $
//  U.S. Government Copyright 1995
//  All rights reserved


#ifndef TRANSIMS_OUT_LINKEVOLOBS
#define TRANSIMS_OUT_LINKEVOLOBS


//  Include TRANSIMS header files.
#include "OUT/Observer.h"


//  A link evolution observer observes evolving data related to links.
class TOutLinkEvolutionObserver
  : public TOutObserver
{
  public:

    //  Construct a link evolution observer.
    TOutLinkEvolutionObserver(OutObserverId id);

    //  Destruct a link evolution observer.
    virtual ~TOutLinkEvolutionObserver();
};


#endif // TRANSIMS_OUT_LINKEVOLOBS
```

2. TOutLinkEvolutionObserver.C

```
//  Project: TRANSIMS
//  Subsystem: Simulation Output
//  $RCSfile: LinkEvolutionObserver.C,v $
//  $Revision: 0.4 $
//  $Date: 1996/06/19 17:27:49 $
//  $State: Stab $
//  $Author: bwb $
//  U.S. Government Copyright 1995
//  All rights reserved


//  Include TRANSIMS header files.
#include "OUT/LinkEvolutionObserver.h"


//  Construct a link evolution observer.
TOutLinkEvolutionObserver::TOutLinkEvolutionObserver(OutObserverId id)
  : TOutObserver(id)
{
}


//  Destruct a link evolution observer.
TOutLinkEvolutionObserver::~TOutLinkEvolutionObserver()
{
}
```

L. TOutLinkSpaceObserver Class

1. LinkSpaceObserver.h

```
//  Project: TRANSIMS
//  Subsystem: Simulation Output
//  $RCSfile: LinkSpaceObserver.h,v $
//  $Revision: 0.1 $
//  $Date: 1996/06/19 17:29:18 $
//  $State: Stab $
//  $Author: bwb $
//  U.S. Government Copyright 1995
//  All rights reserved


#ifndef TRANSIMS_OUT_LINKSPACEOBSERVER
#define TRANSIMS_OUT_LINKSPACEOBSERVER


//  Include Booch Components header files.
#include <BCStoreM.h>
#include <BCCollU.h>


//  Include TRANSIMS header files.
#include "OUT/Observer.h"
#include "NET/Id.h"


//  A link space observer summarizes vehicle spatial data on a link.
class TOutLinkSpaceObserver
  : public TOutObserver
{
  public:

    //  Construct a link space observer.
    TOutLinkSpaceObserver(OutObserverId id);

    //  Destroy a link space observer.
    virtual ~TOutLinkSpaceObserver();

  protected:

    //  Return whether the observer has been initialized.
    bool IsInitialized() const;

    //  Clear the box data for the observer.
    void ClearBoxData();

    //  Set the link id.
    void SetLink(NetLinkId id);

    //  Set the departure node id.
    void SetNode(NetNodeId id);

    //  Set the link, box, and cell lengths.
    void SetLengths(REAL linkLength, REAL boxLength, REAL cellLength = 0);

    //  Add a vehicle.
    void AddVehicle(REAL distance, REAL velocity);

    //  Report the observations to the specified processor.
    void ReportObservations(TOutProcessor& processor);

  private:

    //  A box datum stores the tally for a box.
    class TBoxDatum
    {
      public:

        //  Construct a box datum.
        TBoxDatum()
          : fCount(0),
```

```
            fVelocityTotal(0)
        {
        }

        //  Make the box datum a copy of the specified box datum.
        //TBoxDatum(const TBoxDatum& boxDatum);

        //  Assign a box datum.
        //TBoxDatum& operator=(const TBoxDatum& boxDatum);

        //  Test equality of box datums.
        bool operator==(const TBoxDatum& boxDatum) const
        {
            return this == &boxDatum;
        }
        //  Test inequality of box datums.
        bool operator!=(const TBoxDatum& boxDatum) const
        {
            return this != &boxDatum;
        }

        //  Clear the data in a box.
        void Clear()
        {
            fCount = 0;
            fVelocityTotal = 0;
        }

        //  Add a measurement to the box datum.
        void Add(REAL velocity)
        {
            fCount += 1;
            fVelocityTotal += velocity;
        }

        //  Return the count.
        UINT GetCount() const
        {
            return fCount;
        }

        //  Return the velocity total.
        REAL GetVelocityTotal() const
        {
            return fVelocityTotal;
        }

      private:

        //  Each box has a vehicle count.
        UINT fCount;

        //  Each box has a velocity total.
        REAL fVelocityTotal;
    };

    //  Type definitions.
    typedef BC_TUnboundedCollection<TBoxDatum, BC_CManaged> BoxDatumCollection;

    //  Each observer has a link length.
    REAL fLinkLength;

    //  Each observer has a box length.
    REAL fBoxLength;

    //  Each observer has box data.
    BoxDatumCollection fBoxData;
};


#endif // TRANSIMS_OUT_LINKSPACEOBSERVER
```

2. LinkSpaceObserver.C

```
//  Project: TRANSIMS
```

```
//  Subsystem: Simulation Output
//  $RCSfile: LinkSpaceObserver.C,v $
//  $Revision: 0.2 $
//  $Date: 1996/06/20 14:32:15 $
//  $State: Rel $
//  $Author: bwb $
//  U.S. Government Copyright 1996
//  All rights reserved


//  Include TRANSIMS header files.
#include "OUT/LinkSpaceObserver.h"
#include "OUT/SummaryProcessor.h"
#include "OUT/Record.h"
#include "OUT/Names.h"


//  Construct a link space observer.
TOutLinkSpaceObserver::TOutLinkSpaceObserver(OutObserverId id)
  : TOutObserver(id)
{
}

//  Destroy a link space observer.
TOutLinkSpaceObserver::~TOutLinkSpaceObserver()
{
}


//  Return whether the observer has been initialized.
bool TOutLinkSpaceObserver::IsInitialized() const
{
    return !fBoxData.IsEmpty();
}


//  Clear the box data for the observer.
void TOutLinkSpaceObserver::ClearBoxData()
{
    fBoxData.Clear();
    for (int i = 0; i < fLinkLength / fBoxLength; ++i)
        fBoxData.Append(TBoxDatum());
}


//  Set the link id.
void TOutLinkSpaceObserver::SetLink(NetLinkId id)
{
    GetRecord().SetField(kLinkField, id);
}


//  Set the link, box, and cell lengths.
void TOutLinkSpaceObserver::SetLengths(REAL linkLength, REAL boxLength, REAL
        cellLength)
{
    if (cellLength == 0)
        fLinkLength = linkLength;
    else {
        fLinkLength = cellLength * int(linkLength / cellLength);
        if (fLinkLength == 0)
            fLinkLength = cellLength;
    }
    fBoxLength = boxLength;

    ClearBoxData();
}


//  Set the departure node id.
void TOutLinkSpaceObserver::SetNode(NetNodeId id)
{
    GetRecord().SetField(kNodeField, id);
}
```

```
//  Add a vehicle.
void TOutLinkSpaceObserver::AddVehicle(REAL distance, REAL velocity)
{
    const int i = int((fLinkLength - distance) / fBoxLength);
    fBoxData[i].Add(velocity);
}


//  Report the observations to the specified processor.
void TOutLinkSpaceObserver::ReportObservations(TOutProcessor& processor)
{
    for (int i = 0; i < fLinkLength / fBoxLength; ++i) {
        TOutRecord& record = GetRecord();
        record.SetField(kDistanceField, fLinkLength - i * fBoxLength);
        record.SetField(kCountField, fBoxData[i].GetCount());
        record.SetField(kSumField, fBoxData[i].GetVelocityTotal());
        ((TOutSummaryProcessor&) processor).RecordSpace(*this);
    }
}
```

M. TOutLinkTimeObserver Class

1. LinkTimeObserver.h

```
//  Project: TRANSIMS
//  Subsystem: Simulation Output
//  $RCSfile: LinkTimeObserver.h,v $
//  $Revision: 0.1 $
//  $Date: 1996/06/19 17:30:28 $
//  $State: Stab $
//  $Author: bwb $
//  U.S. Government Copyright 1996
//  All rights reserved


#ifndef TRANSIMS_OUT_LINKTIMEOBSERVER
#define TRANSIMS_OUT_LINKTIMEOBSERVER


//  Include Booch Components header files.
#include <BCStoreM.h>
#include <BCCollU.h>


//  Include TRANSIMS header files.
#include "OUT/Observer.h"
#include "NET/Id.h"


//  A link time observer records vehicle travel times on a link.
class TOutLinkTimeObserver
  : public TOutObserver
{
  public:

    //  Construct a link time observer.
    TOutLinkTimeObserver(OutObserverId id);

    //  Destroy a link time observer.
    virtual ~TOutLinkTimeObserver();

    //  Add a vehicle.
    void AddVehicle(REAL time);

  protected:

    //  Clear the data for the observer.
    void ClearData();

    //  Set the link id.
    void SetLink(NetLinkId id);

    //  Set the departure node id.
    void SetNode(NetNodeId id);
```

```
    //  Report the observations to the specified processor.
    void ReportObservations(TOutProcessor& processor);

  private:

    //  Each link time observer has a vehicle count.
    UINT fCount;

    //  Each link time observer has a total of travel times.
    REAL fSum;

    //  Each link time observer has a total of travel time squares.
    REAL fSumSquares;

};


#endif // TRANSIMS_OUT_LINKTIMEOBSERVER
```

2. LinkTimeObserver.C

```
//  Project: TRANSIMS
//  Subsystem: Simulation Output
//  $RCSfile: LinkTimeObserver.C,v $
//  $Revision: 0.2 $
//  $Date: 1996/06/20 14:32:15 $
//  $State: Rel $
//  $Author: bwb $
//  U.S. Government Copyright 1996
//  All rights reserved


//  Include TRANSIMS header files.
#include "OUT/LinkTimeObserver.h"
#include "OUT/SummaryProcessor.h"
#include "OUT/Record.h"
#include "OUT/Names.h"


//  Construct a link time observer.
TOutLinkTimeObserver::TOutLinkTimeObserver(OutObserverId id)
  : TOutObserver(id),
    fCount(0),
    fSum(0),
    fSumSquares(0)
{
}


//  Destroy a link time observer.
TOutLinkTimeObserver::~TOutLinkTimeObserver()
{
}


//  Clear the data for the observer.
void TOutLinkTimeObserver::ClearData()
{
    fCount = 0;
    fSum = 0;
    fSumSquares = 0;
}


//  Set the link id.
void TOutLinkTimeObserver::SetLink(NetLinkId id)
{
    GetRecord().SetField(kLinkField, id);
}


//  Set the departure node id.
void TOutLinkTimeObserver::SetNode(NetNodeId id)
{
```

```
    GetRecord().SetField(kNodeField, id);
}


//  Add a vehicle.
void TOutLinkTimeObserver::AddVehicle(REAL time)
{
    fCount += 1;
    fSum += time;
    fSumSquares += time * time;
}


//  Report the observations to the specified processor.
void TOutLinkTimeObserver::ReportObservations(TOutProcessor& processor)
{
    TOutRecord& record = GetRecord();
    record.SetField(kCountField, fCount);
    record.SetField(kSumField, fSum);
    record.SetField(kSumSquaresField, fSumSquares);
    ((TOutSummaryProcessor&) processor).RecordTime(*this);
}
```

N. TOutNodeEvolutionObserver Class

1. NodeEvolutionObserver.h

```
//  Project: TRANSIMS
//  Subsystem: Simulation Output
//  $RCSfile: NodeEvolutionObserver.h,v $
//  $Revision: 0.2 $
//  $Date: 1996/06/19 17:32:17 $
//  $State: Stab $
//  $Author: bwb $
//  U.S. Government Copyright 1995
//  All rights reserved


#ifndef TRANSIMS_OUT_NODEEVOLOBS
#define TRANSIMS_OUT_NODEEVOLOBS


//  Include TRANSIMS header files.
#include "OUT/Observer.h"


//  A node evolution observer observes evolving data related to nodes.
class TOutNodeEvolutionObserver
  : public TOutObserver
{
  public:

    //  Construct a node evolution observer.
    TOutNodeEvolutionObserver(OutObserverId id);

    //  Destruct a node evolution observer.
    virtual ~TOutNodeEvolutionObserver();
};


#endif // TRANSIMS_OUT_NODEEVOLOBS
```

2. NodeEvolutionObserver.C

```
//  Project: TRANSIMS
//  Subsystem: Simulation Output
//  $RCSfile: NodeEvolutionObserver.C,v $
//  $Revision: 0.2 $
//  $Date: 1996/06/19 17:31:59 $
//  $State: Stab $
//  $Author: bwb $
//  U.S. Government Copyright 1995
//  All rights reserved
```

```
//  Include TRANSIMS header files.
#include "OUT/NodeEvolutionObserver.h"


//  Construct a node evolution observer.
TOutNodeEvolutionObserver::TOutNodeEvolutionObserver(OutObserverId id)
  : TOutObserver(id)
{
}


//  Destruct a node evolution observer.
TOutNodeEvolutionObserver::~TOutNodeEvolutionObserver()
{
}
```

O. TOutObserver Class

1. Observer.h

```
//  Project: TRANSIMS
//  Subsystem: Simulation Output
//  $RCSfile: Observer.h,v $
//  $Revision: 0.4 $
//  $Date: 1996/06/19 17:33:26 $
//  $State: Stab $
//  $Author: bwb $
//  U.S. Government Copyright 1995
//  All rights reserved


#ifndef TRANSIMS_OUT_OBSERVER
#define TRANSIMS_OUT_OBSERVER


//  Include TRANSIMS header files.
#include "GBL/Globals.h"
#include "OUT/Id.h"
#include "OUT/Record.h"


//  Include Booch Component header files.
#include "BCStoreM.h"
#include "BCMapU.h"


//  Forward declarations.
class TOutProcessor;


//  An observer converts data from the object to which it is attached
//  into the generic form understood by the output subsystem.
class TOutObserver
{
  public:

    //  Type definitions
    typedef BC_TUnboundedMap<TOutProcessor*, TOutObserver*, 1U, BC_CManaged>
            ObserverMap;
    typedef BC_TMapActiveIterator<TOutProcessor*, TOutObserver*>
            ObserverMapIterator;

    //  Construct an observer.
    TOutObserver(OutObserverId id);

    //  Destruct an observer.
    virtual ~TOutObserver();

    //  An observer has an observe function for noting the values of data
    //  members of interest in the observed object.
    //  This virtual function in the basic representation must be
    //  overridden in the view.
```

```
    virtual void Observe(const void* object, const TOutProcessor& processor) =
            0;

    //  Return the observer's id.
    OutObserverId GetId();
    const OutObserverId GetId() const;

    //  Return the associated record.
    TOutRecord& GetRecord();
    const TOutRecord& GetRecord() const;

    //  Return the next unused observer id.
    static OutObserverId GetNextId();

  protected:

    //  Define the record's time.
    void SetTime(REAL time);

  private:

    //  Do not allow observers to be copied.
    TOutObserver(const TOutObserver&) {}

    //  Do not allow observers to be assigned.
    TOutObserver& operator=(const TOutObserver&) {return *this;}

    //  An observer has an id.
    OutObserverId fId;

    //  An observer has an output record.
    TOutRecord fRecord;

    //  Keep track of the next observer id.
    static OutObserverId fNextId;
};


#endif // TRANSIMS_OUT_OBSERVER
```

2. Observer.C

```
//  Project: TRANSIMS
//  Subsystem: Simulation Output
//  $RCSfile: Observer.C,v $
//  $Revision: 0.5 $
//  $Date: 1996/06/19 17:32:51 $
//  $State: Stab $
//  $Author: bwb $
//  U.S. Government Copyright 1995
//  All rights reserved


//  Include TRANSIMS header files.
#include "OUT/Observer.h"
#include "OUT/Names.h"


//  Include C header files.
#include <stddef.h>


//  Keep track of the next observer id.
OutObserverId TOutObserver::fNextId = 1000;


//  Construct an observer.
TOutObserver::TOutObserver(OutObserverId id)
  : fId(id),
    fRecord()
{
}


//  Destruct an observer.
```

```
TOutObserver::~TOutObserver()
{
}


//  Return the observer's id.
OutObserverId TOutObserver::GetId()
{
    return fId;
}

const OutObserverId TOutObserver::GetId() const
{
    return fId;
}


//  Return the associated record.
TOutRecord& TOutObserver::GetRecord()
{
    return fRecord;
}

const TOutRecord& TOutObserver::GetRecord() const
{
    return fRecord;
}


//  Return the next unused observer id.
OutObserverId TOutObserver::GetNextId()
{
    return fNextId++;
}


//  Define the record's time.
void TOutObserver::SetTime(REAL time)
{
    GetRecord().SetField(kTimeField, time);
}
```

P. TOutProcessor Class

1. Processor.h

```
//  Project: TRANSIMS
//  Subsystem: Simulation Output
//  $RCSfile: Processor.h,v $
//  $Revision: 0.4 $
//  $Date: 1996/06/19 17:34:39 $
//  $State: Stab $
//  $Author: bwb $
//  U.S. Government Copyright 1995
//  All rights reserved


#ifndef TRANSIMS_OUT_PROCESSOR
#define TRANSIMS_OUT_PROCESSOR


//  Include TRANSIMS header files.
#include "GBL/Globals.h"
#include "OUT/Id.h"
#include "OUT/GeneralSpecification.h"


//  An output processor coordinates the processing of domain information
//  into a domain-independent representation that is filtered and summarized
//  before storage.
class TOutProcessor
{
  public:
```

```
    //  Processor types.
    enum EProcessorType {kEvolutionProcessor, kEventProcessor,
            kSummaryProcessor};

    //  Construct a processor.
    TOutProcessor(OutProcessorId id, const TOutGeneralSpecification&
            specification);

    //  Destruct a processor.
    virtual ~TOutProcessor();

    //  Return the processor type.
    virtual EProcessorType GetProcessorType() const = 0;

    //  Begin recording output for this time.
    virtual void RecordOutput(REAL time) = 0;

    //  Return the processor's id.
    OutProcessorId GetId();
    const OutProcessorId GetId() const;

    //  Return the next unused processor id.
    static OutProcessorId GetNextId();

    //  Return general specification.
    TOutGeneralSpecification& GetGeneralSpecification ();
    const TOutGeneralSpecification& GetGeneralSpecification() const;

  private:

    //  Do not allow processors to be copied.
    //TOutProcessor(const TOutProcessor&) {}

    //  Do not allow processors to be assigned.
    TOutProcessor& operator=(const TOutProcessor&) {return *this;}

    //  A processor has an id.
    OutProcessorId  fId;

    //  A processor has a general output specification.
    TOutGeneralSpecification fGeneralSpecification;

    //  Keep track of next processor id.
    static OutProcessorId fNextId;
};


#endif // TRANSIMS_OUT_PROCESSOR
```

2. Processor.C

```
//  Project: TRANSIMS
//  Subsystem: Simulation Output
//  $RCSfile: Processor.C,v $
//  $Revision: 0.4 $
//  $Date: 1996/06/19 17:34:19 $
//  $State: Stab $
//  $Author: bwb $
//  U.S. Government Copyright 1995
//  All rights reserved


//  Include TRANSIMS header files.
#include "OUT/Processor.h"


//  Initialize the next processor id.
OutProcessorId  TOutProcessor::fNextId = 1;


//  Return the next unused processor id.
OutProcessorId TOutProcessor::GetNextId()
{
    return fNextId++;
}
```

```
//  Construct a processor.
TOutProcessor::TOutProcessor(OutProcessorId id, const TOutGeneralSpecification&
        spec)
  : fId(id),
    fGeneralSpecification(spec)
{
}


//  Destruct a processor.
TOutProcessor::~TOutProcessor()
{
}


//  Return the processor's id.
OutProcessorId TOutProcessor::GetId()
{
    return fId;
}

const OutProcessorId TOutProcessor::GetId() const
{
    return fId;
}


//  Return the general specification.
TOutGeneralSpecification& TOutProcessor::GetGeneralSpecification()
{
    return fGeneralSpecification;
}

const TOutGeneralSpecification& TOutProcessor::GetGeneralSpecification() const
{
    return fGeneralSpecification;
}
```

Q. TOutRecord Class

1. Record.h

```
//  Project: TRANSIMS
//  Subsystem: Simulation Output
//  $RCSfile: Record.h,v $
//  $Revision: 0.6 $
//  $Date: 1996/06/19 17:35:23 $
//  $State: Stab $
//  $Author: bwb $
//  U.S. Government Copyright 1995
//  All rights reserved


#ifndef TRANSIMS_OUT_RECORD
#define TRANSIMS_OUT_RECORD


//  Include Booch Components header files.
#include <BCStoreM.h>
#include <BCMapU.h>


//  Include DBtools.h++ header files.
#include <rw/db/value.h>


//  Fix for Booch Maps.
inline int operator!=(const RWDBValue& a, const RWDBValue& b)
{
    return !a.isEqual(&b);
}
```

```
//  Include TRANSIMS header files.
#include <GBL/Globals.h>


//  This class is used for storing the values of a collection of fields.
class TOutRecord
{
  public:

    //  Type definitions.
    typedef BC_TUnboundedMap<const string, RWDBValue, 30U, BC_CManaged>
            FieldMap;
    typedef BC_TMapActiveIterator<const string, RWDBValue> FieldMapIterator;

    //  Field types.
    enum Type {kNoType = RWDBValue::NoType, kChar = RWDBValue::Char,
            kUnsignedChar = RWDBValue::UnsignedChar, kShort = RWDBValue::Short,
            kUnsignedShort = RWDBValue::UnsignedShort, kInt = RWDBValue::Int,
            kUnsignedInt = RWDBValue::UnsignedInt, kLong = RWDBValue::Long,
            kUnsignedLong = RWDBValue::UnsignedLong, kFloat = RWDBValue::Float,
            kDouble = RWDBValue::Double, kString = RWDBValue::String};

    //  Construct a record.
    TOutRecord();

    //  Make a copy of the given record.
//  TOutRecord(const TOutRecord& record);

    //  Make the record a copy of the given record.
//  TOutRecord& operator=(const TOutRecord& record);

    //  Set the value of the specified field.
    void SetField(const string& field, char value);
    void SetField(const string& field, unsigned char value);
    void SetField(const string& field, short value);
    void SetField(const string& field, unsigned short value);
    void SetField(const string& field, int value);
    void SetField(const string& field, unsigned int value);
    void SetField(const string& field, long value);
    void SetField(const string& field, unsigned long value);
    void SetField(const string& field, float value);
    void SetField(const string& field, double value);
    void SetField(const string& field, const string& value);
    void SetField(const string& field);

    //  Get the value of the specified field.
    void GetField(const string& field, char& value) const;
    void GetField(const string& field, unsigned char& value) const;
    void GetField(const string& field, short& value) const;
    void GetField(const string& field, unsigned short& value) const;
    void GetField(const string& field, int& value) const;
    void GetField(const string& field, unsigned int& value) const;
    void GetField(const string& field, long& value) const;
    void GetField(const string& field, unsigned long& value) const;
    void GetField(const string& field, float& value) const;
    void GetField(const string& field, double& value) const;
    void GetField(const string& field, string& value) const;

    //  Return the type of the specified field.
    Type GetType(const string& field) const;

    //  Return the map for the record.
    FieldMap& GetMap();
    const FieldMap& GetMap() const;

    //  Return an iterator for the record’s map.
    FieldMapIterator GetIterator() const;

  private:

    //  Each record has a field map holding the current values of the fields.
    FieldMap fFields;
};
```

```
#endif // TRANSIMS_OUT_RECORD
```

2. Record.C

```
//  Project: TRANSIMS
//  Subsystem: Simulation Output
//  $RCSfile: Record.C,v $
//  $Revision: 0.5 $
//  $Date: 1996/06/19 17:35:17 $
//  $State: Stab $
//  $Author: bwb $
//  U.S. Government Copyright 1995
//  All rights reserved


//  Include TRANSIMS header files.
#include <OUT/Record.h>



//  Define the hash function for strings.
static BC_Index StringHashValue(const string& s)
{
    return HashValue(s);
}


//  Construct a record.
TOutRecord::TOutRecord()
  : fFields(StringHashValue)
{
}

//  Set the value of the specified field.

void TOutRecord::SetField(const string& field, char value)
{
    if (fFields.IsBound(field))
        fFields.Rebind(field, RWDBValue(value));
    else
        fFields.Bind(field, RWDBValue(value));
}

void TOutRecord::SetField(const string& field, unsigned char value)
{
    if (fFields.IsBound(field))
        fFields.Rebind(field, RWDBValue(value));
    else
        fFields.Bind(field, RWDBValue(value));
}

void TOutRecord::SetField(const string& field, short value)
{
    if (fFields.IsBound(field))
        fFields.Rebind(field, RWDBValue(value));
    else
        fFields.Bind(field, RWDBValue(value));
}

void TOutRecord::SetField(const string& field, unsigned short value)
{
    if (fFields.IsBound(field))
        fFields.Rebind(field, RWDBValue(value));
    else
        fFields.Bind(field, RWDBValue(value));
}

void TOutRecord::SetField(const string& field, int value)
{
    if (fFields.IsBound(field))
        fFields.Rebind(field, RWDBValue(value));
    else
        fFields.Bind(field, RWDBValue(value));
}
```

```
void TOutRecord::SetField(const string& field, unsigned int value)
{
    if (fFields.IsBound(field))
        fFields.Rebind(field, RWDBValue(value));
    else
        fFields.Bind(field, RWDBValue(value));
}

void TOutRecord::SetField(const string& field, long value)
{
    if (fFields.IsBound(field))
        fFields.Rebind(field, RWDBValue(value));
    else
        fFields.Bind(field, RWDBValue(value));
}

void TOutRecord::SetField(const string& field, unsigned long value)
{
    if (fFields.IsBound(field))
        fFields.Rebind(field, RWDBValue(value));
    else
        fFields.Bind(field, RWDBValue(value));
}

void TOutRecord::SetField(const string& field, float value)
{
    if (fFields.IsBound(field))
        fFields.Rebind(field, RWDBValue(value));
    else
        fFields.Bind(field, RWDBValue(value));
}

void TOutRecord::SetField(const string& field, double value)
{
    if (fFields.IsBound(field))
        fFields.Rebind(field, RWDBValue(value));
    else
        fFields.Bind(field, RWDBValue(value));
}

void TOutRecord::SetField(const string& field, const string& value)
{
    if (fFields.IsBound(field))
        fFields.Rebind(field, RWDBValue(value));
    else
        fFields.Bind(field, RWDBValue(value));
}

void TOutRecord::SetField(const string& field)
{
    if (fFields.IsBound(field))
        fFields.Rebind(field, RWDBValue());
    else
        fFields.Bind(field, RWDBValue());
}


//  Get the value of the specified field.

void TOutRecord::GetField(const string& field, char& value) const
{
    const RWDBValue* const p = fFields.ValueOf(field);
    if (p != NULL && p->canConvert(RWDBValue::Char))
        value = p->asChar();
    else
        value = 0;
}

void TOutRecord::GetField(const string& field, unsigned char& value) const
{
    const RWDBValue* const p = fFields.ValueOf(field);
    if (p != NULL && p->canConvert(RWDBValue::UnsignedChar))
        value = p->asUnsignedChar();
    else
        value = 0;
}
```

```
void TOutRecord::GetField(const string& field, short& value) const
{
    const RWDBValue* const p = fFields.ValueOf(field);
    if (p != NULL && p->canConvert(RWDBValue::Short))
        value = p->asShort();
    else
        value = 0;
}

void TOutRecord::GetField(const string& field, unsigned short& value) const
{
    const RWDBValue* const p = fFields.ValueOf(field);
    if (p != NULL && p->canConvert(RWDBValue::UnsignedShort))
        value = p->asUnsignedShort();
    else
        value = 0;
}

void TOutRecord::GetField(const string& field, int& value) const
{
    const RWDBValue* const p = fFields.ValueOf(field);
    if (p != NULL && p->canConvert(RWDBValue::Int))
        value = p->asInt();
    else
        value = 0;
}

void TOutRecord::GetField(const string& field, unsigned int& value) const
{
    const RWDBValue* const p = fFields.ValueOf(field);
    if (p != NULL && p->canConvert(RWDBValue::UnsignedInt))
        value = p->asUnsignedInt();
    else
        value = 0;
}

void TOutRecord::GetField(const string& field, long& value) const
{
    const RWDBValue* const p = fFields.ValueOf(field);
    if (p != NULL && p->canConvert(RWDBValue::Long))
        value = p->asLong();
    else
        value = 0;
}

void TOutRecord::GetField(const string& field, unsigned long& value) const
{
    const RWDBValue* const p = fFields.ValueOf(field);
    if (p != NULL && p->canConvert(RWDBValue::UnsignedLong))
        value = p->asUnsignedLong();
    else
        value = 0;
}

void TOutRecord::GetField(const string& field, float& value) const
{
    const RWDBValue* const p = fFields.ValueOf(field);
    if (p != NULL && p->canConvert(RWDBValue::Float))
        value = p->asFloat();
    else
        value = 0;
}

void TOutRecord::GetField(const string& field, double& value) const
{
    const RWDBValue* const p = fFields.ValueOf(field);
    if (p != NULL && p->canConvert(RWDBValue::Double))
        value = p->asDouble();
    else
        value = 0;
}

void TOutRecord::GetField(const string& field, string& value) const
{
    const RWDBValue* const p = fFields.ValueOf(field);
```

```
    if (p != NULL && p->canConvert(RWDBValue::String))
        value = p->asString();
    else
        value = "";
}


//  Return the type of the specified field.
TOutRecord::Type TOutRecord::GetType(const string& field) const
{
    const RWDBValue* const p = fFields.ValueOf(field);
    return p != NULL ? (Type) p->type() : kNoType;
}


//  Return the map for the record.
TOutRecord::FieldMap& TOutRecord::GetMap()
{
    return fFields;
}

const TOutRecord::FieldMap& TOutRecord::GetMap() const
{
    return fFields;
}


//  Return an iterator for the record's map.
TOutRecord::FieldMapIterator TOutRecord::GetIterator() const
{
    return FieldMapIterator(fFields);
}
```

R. TOutRetriever Class

1. Retriever.h

```
//  Project: TRANSIMS
//  Subsystem: Simulation Output
//  $RCSfile: Retriever.h,v $
//  $Revision: 0.3 $
//  $Date: 1996/06/19 17:36:07 $
//  $State: Stab $
//  $Author: bwb $
//  U.S. Government Copyright 1995
//  All rights reserved


#ifndef TRANSIMS_OUT_RETRIEVER
#define TRANSIMS_OUT_RETRIEVER


//  Include TRANSIMS header files.
#include <GBL/Globals.h>
#include <NET/Network.h>
#include <OUT/GeneralSpecification.h>
#include <OUT/Record.h>
#include <OUT/Storage.h>
#include <OUT/Writer.h>


//  An output retriever acts as an interface to coordinate the retrieval of
//  data stored in a simulation.
class TOutRetriever
{
  public:

    //  Destroy the retriever.
    virtual ~TOutRetriever() {}

    //  Perform the retrieval.
//  virtual void Retrieve() = 0;

  protected:
```

```
    //  Construct a retriever based on the given specification and network.
    TOutRetriever(const TOutGeneralSpecification& specification, const
            TNetNetwork* network = NULL);

    //  Return the general specification.
    TOutGeneralSpecification& GetGeneralSpecification();
    const TOutGeneralSpecification& GetGeneralSpecification() const;

    //  Return the network, if any.
    const TNetNetwork* GetNetwork() const;

    //  Retrieve data from the specified storage and put it in the specified
    //  writer, sorting it if indicated, and performing time and space filtering
    //  if indicated.
    void BasicRetrieve(TOutStorage& storage, TOutWriter& writer, bool sort, bool
        filter = TRUE);

  private:

    //  Do not allow retrievers to be copied.
//  TOutRetriever(const TOutRetriever&) {}

    //  Do not allow retrievers to be assigned.
    TOutRetriever& operator=(const TOutRetriever&) {return *this;}

    //  Each retriever has a general specification.
    TOutGeneralSpecification fGeneralSpecification;

    //  Each retriever may refer to a network.
    const TNetNetwork* fNetwork;
};


#endif // TRANSIMS_OUT_RETRIEVER
```

2. Retriever.C

```
//  Project: TRANSIMS
//  Subsystem: Simulation Output
//  $RCSfile: Retriever.C,v $
//  $Revision: 0.3 $
//  $Date: 1996/06/19 17:35:56 $
//  $State: Stab $
//  $Author: bwb $
//  U.S. Government Copyright 1995
//  All rights reserved


//  Include TRANSIMS header files.
#include <GBL/Globals.h>
#include <OUT/Names.h>
#include <OUT/Retriever.h>


//  Construct a retriever based on the given specification.
TOutRetriever::TOutRetriever(const TOutGeneralSpecification& specification,
        const TNetNetwork* network)
  : fGeneralSpecification(specification),
    fNetwork(network)
{
}


//  Return the general specification.
TOutGeneralSpecification& TOutRetriever::GetGeneralSpecification()
{
    return fGeneralSpecification;
}

const TOutGeneralSpecification& TOutRetriever::GetGeneralSpecification() const
{
    return fGeneralSpecification;
}
```

```
//  Return the network, if any.
const TNetNetwork* TOutRetriever::GetNetwork() const
{
    return fNetwork;
}


//  Retrieve data from the specified storage and put it in the specified
//  writer, sorting it if indicated, and performing time and space filtering
//  if indicated.
void TOutRetriever::BasicRetrieve(TOutStorage& storage, TOutWriter& writer,
        bool sort, bool filter)
{
    //  Allocate caches.
    const int nh = storage.GetHosts().Extent();
    TOutStorage::HostHandle* handleCache = new TOutStorage::HostHandle[nh];
    TOutRecord* recordCache = new TOutRecord[nh];

    //  Setup caches.
    TOutRecord record;
    TOutStorage::HostSetIterator i(storage.GetHosts());
    int h = 0;
    for (i.Reset(); !i.IsDone(); i.Next()) {
        const string& host = *i.CurrentItem();
        storage.Seek(host, TOutStorage::kBegin);
        storage.ReadHeader(host, record);
        handleCache[h++] = storage.GetHostHandle(host);
    }

    //  Setup output.
    TOutWriter::FieldCollection fields;
    for (TOutRecord::FieldMapIterator f = record.GetIterator(); !f.IsDone();
            f.Next())
        fields.Append(*f.CurrentItem());
    const TOutGeneralSpecification& specification = GetGeneralSpecification();

    //  Sort the output if necessary.
    if (sort) {

        REAL currentTime = TOutGeneralSpecification::kMinusInfinity;
        REAL time;
        NetLinkId link;
        for (h = 0; h < nh; ++h) {
            recordCache[h] = record;
            storage.Read(handleCache[h], recordCache[h]);
        }
        do {
            currentTime = TOutGeneralSpecification::kMinusInfinity;
            for (h = 0; h < nh; ++h) {
                if (storage.AtEnd(handleCache[h]))
                    break;
                recordCache[h].GetField(kTimeField, time);
                currentTime = currentTime < time ? currentTime : time;
            }
            if (currentTime == TOutGeneralSpecification::kMinusInfinity)
                break;
            for (h = 0; h < nh; ++h) {
                TOutRecord& cachedRecord = recordCache[h];
                do {
                    cachedRecord.GetField(kTimeField, time);
                    cachedRecord.GetField(kLinkField, link);
                    if (storage.AtEnd(handleCache[h]) || time != currentTime)
                        break;
                    //ISSUE(bwb): no geographic filtering yet.
                    if (filter && specification.CollectForTime(time) &&
                            specification.CollectForLink(link))
                        writer.Write(cachedRecord, fields);
                    storage.Read(handleCache[h], cachedRecord);
                } while(TRUE);
            }
        } while (TRUE);

    //  Otherwise, copy the data from the storage to the writer.
    } else {
```

```
        REAL time;
        NetLinkId link;
        for (h = 0; h < nh; ++h) {
            for (storage.Read(handleCache[h], record);
                    !storage.AtEnd(handleCache[h]);
                    storage.Read(handleCache[h], record)) {
                record.GetField(kTimeField, time);
                record.GetField(kLinkField, link);
                if (specification.CollectForTime(time) &&
                        specification.CollectForLink(link))
                    writer.Write(record, fields);
            }
        }

    }

        //  Free caches.
        delete [] handleCache;
        delete [] recordCache;
}
```

S. TOutSignalCoordinatorEvolutionObserver Class

1. SignalCoordinatorEvolutionObserver.h

```
//  Project: TRANSIMS
//  Subsystem: Simulation Output
//  $RCSfile: SignalCoordinatorEvolutionObserver.h,v $
//  $Revision: 0.2 $
//  $Date: 1996/06/19 17:37:18 $
//  $State: Stab $
//  $Author: bwb $
//  U.S. Government Copyright 1995
//  All rights reserved


#ifndef TRANSIMS_OUT_COORDEVOLOBS
#define TRANSIMS_OUT_COORDEVOLOBS


//  Include TRANSIMS header files.
#include "OUT/Observer.h"


//  A signal coordinator evolution observer observes evolving data related to
//  signal coordinators.
class TOutSignalCoordinatorEvolutionObserver
  : public TOutObserver
{
  public:

    //  Construct a signal coordinator evolution observer.
    TOutSignalCoordinatorEvolutionObserver(OutObserverId id);

    //  Destruct a signal coordinator evolution observer.
    virtual ~TOutSignalCoordinatorEvolutionObserver();
};


#endif // TRANSIMS_OUT_COORDEVOLOBS
```

2. SignalCoordinatorEvolutionObserver.C

```
//  Project: TRANSIMS
//  Subsystem: Simulation Output
//  $RCSfile: SignalCoordinatorEvolutionObserver.C,v $
//  $Revision: 0.2 $
//  $Date: 1996/06/19 17:36:47 $
//  $State: Stab $
//  $Author: bwb $
//  U.S. Government Copyright 1995
//  All rights reserved
```

```
//  Include TRANSIMS header files.
#include "OUT/SignalCoordinatorEvolutionObserver.h"


//  Construct a signal coordinator evolution observer.
TOutSignalCoordinatorEvolutionObserver::TOutSignalCoordinatorEvolutionObserver
        (OutObserverId id)
  : TOutObserver(id)
{
}


//  Destruct a signal coordinator evolution observer.
TOutSignalCoordinatorEvolutionObserver::~TOutSignalCoordinatorEvolutionObserver()
{
}
```

T. TOutSignalizedControlObserver Class

1. SignalizedControlObserver.h

```
//  Project: TRANSIMS
//  Subsystem: Simulation Output
//  $RCSfile: SignalizedControlObserver.h,v $
//  $Revision: 0.2 $
//  $Date: 1996/06/19 17:38:02 $
//  $State: Stab $
//  $Author: bwb $
//  U.S. Government Copyright 1995
//  All rights reserved


#ifndef TRANSIMS_OUT_SIGOBS
#define TRANSIMS_OUT_SIGOBS


//  Include TRANSIMS header files.
#include "OUT/Observer.h"
#include "NET/Id.h"
#include "NET/TrafficControl.h"


//  A signalized control observer observes data related to signals.
class TOutSignalizedControlObserver
  : public TOutObserver
{
  public:

    //  Construct a signalized control observer.
    TOutSignalizedControlObserver(OutObserverId id);

    //  Destruct a signalized control observer.
    virtual ~TOutSignalizedControlObserver();

  protected:

  //  Define values for the fields being observed.
  //  The Observe method will use these methods.

    //  Define the id of the link entering the intersection.
    void SetLink(NetLinkId link);

    //  Define the lane number of the lane entering the intersection.
    void SetLane(NetLaneNumber lane);

    //  Define the id of the node associated with the intersection.
    void SetNode(NetNodeId node);

    //  Define the current signal state.
    void SetSignal(TNetTrafficControl::ETrafficControl control);
};
```

```
#endif // TRANSIMS_OUT_SIGOBS
```

2. SignalizedControlObserver.C

```
//  Project: TRANSIMS
//  Subsystem: Simulation Output
//  $RCSfile: SignalizedControlObserver.C,v $
//  $Revision: 0.2 $
//  $Date: 1996/06/19 17:37:46 $
//  $State: Stab $
//  $Author: bwb $
//  U.S. Government Copyright 1995
//  All rights reserved


//  Include TRANSIMS header files.
#include "OUT/SignalizedControlObserver.h"
#include "OUT/Record.h"
#include "OUT/Names.h"


//  Construct a signalized control observer.
TOutSignalizedControlObserver::TOutSignalizedControlObserver(OutObserverId id)
  : TOutObserver(id)
{
    SetTime(-99.);
    SetLink(-99);
    SetLane(-99);
    SetNode(-99);
    SetSignal(TNetTrafficControl::kNone);
}


//  Destruct a signalized control observer.
TOutSignalizedControlObserver::~TOutSignalizedControlObserver()
{
}


//  Define the id of the link entering the intersection.
void TOutSignalizedControlObserver::SetLink(NetLinkId id)
{
    GetRecord().SetField(kLinkField, id);
}


//  Define the lane number of the lane entering the intersection.
void TOutSignalizedControlObserver::SetLane(NetLaneNumber n)
{
    GetRecord().SetField(kLaneField, n);
}


//  Define the id of the node associated with the intersection.
void TOutSignalizedControlObserver::SetNode(NetNodeId id)
{
    GetRecord().SetField(kNodeField, id);
}


//  Define the current signal state.
void TOutSignalizedControlObserver::SetSignal(TNetTrafficControl::ETrafficControl
        c)
{
    GetRecord().SetField(kSignalField, c);
}
```

U. TOutSpecificationReader Class

1. SpecificationReader.h

```
//  Project: TRANSIMS
//  Subsystem: Simulation Output
```

```
//  $RCSfile: SpecificationReader.h,v $
//  $Revision: 0.3 $
//  $Date: 1996/06/19 17:38:30 $
//  $State: Stab $
//  $Author: bwb $
//  U.S. Government Copyright 1995
//  All rights reserved


#ifndef TRANSIMS_OUT_SPECIFICATIONREADER
#define TRANSIMS_OUT_SPECIFICATIONREADER


//  Include TRANSIMS header files.
#include <GBL/Globals.h>
#include <DBS/Table.h>


//  A specification reader reads an output specification from the database.
class TOutSpecificationReader
{
  public:

    //  Construct a reader for the specified tables.
    TOutSpecificationReader(TDbTable generalTable, TDbTable nodeTable, TDbTable
            linkTable);

    //  Return the general specification table.
    TDbTable& GetGeneralTable();

    //  Return the node specification table.
    TDbTable& GetNodeTable();

    //  Return the link specification table.
    TDbTable& GetLinkTable();

  private:

    //  Each reader has a general specification table.
    TDbTable fGeneralTable;

    //  Each reader has a node specification table.
    TDbTable fNodeTable;

    //  Each reader has a link specification table.
    TDbTable fLinkTable;
};


#endif // TRANSIMS_OUT_SPECIFICATIONREADER
```

2. SpecificationReader.C

```
//  Project: TRANSIMS
//  Subsystem: Simulation Output
//  $RCSfile: SpecificationReader.C,v $
//  $Revision: 0.3 $
//  $Date: 1996/06/19 17:38:22 $
//  $State: Stab $
//  $Author: bwb $
//  U.S. Government Copyright 1995
//  All rights reserved


//  Include TRANSIMS header files.
#include <OUT/SpecificationReader.h>


//  Construct a reader for the specified tables.
TOutSpecificationReader::TOutSpecificationReader(TDbTable generalTable, TDbTable
        nodeTable, TDbTable linkTable)
  : fGeneralTable(generalTable),
    fNodeTable(nodeTable),
    fLinkTable(linkTable)
{
```

```
}


//  Return the general specification table.
TDbTable& TOutSpecificationReader::GetGeneralTable()
{
    return fGeneralTable;
}


//  Return the node specification table.
TDbTable& TOutSpecificationReader::GetNodeTable()
{
    return fNodeTable;
}


//  Return the link specification table.
TDbTable& TOutSpecificationReader::GetLinkTable()
{
    return fLinkTable;
}
```

V. TOutStorage Class

1. Storage.h

```
//  Project: TRANSIMS
//  Subsystem: Simulation Output
//  $RCSfile: Storage.h,v $
//  $Revision: 0.4 $
//  $Date: 1996/06/19 17:38:55 $
//  $State: Stab $
//  $Author: bwb $
//  U.S. Government Copyright 1995
//  All rights reserved


#ifndef TRANSIMS_OUT_STORAGE
#define TRANSIMS_OUT_STORAGE


//  Include Booch Components header files.
#include <BCStoreM.h>
#include <BCMapU.h>
#include <BCSetU.h>


//  Include DBtools.h++ header files.
#include <rw/rwfile.h>


//  Include TRANSIMS header files.
#include <GBL/Globals.h>
#include <OUT/Exception.h>
#include <OUT/Record.h>


//  An output storage manages the distributed file system and isolates the rest
//  of the simulation output objects from the details of the physical storage.
//  Member functions throw the exception TOutStorageFailure when errors occur.
class TOutStorage
{
  public:

    //  Type definitions.
    typedef BC_TUnboundedSet<const string, 30U, BC_CManaged> HostSet;
    typedef BC_TSetActiveIterator<const string> HostSetIterator;
    typedef RWFile* HostHandle;

    //  Constants for seek positions.
    static const long kBegin;
    static const long kEnd;
```

```
  //  Storage modes:  Use read mode for opening existing files, write mode
  //  for creating a new file, and delete mode for deleting existing files.
  enum Mode {kRead, kWrite, kDelete};

  //  Connect the storage with the given root and basic name to the specified
  //  hosts.
  TOutStorage(const string& root, const string& name, Mode mode = kRead);
  TOutStorage(const HostSet& hosts, const string& root, const string& name,
          Mode mode = kRead);

  //  Disconnect the storage.
  ~TOutStorage();

  //  Return the root name.
  const string& GetRoot() const;

  //  Return the basic name.
  const string& GetName() const;

  //  Return the host names.
  HostSet GetHosts() const;

  //  Return the host handle.
  HostHandle GetHostHandle(const string& host) const;

  //  Return the current offset for the specified host file.
  long GetOffset() const;
  long GetOffset(const string& host) const;
  long GetOffset(HostHandle host) const;

  //  Return whether an end-of-file has occured for the specified host file.
  bool AtEnd() const;
  bool AtEnd(const string& host) const;
  bool AtEnd(HostHandle host) const;

  //  Position the specified host file to the given location.
  void Seek(long position = kBegin);
  void Seek(const string& host, long position = kBegin);
  void Seek(HostHandle host, long position = kBegin);

  //  Write the given record on the specified host.
  void Write(const TOutRecord& record);
  void Write(const string& host, const TOutRecord& record);
  void Write(HostHandle host, const TOutRecord& record);

  //  Read the given record on the specified host.  Return whether a record
  //  was available for reading.
  bool Read(TOutRecord& record);
  bool Read(const string& host, TOutRecord& record);
  bool Read(HostHandle host, TOutRecord& record);

  //  Write the given record header on the specified host.
  void WriteHeader(const TOutRecord& record);
  void WriteHeader(const string& host, const TOutRecord& record);
  void WriteHeader(HostHandle host, const TOutRecord& record);

  //  Read the given record header on the specified host.  Return whether a
  //  record was available for reading.
  bool ReadHeader(TOutRecord& record);
  bool ReadHeader(const string& host, TOutRecord& record);
  bool ReadHeader(HostHandle host, TOutRecord& record);

  //  Flush any pending operations.
  void Flush();

private:

  //  Type definitions.
  typedef BC_TUnboundedMap<const string, RWFile*, 30U, BC_CManaged> FileMap;
  typedef BC_TMapActiveIterator<const string, RWFile*> FileMapIterator;

  //  Local host name.
  static const string fgLocal;

  //  File suffix.
  static const string fgSuffix;
```

```
    //  Do not allow storages to be copied.
    TOutStorage(const TOutStorage&) {}

    //  Do not allow storages to be assigned.
    TOutStorage& operator=(const TOutStorage&) {return *this;}

    //  A storage has a root location.
    const string fRoot;

    //  A storage has a basic name.
    const string fName;

    //  A storage has a file on each host.
    FileMap fFiles;
};


#endif // TRANSIMS_OUT_STORAGE
```

2. Storage.C

```
//  Project: TRANSIMS
//  Subsystem: Simulation Output
//  $RCSfile: Storage.C,v $
//  $Revision: 0.5 $
//  $Date: 1996/06/19 17:38:48 $
//  $State: Stab $
//  $Author: bwb $
//  U.S. Government Copyright 1995
//  All rights reserved


//  Include Booch Component header files.
#include <BCStoreM.h>
#include <BCStacU.h>


//  Include TRANSIMS header files.
#include <OUT/Storage.h>


//  Define the hash function for strings.
static BC_Index StringHashValue(const string& s)
{
    return HashValue(s);
}


//  Constants for seek positions.
const long TOutStorage::kBegin = 0;
const long TOutStorage::kEnd = -1;


//  Local host name.
const string TOutStorage::fgLocal = "local";


//  File suffix.
const string TOutStorage::fgSuffix = ".stg";


//  Connect the storage with the given root and basic name to the specified
//  hosts.
TOutStorage::TOutStorage(const string& root, const string& name, Mode mode)
  : fRoot(root),
    fName(name),
    fFiles(StringHashValue)
{
    string modeString;
    switch (mode) {
        case kRead:
            modeString = "rb";
            break;
        case kWrite:
```

```
            modeString = "wb+";
            break;
        case kDelete:
            throw TOutStorageFailure("Deletion not supported.");
    }

    RWFile* const file = new RWFile(fRoot + "/" + fgLocal + "/" + fName +
            fgSuffix, modeString);
    if (!file->isValid())
        throw TOutStorageFailure("Cannot open file.");
    fFiles.Bind(fgLocal, file);
}

TOutStorage::TOutStorage(const HostSet& hosts, const string& root, const string&
        name, Mode mode)
  : fRoot(root),
    fName(name),
    fFiles(StringHashValue)
{
    string modeString;
    switch (mode) {
        case kRead:
            modeString = "rb";
            break;
        case kWrite:
            modeString = "wb+";
            break;
        case kDelete:
            throw TOutStorageFailure("Deletion not supported.");
    }

    for (HostSetIterator i(hosts); !i.IsDone(); i.Next()) {
        const string& host = *i.CurrentItem();
        RWFile* const file = new RWFile(fRoot + "/" + host + "/" + fName +
                fgSuffix, modeString);
        if (!file->isValid())
            throw TOutStorageFailure("Cannot open file.");
        fFiles.Bind(host, file);
    }
}


//  Disconnect the storage.
TOutStorage::~TOutStorage()
{
    for (FileMapIterator i(fFiles); !i.IsDone(); i.Next())
        delete *i.CurrentValue();
}


//  Return the root name.
const string& TOutStorage::GetRoot() const
{
    return fRoot;
}


//  Return the basic name.
const string& TOutStorage::GetName() const
{
    return fName;
}


//  Return the host names.
TOutStorage::HostSet TOutStorage::GetHosts() const
{
    HostSet hosts(StringHashValue);

    for (FileMapIterator i(fFiles); !i.IsDone(); i.Next())
        hosts.Add(*i.CurrentItem());

    return hosts;
}
```

```
//  Return the host handle.
TOutStorage::HostHandle TOutStorage::GetHostHandle(const string& host) const
{
    RWFile* const* file = fFiles.ValueOf(host);
    if (file != NULL)
        return *file;
    else
        throw TOutStorageFailure("Invalid host.");
}


//  Return the current offset for the specified host file.
long TOutStorage::GetOffset() const
{
    return GetOffset(fgLocal);
}

long TOutStorage::GetOffset(const string& host) const
{
    RWFile* const* file = fFiles.ValueOf(host);
    if (file != NULL)
        return (*file)->CurOffset();
    else
        throw TOutStorageFailure("Invalid host.");
}


//  Return whether an end-of-file has occured for the specified host file.
bool TOutStorage::AtEnd() const
{
    return AtEnd(fgLocal);
}

bool TOutStorage::AtEnd(const string& host) const
{
    RWFile* const* file = fFiles.ValueOf(host);
    if (file != NULL)
        return AtEnd(*file);
    else
        throw TOutStorageFailure("Invalid host.");
}

bool TOutStorage::AtEnd(HostHandle host) const
{
    return host->Eof();
}


//  Position the specified host file to the given location.
void TOutStorage::Seek(long position)
{
    Seek(fgLocal, position);
}

void TOutStorage::Seek(const string& host, long position)
{
    RWFile* const* file = fFiles.ValueOf(host);
    if (file != NULL)
        Seek(*file, position);
    else
        throw TOutStorageFailure("Invalid host.");
}

void TOutStorage::Seek(HostHandle host, long position)
{
    bool okay;
    if (position == kBegin)
        okay = host->SeekToBegin();
    else if (position == kEnd)
        okay = host->SeekToEnd();
    else
        okay = host->SeekTo(position);
    if (!okay)
        throw TOutStorageFailure("Seek failed.");
}
```

```
//  Write the given record on the specified host.
void TOutStorage::Write(const TOutRecord& record)
{
    Write(fgLocal, record);
}

void TOutStorage::Write(const string& host, const TOutRecord& record)
{
    RWFile* const* file = fFiles.ValueOf(host);
    if (file != NULL)
        Write(*file, record);
    else
        throw TOutStorageFailure("Invalid host.");
}

void TOutStorage::Write(HostHandle host, const TOutRecord& record)
{
    bool fail = FALSE;
    for (TOutRecord::FieldMapIterator i = record.GetIterator(); !i.IsDone();
            i.Next()) {
        (*i.CurrentValue()).saveGuts(*host);
        fail |= host->Error();
    }
    if (fail)
        throw TOutStorageFailure("Write failure.");
}


//  Read the given record on the specified host.  Return whether a record was
//  available for reading.
bool TOutStorage::Read(TOutRecord& record)
{
    return Read(fgLocal, record);
}

bool TOutStorage::Read(const string& host, TOutRecord& record)
{
    RWFile* const* file = fFiles.ValueOf(host);
    if (file != NULL)
        return Read(*file, record);
    else
        throw TOutStorageFailure("Invalid host.");
}

bool TOutStorage::Read(HostHandle host, TOutRecord& record)
{
    bool fail = FALSE;
    for (TOutRecord::FieldMapIterator i = record.GetIterator(); !i.IsDone();
            i.Next()) {
        (*i.CurrentValue()).restoreGuts(*host);
        fail |= host->Error();
    }
    if (fail)
        throw TOutStorageFailure("Read failure.");
    else
        return !host->Eof();
}


//  Write the given record header on the specified host.
void TOutStorage::WriteHeader(const TOutRecord& record)
{
    WriteHeader(fgLocal, record);
}

void TOutStorage::WriteHeader(const string& host, const TOutRecord& record)
{
    RWFile* const* file = fFiles.ValueOf(host);
    if (file != NULL)
        WriteHeader(*file, record);
    else
        throw TOutStorageFailure("Invalid host.");
}

void TOutStorage::WriteHeader(HostHandle host, const TOutRecord& record)
```

```
{
    RWDBValue((size_t) record.GetMap().Extent()).saveGuts(*host);
    bool fail = host->Error();
    for (TOutRecord::FieldMapIterator i = record.GetIterator(); !i.IsDone();
            i.Next()) {
        RWDBValue(*i.CurrentItem()).saveGuts(*host);
        fail |= host->Error();
    }
    if (fail)
        throw TOutStorageFailure("Write failure.");
}


//  Read the given record header on the specified host.  Return whether a
//  record was available for reading.
bool TOutStorage::ReadHeader(TOutRecord& record)
{
    return ReadHeader(fgLocal, record);
}

bool TOutStorage::ReadHeader(const string& host, TOutRecord& record)
{
    RWFile* const* file = fFiles.ValueOf(host);
    if (file != NULL)
        return ReadHeader(*file, record);
    else
        throw TOutStorageFailure("Invalid host.");
}

bool TOutStorage::ReadHeader(HostHandle host, TOutRecord& record)
{
    //ISSUE(bwb): This relies on the details of the Booch map implementation.
    BC_TUnboundedStack<string, BC_CManaged> fields;
    TOutRecord::FieldMap& map = record.GetMap();
    RWDBValue temp;
    temp.restoreGuts(*host);
    bool fail = host->Error();
    const size_t n = temp.asUnsignedInt();
    for (size_t i = 0; i < n; ++i) {
        temp.restoreGuts(*host);
        fail |= host->Error();
        fields.Push(temp.asString());
    }
    map.Clear();
    for (; !fields.IsEmpty(); fields.Pop())
        map.Bind(fields.Top(), RWDBValue());
    if (fail)
        throw TOutStorageFailure("Read failure.");
    else
        return !host->Eof();
}


//  Flush any pending operations.
void TOutStorage::Flush()
{
    bool fail = FALSE;
    for (FileMapIterator i(fFiles); !i.IsDone(); i.Next())
        fail |= (*i.CurrentValue())->Flush();
    if (fail)
        throw TOutStorageFailure("Flush failure.");
}
```

W. TOutSummaryProcessor Class

1. SummaryProcessor.h

```
//  Project: TRANSIMS
//  Subsystem: Simulation Output
//  $RCSfile: SummaryProcessor.h,v $
//  $Revision: 0.1 $
//  $Date: 1996/06/19 17:39:48 $
//  $State: Stab $
//  $Author: bwb $
```

```
//  U.S. Government Copyright 1996
//  All rights reserved


#ifndef TRANSIMS_OUT_SUMMARYPROCESSOR
#define TRANSIMS_OUT_SUMMARYPROCESSOR


//  Include Booch Components header files.
#include <BCStoreM.h>
#include <BCSetU.h>


//  Include TRANSIMS header files.
#include "OUT/Processor.h"
#include "OUT/Storage.h"


//  Forward declarations.
class TOutObserver;


//  An output summary processor collects statistics on the
//  simulation.
class TOutSummaryProcessor
  : public TOutProcessor
{
  public:

    //  Type definitions.
    typedef BC_TUnboundedSet<TOutObserver*, 10000U, BC_CManaged> ObserverSet;
    typedef BC_TSetActiveIterator<TOutObserver*> ObserverSetIterator;

    //  Construct a summary processor.
    TOutSummaryProcessor(OutProcessorId id, const TOutGeneralSpecification&
            specification);

    //  Destroy a summary processor.
    virtual ~TOutSummaryProcessor();

    //  Return processor type.
    EProcessorType GetProcessorType() const;

    //  Begin recording output for this time step.
    virtual void RecordOutput(REAL time);

    //  Finish recording output for link space.
    virtual void RecordSpace(const TOutObserver& observer);

    //  Finish recording output for link time.
    virtual void RecordTime(const TOutObserver& observer);

    //  Return the space observers.
    ObserverSet& GetSpaceObservers();
    const ObserverSet& GetSpaceObservers() const;

    //  Return the time observers.
    ObserverSet& GetTimeObservers();
    const ObserverSet& GetTimeObservers() const;

    //  Define a space observer.
    void AddSpaceObserver(TOutObserver& observer);

    //  Define a time observer.
    void AddTimeObserver(TOutObserver& observer);

    //  Undefine a space observer.
    void RemoveSpaceObserver(TOutObserver& observer);

    //  Undefine a time observer.
    void RemoveTimeObserver(TOutObserver& observer);

  private:

    //  Do not allow summary processors to be copied.
    //TOutSummaryProcessor(const TOutSummaryProcessor&) {}
```

```
    //  Do not allow summary processors to be assigned.
    //TOutSummaryProcessor& operator=(const TOutSummaryProcessor&) {return
    //      *this;}

    //  A summary processor has space observers.
    ObserverSet fSpaceObservers;

    //  A summary processor has time observers;
    ObserverSet fTimeObservers;

    //  An summary processor is connected to a space storage.
    TOutStorage fSpaceStorage;

    //  A summary processor is connected to a time storage.
    TOutStorage fTimeStorage;
};


#endif // TRANSIMS_OUT_SUMMARYPROCESSOR
```

2. SummaryProcessor.C

```
//  Project: TRANSIMS
//  Subsystem: Simulation Output
//  $RCSfile: SummaryProcessor.C,v $
//  $Revision: 0.1 $
//  $Date: 1996/06/19 17:39:17 $
//  $State: Stab $
//  $Author: bwb $
//  U.S. Government Copyright 1996
//  All rights reserved


//  Include TRANSIMS header files.
#include "OUT/SummaryProcessor.h"
#include "OUT/Exception.h"
#include "OUT/Observer.h"
#include "OUT/Names.h"


//  Define the hash function for observer sets.
static BC_Index ObserverHashValue(TOutObserver* const& observer)
{
    return BC_Index(observer->GetId());
}


//  Construct a summary processor.
TOutSummaryProcessor::TOutSummaryProcessor(OutProcessorId id, const
        TOutGeneralSpecification& specification)
  : TOutProcessor(id, specification),
    fSpaceObservers(ObserverHashValue),
    fSpaceStorage(specification.GetRoot(), specification.GetName() + "." +
            kSpaceSuffix, TOutStorage::kWrite),
    fTimeObservers(ObserverHashValue),
    fTimeStorage(specification.GetRoot(), specification.GetName() + "." +
            kTimeSuffix, TOutStorage::kWrite)
{
}


//  Destroy a summary processor.
TOutSummaryProcessor::~TOutSummaryProcessor()
{
}


//  Return the processor type.
TOutProcessor::EProcessorType TOutSummaryProcessor::GetProcessorType() const
{
    return kSummaryProcessor;
}
```

```
//  Begin recording output for this time step.
void TOutSummaryProcessor::RecordOutput(REAL)
{
    throw TOutException("Client must override"
            " TOutSummaryProcessor::RecordOutput(time).");
}


//  Finish recording output for link space.
void TOutSummaryProcessor::RecordSpace(const TOutObserver& observer)
{
    if (fSpaceStorage.GetOffset() == 0)
        fSpaceStorage.WriteHeader(observer.GetRecord());
    fSpaceStorage.Write(observer.GetRecord());
}


//  Finish recording output for link time.
void TOutSummaryProcessor::RecordTime(const TOutObserver& observer)
{
    if (fTimeStorage.GetOffset() == 0)
        fTimeStorage.WriteHeader(observer.GetRecord());
    fTimeStorage.Write(observer.GetRecord());
}


//  Return the space observers.
TOutSummaryProcessor::ObserverSet& TOutSummaryProcessor::GetSpaceObservers()
{
    return fSpaceObservers;
}

const TOutSummaryProcessor::ObserverSet&
        TOutSummaryProcessor::GetSpaceObservers() const
{
    return fSpaceObservers;
}


//  Return the time observers.
TOutSummaryProcessor::ObserverSet& TOutSummaryProcessor::GetTimeObservers()
{
    return fTimeObservers;
}

const TOutSummaryProcessor::ObserverSet&
        TOutSummaryProcessor::GetTimeObservers() const
{
    return fTimeObservers;
}


//  Define a space observer.
void TOutSummaryProcessor::AddSpaceObserver(TOutObserver& observer)
{
    fSpaceObservers.Add(&observer);
}


//  Define a time observer.
void TOutSummaryProcessor::AddTimeObserver(TOutObserver& observer)
{
    fTimeObservers.Add(&observer);
}


//  Undefine a space observer.
void TOutSummaryProcessor::RemoveSpaceObserver(TOutObserver& observer)
{
    fSpaceObservers.Remove(&observer);
}


//  Undefine a time observer.
void TOutSummaryProcessor::RemoveTimeObserver(TOutObserver& observer)
```

```
{
    fTimeObservers.Remove(&observer);
}
```

X. TOutSummaryRetriever Class

1. SummaryRetriever.h

```
//  Project: TRANSIMS
//  Subsystem: Simulation Output
//  $RCSfile: SummaryRetriever.h,v $
//  $Revision: 0.1 $
//  $Date: 1996/06/19 17:41:04 $
//  $State: Stab $
//  $Author: bwb $
//  U.S. Government Copyright 1995
//  All rights reserved


#ifndef TRANSIMS_OUT_SUMMARYRETRIEVER
#define TRANSIMS_OUT_SUMMARYRETRIEVER


//  Include TRANSIMS header files.
#include <GBL/Globals.h>
#include <OUT/Retriever.h>


//  An event retriever gets specific summary data from storage and
//  coordinates its conversion and filtering.
class TOutSummaryRetriever
  : public TOutRetriever
{
  public:

    //  Construct a reader for the specified hosts and given specification,
    //  and network.
    TOutSummaryRetriever(const TOutStorage::HostSet& hosts, const
            TOutGeneralSpecification& specification, const TNetNetwork* network
            = NULL);

    //  Perform the retrieval on the specified writers.
    void Retrieve(TOutWriter& spaceWriter, TOutWriter& timeWriter, bool sort =
        TRUE);

  private:

    //  The retriever is connected to a space storage.
    TOutStorage fSpaceStorage;

    //  The retriever is connected to a time storage.
    TOutStorage fTimeStorage;
};


#endif // TRANSIMS_OUT_SUMMARYRETRIEVER
```

2. SummaryRetriever.C

```
//  Project: TRANSIMS
//  Subsystem: Simulation Output
//  $RCSfile: SummaryRetriever.C,v $
//  $Revision: 0.1 $
//  $Date: 1996/06/19 17:40:36 $
//  $State: Stab $
//  $Author: bwb $
//  U.S. Government Copyright 1996
//  All rights reserved


//  Include TRANSIMS header files.
#include <OUT/SummaryRetriever.h>
#include <OUT/Names.h>
```

```
//  Construct a reader for the specified hosts and given specification.
TOutSummaryRetriever::TOutSummaryRetriever(const TOutStorage::HostSet&
        hosts, const TOutGeneralSpecification& specification, const TNetNetwork*
        network)
  : TOutRetriever(specification, network),
    fSpaceStorage(hosts, specification.GetRoot(), specification.GetName() +
            "." + kSpaceSuffix),
    fTimeStorage(hosts, specification.GetRoot(), specification.GetName() +
            "." + kTimeSuffix)
{
}


//  Perform the retrieval on the specified writers.
void TOutSummaryRetriever::Retrieve(TOutWriter& spaceWriter, TOutWriter&
        timeWriter, bool sort)
{
    BasicRetrieve(fSpaceStorage, spaceWriter, sort);
    BasicRetrieve(fTimeStorage, timeWriter, sort);
}
```

Y. TOutTextWriter Class

1. TextWriter.h

```
//  Project: TRANSIMS
//  Subsystem: Simulation Output
//  $RCSfile: TextWriter.h,v $
//  $Revision: 0.5 $
//  $Date: 1996/06/19 17:42:54 $
//  $State: Stab $
//  $Author: bwb $
//  U.S. Government Copyright 1995
//  All rights reserved


#ifndef TRANSIMS_OUT_TEXTWRITER
#define TRANSIMS_OUT_TEXTWRITER


//  Include Standard C++ header files.
#include <fstream.h>


//  Include Booch Components header files.
#include <BCStoreM.h>
#include <BCCollU.h>


//  Include TRANSIMS header files.
#include <GBL/Globals.h>
#include <OUT/Writer.h>


//  A text writer puts simulation output data into a formatted text file.  The
//  exception TOutWriterFailure is thrown if an operation fails.
class TOutTextWriter
  : public TOutWriter
{
  public:

    //  Open the specified file for writing, including the record header
    TOutTextWriter(const string& file, const string& delimiter = "\t", bool
            includeHeader = FALSE);

    //  Write the specified record with fields in the given order.
    void Write(const TOutRecord& record);
    void Write(const TOutRecord& record, const FieldCollection& fields);

  private:

    //  Each text writer is connected to a file.
```

```
    ofstream fOut;

    //  Each text writer has a delimiter.
    string fDelimiter;

    //  A text writer may have to include header information.
    bool fIncludeHeader;
};


#endif // TRANSIMS_OUT_TEXTWRITER
```

2. TextWriter.C

```
//  Project: TRANSIMS
//  Subsystem: Simulation Output
//  $RCSfile: TextWriter.C,v $
//  $Revision: 0.6 $
//  $Date: 1996/06/19 17:42:43 $
//  $State: Stab $
//  $Author: bwb $
//  U.S. Government Copyright 1995
//  All rights reserved


//  Include TRANSIMS header files.
#include <OUT/TextWriter.h>


//  Open the specified file for writing, including the record header
TOutTextWriter::TOutTextWriter(const string& file, const string& delimiter,
        bool includeHeader)
  : fOut(file),
    fDelimiter(delimiter),
    fIncludeHeader(includeHeader)
{
    if (!fOut.rdbuf()->is_open())
        throw TOutWriterFailure("Cannot open file.");
}


//  Write the specified record.
void TOutTextWriter::Write(const TOutRecord& record)
{
    FieldCollection fields;
    for (TOutRecord::FieldMapIterator i = record.GetIterator(); !i.IsDone();
            i.Next())
        fields.Append(*i.CurrentItem());
    Write(record, fields);
}

void TOutTextWriter::Write(const TOutRecord& record, const FieldCollection&
        fields)
{
    const TOutRecord::FieldMap& map = record.GetMap();

    if (fIncludeHeader) {
        fIncludeHeader = FALSE;
        bool first = TRUE;
        for (FieldCollectionIterator i(fields); !i.IsDone(); i.Next()) {
            if (first) {
                first = FALSE;
                fOut << *i.CurrentItem();
            } else
                fOut << fDelimiter << *i.CurrentItem();
        }
        fOut << endl;
    }

    bool first = TRUE;
    for (FieldCollectionIterator i(fields); !i.IsDone(); i.Next()) {
        if (first)
            first = FALSE;
        else
            fOut << fDelimiter;
```

```
        const RWDBValue& value = *map.ValueOf(*i.CurrentItem());
        switch (value.type()) {
            case TOutRecord::kChar:
            case TOutRecord::kUnsignedChar:
                fOut << value.asInt();
                break;
            default:
                fOut << value.asString();
                break;
        }
    }
    fOut << endl;

    if (fOut.fail())
        throw TOutWriterFailure("Cannot write file.");
}
```

Z. TOutVehicleObserver Class

1. VehicleObserver.h

```
//  Project: TRANSIMS
//  Subsystem: Simulation Output
//  $RCSfile: VehicleObserver.h,v $
//  $Revision: 0.4 $
//  $Date: 1996/06/19 17:43:24 $
//  $State: Stab $
//  $Author: bwb $
//  U.S. Government Copyright 1995
//  All rights reserved


#ifndef TRANSIMS_OUT_VEHOBS
#define TRANSIMS_OUT_VEHOBS

//  Include TRANSIMS header files.
#include "OUT/Observer.h"
#include "NET/Id.h"


//  A vehicle observer observes data related to vehicles.
class TOutVehicleObserver
  : public TOutObserver
{
  public:

    //  Construct a vehicle observer.
    TOutVehicleObserver(OutObserverId id);

    //  Destruct a vehicle observer.
    virtual ~TOutVehicleObserver();

  protected:

  //  Define values for the fields being observed.
  //  The Observe method will use these methods.

    //  Define the vehicle’s id.
    void SetId(UINT id);

    //  Define the id of the link the vehicle is on.
    void SetLink(NetLinkId link);

    //  Define the lane number the vehicle is on.
    void SetLane(NetLaneNumber lane);

    //  Define the vehicle’s distance from a node.
    void SetDistance(REAL distance);

    //  Define the id of the node the distance is measured from.
    void SetNode(NetNodeId node);

    //  Define the vehicle’s velocity.
    void SetVelocity(REAL velocity);
```

```
    //  Define the vehicle's status.
    void SetStatus(BYTE status);
};


#endif // TRANSIMS_OUT_VEHOBS
```

2. VehicleObserver.C

```
//  Project: TRANSIMS
//  Subsystem: Simulation Output
//  $RCSfile: VehicleObserver.C,v $
//  $Revision: 0.5 $
//  $Date: 1996/06/19 17:43:08 $
//  $State: Stab $
//  $Author: bwb $
//  U.S. Government Copyright 1995
//  All rights reserved


//  Include TRANSIMS header files.
#include "OUT/VehicleObserver.h"
#include "OUT/Record.h"
#include "OUT/Names.h"


//  Construct a vehicle observer.
TOutVehicleObserver::TOutVehicleObserver(OutObserverId id)
  : TOutObserver(id)
{
    SetTime(-99.);
    SetId(-99);
    SetLink(-99);
    SetLane(-99);
    SetNode(-99);
    SetDistance(-99.0);
    SetVelocity(-99.0);
    SetStatus(255);
}


//  Destruct a vehicle observer.
TOutVehicleObserver::~TOutVehicleObserver()
{
}


//  Define the vehicle's id.
void TOutVehicleObserver::SetId(UINT id)
{
    GetRecord().SetField(kVehicleField, id);
}


//  Define the id of the link the vehicle is on.
void TOutVehicleObserver::SetLink(NetLinkId id)
{
    GetRecord().SetField(kLinkField, id);
}


//  Define the lane number the vehicle is on.
void TOutVehicleObserver::SetLane(NetLaneNumber n)
{
    GetRecord().SetField(kLaneField, n);
}


//  Define the vehicle's distance from a node.
void TOutVehicleObserver::SetDistance(REAL d)
{
    GetRecord().SetField(kDistanceField, d);
}
```

```
//  Define the id of the node the distance is measured from.
void TOutVehicleObserver::SetNode(NetNodeId id)
{
    GetRecord().SetField(kNodeField, id);
}


//  Define the vehicle's velocity.
void TOutVehicleObserver::SetVelocity(REAL v)
{
    GetRecord().SetField(kVelocityField, v);
}


//  Define the vehicle's status flag.
void TOutVehicleObserver::SetStatus(BYTE s)
{
    GetRecord().SetField(kStatusField, s);
}
```

AA. TOutWriter Class

1. Writer.h

```
//  Project: TRANSIMS
//  Subsystem: Simulation Output
//  $RCSfile: Writer.h,v $
//  $Revision: 0.4 $
//  $Date: 1996/06/19 17:43:56 $
//  $State: Stab $
//  $Author: bwb $
//  U.S. Government Copyright 1995
//  All rights reserved


#ifndef TRANSIMS_OUT_WRITER
#define TRANSIMS_OUT_WRITER


//  Include TRANSIMS header files.
#include <GBL/Globals.h>
#include <OUT/Exception.h>
#include <OUT/Record.h>


//  An output writer provides an interface for the external writing of data
//  from simulation output.  The exception TOutWriterFailure is thrown if an
//  operation fails.
class TOutWriter
{
  public:

    //  Type definitions.
    typedef BC_TUnboundedCollection<string, BC_CManaged> FieldCollection;
    typedef BC_TCollectionActiveIterator<string> FieldCollectionIterator;

    //  Destroy the writer.
    virtual ~TOutWriter() {}

    //  Write the specified record.
    virtual void Write(const TOutRecord& record) = 0;
    virtual void Write(const TOutRecord& record, const FieldCollection& fields)
            = 0;

  protected:

    //  Construct a writer.
    TOutWriter() {}

  private:

    //  Do not allow writers to be copied.
    TOutWriter(const TOutWriter&) {}
```

```
    //  Do not allow writers to be assigned.
    TOutWriter& operator=(const TOutWriter&) {return *this;}
};


#endif // TRANSIMS_OUT_WRITER
```

BB. Type Definitions

1. Id.h

```
//  Project: TRANSIMS
//  Subsystem: Simulation Output
//  $RCSfile: Id.h,v $
//  $Revision: 0.4 $
//  $Date: 1996/06/19 17:26:19 $
//  $State: Stab $
//  $Author: bwb $
//  U.S. Government Copyright 1995
//  All rights reserved


#ifndef TRANSIMS_OUT_ID
#define TRANSIMS_OUT_ID


//  Include TRANSIMS header files.
#include "GBL/Globals.h"


//  A processor id is four bytes long.
typedef DWORD OutProcessorId;

//  An observer id is four bytes long.
typedef DWORD OutObserverId;


#endif // TRANSIMS_OUT_ID
```

2. Names.h

```
//  Project: TRANSIMS
//  Subsystem: Simulation Output
//  $RCSfile: Names.h,v $
//  $Revision: 0.5 $
//  $Date: 1996/06/19 17:31:05 $
//  $State: Stab $
//  $Author: bwb $
//  U.S. Government Copyright 1995
//  All rights reserved


#ifndef TRANSIMS_OUT_NAMES
#define TRANSIMS_OUT_NAMES


//  The suffixes.
static const string kVehicleSuffix = "veh";
static const string kIntersectionSuffix = "int";
static const string kSignalSuffix = "sig";
static const string kLinkSuffix = "lnk";
static const string kSpaceSuffix = "spa";
static const string kTimeSuffix = "tim";


//  Field names.
static const string kTimeField = "TIME";
static const string kLinkField = "LINK";
static const string kVehicleField = "VEHICLE";
static const string kLaneField = "LANE";
static const string kDistanceField = "DISTANCE";
static const string kNodeField = "NODE";
```

```
static const string kVelocityField = "VELOCITY";
static const string kStatusField = "STATUS";
static const string kQIndexField = "QINDEX";
static const string kSignalField = "SIGNAL";
static const string kCountField = "COUNT";
static const string kSumField = "SUM";
static const string kSumSquaresField = "SUMSQUARES";


#endif // TRANSIMS_OUT_NAMES
```

IX. APPENDIX: Test Program

This appendix contains the complete C++ source code and the database import file for the simulation output subsystem test program.

A. Test.C

```
//  Project: TRANSIMS
//  Subsystem: Simulation Output
//  $RCSfile: Test.C,v $
//  $Revision: 0.7 $
//  $Date: 1996/06/19 17:41:35 $
//  $State: Stab $
//  $Author: bwb $
//  U.S. Government Copyright 1995
//  All rights reserved


//  Include Standard C++ header files.
#include <iostream.h>


//  Include TRANSIMS header files.
#include <GBL/Globals.h>
#include <DBS/Exception.h>
#include <OUT/Exception.h>


//  Function prototypes.
extern bool TestRecord();
extern bool TestStorage();
extern bool TestTextWriter();
extern bool TestGeneralSpecification();
extern bool TestEvolutionRetriever();


//  Main program.
int main(int, char*[])
{
    try {

        bool fail = FALSE;

        cout << "Simulation Output Subsystem Tests"
            << " [$Revision: 0.7 $]"
            << endl;

        fail |= !TestRecord();
        fail |= !TestStorage();
        fail |= !TestTextWriter();
        fail |= !TestGeneralSpecification();
        fail |= !TestEvolutionRetriever();

        cout << (fail ? "  F" : "  No f") << "ailures occured." << endl;

    } catch(const TOutException& exception) {

        cout << endl;
        cout << "Unexpected simulation output exception: " <<
                exception.GetMessage() << endl;

    } catch(const TDbException& exception) {
```

```
        cout << endl;
        cout << "Unexpected database exception: " << exception.GetMessage() <<
                endl;

    } catch(...) {

        cout << endl;
        cout << "Unexpected unknown exception occurred--aborting." << endl;

    }

    return 0;
}
```

B. TestEvolutionRetriever.C

```
//  Project: TRANSIMS
//  Subsystem: Simulation Output
//  $RCSfile: TestEvolutionRetriever.C,v $
//  $Revision: 0.5 $
//  $Date: 1996/06/19 17:41:48 $
//  $State: Stab $
//  $Author: bwb $
//  U.S. Government Copyright 1995
//  All rights reserved


//  Include Standard C++ header files.
#include <fstream.h>
#include <iostream.h>


//  Include TRANSIMS header files.
#include <DBS/Exception.h>
#include <DBS/Directory.h>
#include <DBS/Source.h>
#include <DBS/Table.h>
#include <OUT/SpecificationReader.h>
#include <OUT/GeneralSpecificationReader.h>
#include <OUT/GeneralSpecification.h>
#include <OUT/EvolutionRetriever.h>
#include <OUT/TextWriter.h>


//  Test evolution retriever class.
bool TestEvolutionRetriever()
{
    cout << "  Evolution Retriever Class Tests"
        << " [$Revision: 0.5 $]"
        << endl;

    bool anyFail = FALSE;
    bool fail;

    {
        TOutStorage::HostSet hosts(HashValue);
        hosts.Add("host1");
        hosts.Add("host2");
        TOutStorage storage(hosts, "/home/projects/transims/test/NET",
            "sample.veh", TOutStorage::kWrite);
        TOutRecord record;

        record.SetField("TIME", 10);
        record.SetField("LINK", 11);
        record.SetField("VELOCITY", 5);
        storage.WriteHeader("host1", record);
        storage.Write("host1", record);

        record.SetField("TIME", 60);
        record.SetField("LINK", 11);
        record.SetField("VELOCITY", 6);
        storage.Write("host1", record);

        record.SetField("TIME", 61);
```

```
        record.SetField("LINK", 12);
        record.SetField("VELOCITY", 7);
        storage.Write("host1", record);

        record.SetField("TIME", 65);
        record.SetField("LINK", 10);
        record.SetField("VELOCITY", 8);
        storage.Write("host1", record);

        record.SetField("TIME", 9);
        record.SetField("LINK", 12);
        record.SetField("VELOCITY", 4);
        storage.WriteHeader("host2", record);
        storage.Write("host2", record);

        record.SetField("TIME", 60);
        record.SetField("LINK", 12);
        record.SetField("VELOCITY", 3);
        storage.Write("host2", record);

        record.SetField("TIME", 65);
        record.SetField("LINK", 12);
        record.SetField("VELOCITY", 2);
        storage.Write("host2", record);

        record.SetField("TIME", 65);
        record.SetField("LINK", 11);
        record.SetField("VELOCITY", 1);
        storage.Write("host2", record);

        record.SetField("TIME", 65);
        record.SetField("LINK", 10);
        record.SetField("VELOCITY", 0);
        storage.Write("host2", record);

        record.SetField("TIME", 65);
        record.SetField("LINK", 11);
        record.SetField("VELOCITY", 0);
        storage.Write("host2", record);
    }

    TDbDirectory directory(TDbDirectoryDescription("IOC-1"));

    TDbSource generalSource(directory,
            directory.GetSource("Output Specification"));
    TDbSource nodeSource(directory,
            directory.GetSource("Output Node Specification"));
    TDbSource linkSource(directory,
            directory.GetSource("Output Link Specification"));

    TDbTable generalTable(generalSource, generalSource.GetTable("Sample Output "
            "Specification Table"));
    TDbTable nodeTable(nodeSource, nodeSource.GetTable("Sample Output Node "
            "Specification Table"));
    TDbTable linkTable(linkSource, linkSource.GetTable("Sample Output Link "
            "Specification Table"));

    TOutGeneralSpecificationReader reader(TOutSpecificationReader(generalTable,
            nodeTable, linkTable));
    reader.Reset();
    TOutStorage::HostSet hosts(HashValue);
    hosts.Add("host1");
    hosts.Add("host2");
    TOutGeneralSpecification specification(reader);
    TOutEvolutionRetriever retriever(hosts, specification);

//  ISSUE (KPB) Intersection and signal information is not really being tested.
//  Sample intersection and signal storages are duplicates of the vehicle
//  storage; therefore the sample.int.txt and sample.sig.txt files produced
//  here are (correctly) empty.
    TOutTextWriter* vehicleWriter = new
            TOutTextWriter("/home/projects/transims/test/NET/sample.veh.txt",
            ", ", TRUE);
    TOutTextWriter* intersectionWriter = new
            TOutTextWriter("/home/projects/transims/test/NET/sample.int.txt",
            ", ", TRUE);
```

```
    TOutTextWriter* signalWriter = new
            TOutTextWriter("/home/projects/transims/test/NET/sample.sig.txt",
            ", ", TRUE);
    retriever.Retrieve(*vehicleWriter, *intersectionWriter, *signalWriter);
    delete vehicleWriter;
    delete intersectionWriter;
    delete signalWriter;

    cout << "    OUT-ER-010: Data retrieval.";
    ifstream in("/home/projects/transims/test/NET/sample.veh.txt");
    char line[1000];
    in.getline(line, 1000);
    fail = strcmp(line, "TIME, VELOCITY, LINK") != 0;
    in.getline(line, 1000);
    fail |= strcmp(line, "60, 6, 11") != 0;
    in.getline(line, 1000);
    fail |= strcmp(line, "60, 3, 12") != 0;
    in.getline(line, 1000);
    fail |= strcmp(line, "65, 2, 12") != 0;
    in.getline(line, 1000);
    fail |= strcmp(line, "65, 1, 11") != 0;
    in.getline(line, 1000);
    fail |= strcmp(line, "65, 0, 11") != 0;
    in.getline(line, 1000);
    fail |= !in.eof();
    cout << (fail ? " [failed]" : " [passed]") << endl;
    anyFail |= fail;

    cout << (anyFail ? "    F" : "    No f") << "ailures occurred." << endl;
    return !anyFail;
}
```

C. TestGeneralSpecification.C

```
//  Project: TRANSIMS
//  Subsystem: Simulation Output
//  $RCSfile: TestGeneralSpecification.C,v $
//  $Revision: 0.5 $
//  $Date: 1996/06/19 17:42:00 $
//  $State: Stab $
//  $Author: bwb $
//  U.S. Government Copyright 1995
//  All rights reserved


//  Include Standard C++ header files.
#include <iostream.h>


//  Include Standard C header files.
#include <math.h>


//  Include TRANSIMS header files.
#include <DBS/Exception.h>
#include <DBS/Directory.h>
#include <DBS/Source.h>
#include <DBS/Table.h>
#include <OUT/SpecificationReader.h>
#include <OUT/GeneralSpecificationReader.h>
#include <OUT/GeneralSpecification.h>


//  Return whether two real numbers are the same.
static bool Equal(REAL a, REAL b)
{
    return fabs(a - b) < 1e-5;
}


//  Test general specification class.
bool TestGeneralSpecification()
{
    cout << "  General Specification Class Tests"
        << " [$Revision: 0.5 $]"
```

```
    << endl;

bool anyFail = FALSE;
bool fail;

TDbDirectory directory(TDbDirectoryDescription("IOC-1"));

TDbSource generalSource(directory,
        directory.GetSource("Output Specification"));
TDbSource nodeSource(directory,
        directory.GetSource("Output Node Specification"));
TDbSource linkSource(directory,
        directory.GetSource("Output Link Specification"));

TDbTable generalTable(generalSource, generalSource.GetTable("Sample Output "
        "Specification Table"));
TDbTable nodeTable(nodeSource, nodeSource.GetTable("Sample Output Node "
        "Specification Table"));
TDbTable linkTable(linkSource, linkSource.GetTable("Sample Output Link "
        "Specification Table"));

TOutGeneralSpecificationReader reader(TOutSpecificationReader(generalTable,
        nodeTable, linkTable));

reader.Reset();
TOutGeneralSpecification specification(reader);

cout << "    OUT-GS-010: Root retrieval.";
fail = specification.GetRoot() != "/home/projects/transims/test/NET";
cout << (fail ? " [failed]" : " [passed]") << endl;
anyFail |= fail;

cout << "    OUT-GS-020: Name retrieval.";
fail = specification.GetName() != "sample";
cout << (fail ? " [failed]" : " [passed]") << endl;
anyFail |= fail;

cout << "    OUT-GS-030: Time collection.";
fail = specification.CollectForTime(59) || !specification.CollectForTime(60)
        || !specification.CollectForTime(100) ||
        specification.CollectForTime(101) ||
        !specification.CollectForTime(120) ||
        specification.CollectForTime(121);
cout << (fail ? " [failed]" : " [passed]") << endl;
anyFail |= fail;

cout << "    OUT-GS-040: Time sampling.";
fail = specification.SampleForTime(50) || !specification.SampleForTime(60)
        || !specification.SampleForTime(100) ||
        specification.SampleForTime(101) ||
        !specification.SampleForTime(120) ||
        specification.SampleForTime(121);
cout << (fail ? " [failed]" : " [passed]") << endl;
anyFail |= fail;

cout << "    OUT-GS-050: Box length.";
fail = !Equal(specification.GetBoxLength(), 150);
cout << (fail ? " [failed]" : " [passed]") << endl;
anyFail |= fail;

cout << "    OUT-GS-060: Point collection.";
fail = specification.CollectForPoint(TGeoPoint(0, 0)) ||
        !specification.CollectForPoint(TGeoPoint(1, 2)) ||
        !specification.CollectForPoint(TGeoPoint(500, 600)) ||
        !specification.CollectForPoint(TGeoPoint(1000, 2000)) ||
        specification.CollectForPoint(TGeoPoint(1001, 2001));
cout << (fail ? " [failed]" : " [passed]") << endl;
anyFail |= fail;

cout << "    OUT-GS-070: Node collection.";
fail = specification.CollectForNode(11) ||
        !specification.CollectForNode(1) ||
        !specification.CollectForNode(2);
cout << (fail ? " [failed]" : " [passed]") << endl;
anyFail |= fail;
```

```
    cout << "    OUT-GS-080: Link collection.";
    fail = specification.CollectForLink(1) ||
            !specification.CollectForLink(11) ||
            !specification.CollectForLink(12);
    cout << (fail ? " [failed]" : " [passed]") << endl;
    anyFail |= fail;

    cout << "    OUT-GS-090: Processor type.";
    fail = reader.GetProcessorType() !=
            TOutGeneralSpecificationReader::kEvolutionProcessor;
    cout << (fail ? " [failed]" : " [passed]") << endl;
    anyFail |= fail;

    cout << (anyFail ? "    F" : "    No f") << "ailures occurred." << endl;
    return !anyFail;
}
```

D. TestRecord.C

```
//  Project: TRANSIMS
//  Subsystem: Simulation Output
//  $RCSfile: TestRecord.C,v $
//  $Revision: 0.5 $
//  $Date: 1996/06/19 17:42:11 $
//  $State: Stab $
//  $Author: bwb $
//  U.S. Government Copyright 1995
//  All rights reserved


//  Include Standard C++ header files.
#include <iostream.h>


//  Include TRANSIMS header files.
#include <OUT/Record.h>


//  Test record class.
bool TestRecord()
{
    cout << "  Record Class Tests"
        << " [$Revision: 0.5 $]"
        << endl;

    bool anyFail = FALSE;
    bool fail;

    TOutRecord record;

    cout << "    OUT-RE-010: Setting/getting char fields.";
    {
        const char in = -1;
        char out;
        record.SetField("char", in);
        record.GetField("char", out);
        fail = in != out || record.GetType("char") != TOutRecord::kChar;
    }
    cout << (fail ? " [failed]" : " [passed]") << endl;
    anyFail |= fail;

    cout << "    OUT-RE-020: Setting/getting unsigned char fields.";
    {
        const unsigned char in = 2;
        unsigned char out;
        record.SetField("unsigned char", in);
        record.GetField("unsigned char", out);
        fail = in != out || record.GetType("unsigned char") !=
                TOutRecord::kUnsignedChar;
    }
    cout << (fail ? " [failed]" : " [passed]") << endl;
    anyFail |= fail;

    cout << "    OUT-RE-030: Setting/getting short fields.";
    {
```

```
        const short in = -300;
        short out;
        record.SetField("short", in);
        record.GetField("short", out);
        fail = in != out || record.GetType("short") != TOutRecord::kShort;
    }
    cout << (fail ? " [failed]" : " [passed]") << endl;
    anyFail |= fail;

    cout << "    OUT-RE-040: Setting/getting unsigned short fields.";
    {
        const unsigned short in = 400;
        unsigned short out;
        record.SetField("unsigned short", in);
        record.GetField("unsigned short", out);
        fail = in != out || record.GetType("unsigned short") !=
                TOutRecord::kUnsignedShort;
    }
    cout << (fail ? " [failed]" : " [passed]") << endl;
    anyFail |= fail;

    cout << "    OUT-RE-050: Setting/getting int fields.";
    {
        const int in = -50000;
        int out;
        record.SetField("int", in);
        record.GetField("int", out);
        fail = in != out || record.GetType("int") != TOutRecord::kInt;
    }
    cout << (fail ? " [failed]" : " [passed]") << endl;
    anyFail |= fail;

    cout << "    OUT-RE-060: Setting/getting unsigned int fields.";
    {
        const unsigned int in = 60000;
        unsigned int out;
        record.SetField("unsigned int", in);
        record.GetField("unsigned int", out);
        fail = in != out || record.GetType("unsigned int") !=
                TOutRecord::kUnsignedInt;
    }
    cout << (fail ? " [failed]" : " [passed]") << endl;
    anyFail |= fail;

    cout << "    OUT-RE-070: Setting/getting long fields.";
    {
        const long in = -70000;
        long out;
        record.SetField("long", in);
        record.GetField("long", out);
        fail = in != out || record.GetType("long") != TOutRecord::kLong;
    }
    cout << (fail ? " [failed]" : " [passed]") << endl;
    anyFail |= fail;

    cout << "    OUT-RE-080: Setting/getting unsigned long fields.";
    {
        const unsigned long in = 80000;
        unsigned long out;
        record.SetField("unsigned long", in);
        record.GetField("unsigned long", out);
        fail = in != out || record.GetType("unsigned long") !=
                TOutRecord::kUnsignedLong;
    }
    cout << (fail ? " [failed]" : " [passed]") << endl;
    anyFail |= fail;

    cout << "    OUT-RE-090: Setting/getting float fields.";
    {
        const float in = 0.9;
        float out;
        record.SetField("float", in);
        record.GetField("float", out);
        fail = in != out || record.GetType("float") != TOutRecord::kFloat;
    }
    cout << (fail ? " [failed]" : " [passed]") << endl;
```

```
    anyFail |= fail;

    cout << "    OUT-RE-100: Setting/getting double fields.";
    {
        const double in = 0.1;
        double out;
        record.SetField("double", in);
        record.GetField("double", out);
        fail = in != out || record.GetType("double") != TOutRecord::kDouble;
    }
    cout << (fail ? " [failed]" : " [passed]") << endl;
    anyFail |= fail;

    cout << "    OUT-RE-110: Setting/getting string fields.";
    {
        const string in = "one ten";
        string out;
        record.SetField("string", in);
        record.GetField("string", out);
        fail = in != out || record.GetType("string") != TOutRecord::kString;
    }
    cout << (fail ? " [failed]" : " [passed]") << endl;
    anyFail |= fail;

    cout << "    OUT-RE-120: Getting map.";
    fail = record.GetMap().Extent() != 11;
    cout << (fail ? " [failed]" : " [passed]") << endl;
    anyFail |= fail;

    cout << "    OUT-RE-130: Clearing field.";
    {
        char out;
        record.SetField("char");
        record.GetField("char", out);
        fail = out != 0;
    }
    cout << (fail ? " [failed]" : " [passed]") << endl;
    anyFail |= fail;

    cout << "    OUT-RE-140: Bad cast.";
    {
        char out;
        record.GetField("string", out);
        fail = out != 0;
    }
    cout << (fail ? " [failed]" : " [passed]") << endl;
    anyFail |= fail;

    cout << "    OUT-RE-150: Bad field.";
    {
        string out;
        record.GetField("undefined", out);
        fail = out != "" || record.GetType("undefined") != TOutRecord::kNoType;
    }
    cout << (fail ? " [failed]" : " [passed]") << endl;
    anyFail |= fail;

    cout << (anyFail ? "    F" : "    No f") << "ailures occurred." << endl;
    return !anyFail;
}
```

E. TestStorage.C

```
//  Project: TRANSIMS
//  Subsystem: Simulation Output
//  $RCSfile: TestStorage.C,v $
//  $Revision: 0.5 $
//  $Date: 1996/06/19 17:42:21 $
//  $State: Stab $
//  $Author: bwb $
//  U.S. Government Copyright 1995
//  All rights reserved


//  Include Standard C++ header files.
```

```
#include <iostream.h>


//  Include TRANSIMS header files.
#include <OUT/Storage.h>


//  Test storage class.
bool TestStorage()
{
    cout << "  Storage Class Tests"
        << " [$Revision: 0.5 $]"
        << endl;

    bool anyFail = FALSE;
    bool fail;

    TOutStorage* stg;
    TOutRecord* inrec1;
    TOutRecord* inrec2;
    TOutRecord* outrec;

    stg = new TOutStorage("/home/projects/transims/test/NET", "temp",
            TOutStorage::kWrite);

    cout << "    OUT-ST-010: Root retrieval.";
    fail = stg->GetRoot() != "/home/projects/transims/test/NET";
    cout << (fail ? " [failed]" : " [passed]") << endl;
    anyFail |= fail;

    cout << "    OUT-ST-020: Name retrieval.";
    fail = stg->GetName() != "temp";
    cout << (fail ? " [failed]" : " [passed]") << endl;
    anyFail |= fail;

    cout << "    OUT-ST-030: Host retrieval.";
    TOutStorage::HostSet hosts = stg->GetHosts();
    {
        TOutStorage::HostSetIterator i(stg->GetHosts());
        fail = *i.CurrentItem() != "local";
        i.Next();
        fail |= !i.IsDone();
    }
    cout << (fail ? " [failed]" : " [passed]") << endl;
    anyFail |= fail;

    inrec1 = new TOutRecord();
    inrec1->SetField("One", 1.0);
    inrec1->SetField("Two", 2);
    inrec1->SetField("Three", "three");

    inrec2 = new TOutRecord();
    inrec2->SetField("One", "one");
    inrec2->SetField("Two", 2.0);
    inrec2->SetField("Three", 3);

    stg->WriteHeader(*inrec1);
    stg->Write(*inrec1);
    stg->Write(*inrec2);

    const long pos = stg->GetOffset();

    delete stg;

    hosts.Clear();
    hosts.Add("local");
    stg = new TOutStorage(hosts, "/home/projects/transims/test/NET", "temp");
    outrec = new TOutRecord();

    cout << "    OUT-ST-040: Check for not end-of-file.";
    fail = stg->AtEnd();
    cout << (fail ? " [failed]" : " [passed]") << endl;
    anyFail |= fail;

    cout << "    OUT-ST-050: Header write/read.";
    stg->ReadHeader(*outrec);
```

```
    {
        const TOutRecord::FieldMap& inmap = inrec1->GetMap();
        const TOutRecord::FieldMap& outmap = outrec->GetMap();
        fail = inmap.Extent() != outmap.Extent();
        for (TOutRecord::FieldMapIterator i(inmap); !i.IsDone(); i.Next())
            fail |= !outmap.IsBound(*i.CurrentItem());
    }
    cout << (fail ? " [failed]" : " [passed]") << endl;
    anyFail |= fail;

    cout << "    OUT-ST-060: Record write/read.";
    stg->Read(*outrec);
    fail = outrec->GetMap() != inrec1->GetMap();
    stg->Read(*outrec);
    fail |= outrec->GetMap() != inrec2->GetMap();
    cout << (fail ? " [failed]" : " [passed]") << endl;
    anyFail |= fail;

    cout << "    OUT-ST-070: Get offset.";
    fail = stg->GetOffset() != pos;
    cout << (fail ? " [failed]" : " [passed]") << endl;
    anyFail |= fail;

    cout << "    OUT-ST-080: Seek to begin.";
    stg->Seek(TOutStorage::kBegin);
    fail = stg->GetOffset() != 0;
    cout << (fail ? " [failed]" : " [passed]") << endl;
    anyFail |= fail;

    cout << "    OUT-ST-090: Seek to end.";
    stg->Seek(TOutStorage::kEnd);
    fail = stg->GetOffset() != pos;
    cout << (fail ? " [failed]" : " [passed]") << endl;
    anyFail |= fail;

    cout << "    OUT-ST-100: Check for end-of-file.";
    stg->Read(*outrec);
    fail = !stg->AtEnd();
    cout << (fail ? " [failed]" : " [passed]") << endl;
    anyFail |= fail;

    cout << "    OUT-ST-110: Seek to middle.";
    stg->Seek(pos / 2);
    fail = stg->GetOffset() != pos / 2;
    cout << (fail ? " [failed]" : " [passed]") << endl;
    anyFail |= fail;

    cout << "    OUT-ST-120: Invalid host.";
    try {
        stg->GetOffset("not a host");
        fail = TRUE;
    } catch(const TOutStorageFailure& exception) {
        fail = FALSE;
    }
    cout << (fail ? " [failed]" : " [passed]") << endl;
    anyFail |= fail;

    delete inrec1;
    delete inrec2;
    delete outrec;
    delete stg;

    cout << (anyFail ? "    F" : "    No f") << "ailures occurred." << endl;
    return !anyFail;
}
```

F. TestTextWriter.C

```
//  Project: TRANSIMS
//  Subsystem: Simulation Output
//  $RCSfile: TestTextWriter.C,v $
//  $Revision: 0.6 $
//  $Date: 1996/06/19 17:42:32 $
//  $State: Stab $
//  $Author: bwb $
```

```
//  U.S. Government Copyright 1995
//  All rights reserved


//  Include Standard C++ header files.
#include <iostream.h>


//  Include TRANSIMS header files.
#include <OUT/TextWriter.h>


//  Test text writer class.
bool TestTextWriter()
{
    cout << "  Text Writer Class Tests"
        << " [$Revision: 0.6 $]"
        << endl;

    bool anyFail = FALSE;
    bool fail;

    ifstream* in;
    TOutTextWriter* writer;
    TOutRecord record;
    TOutTextWriter::FieldCollection fields;
    char line[1000];

    writer = new TOutTextWriter("/home/projects/transims/test/NET/test", ",",
        TRUE);

    fields.Append("One");
    fields.Append("Two");
    fields.Append("Three");

    record.SetField("One", 1);
    record.SetField("Two", 2.0);
    record.SetField("Three", "three");
    writer->Write(record, fields);

    record.SetField("One", "one");
    record.SetField("Two", 2);
    record.SetField("Three", 3.0);
    writer->Write(record, fields);

    delete writer;

    in = new ifstream("/home/projects/transims/test/NET/test");

    cout << "    OUT-TW-010: Header writing.";
    in->getline(line, 1000);
    fail = strcmp(line, "One,Two,Three") != 0;
    cout << (fail ? " [failed]" : " [passed]") << endl;
    anyFail |= fail;

    cout << "    OUT-TW-020: Data writing.";
    in->getline(line, 1000);
    fail = strcmp(line, ",2.000000,three") != 0;
    in->getline(line, 1000);
    fail = strcmp(line, "one,2,3.000000") != 0;
    cout << (fail ? " [failed]" : " [passed]") << endl;
    anyFail |= fail;

    delete in;

    writer = new TOutTextWriter("/home/projects/transims/test/NET/test");
    writer->Write(record);
    delete writer;

    in = new ifstream("/home/projects/transims/test/NET/test");

    cout << "    OUT-TW-030: Default delimiter.";
    in->getline(line, 1000);
    fail = strcmp(line, "3.000000\t2\tone") != 0;
    cout << (fail ? " [failed]" : " [passed]") << endl;
    anyFail |= fail;
```

```
    delete in;

    cout << (anyFail ? "    F" : "    No f") << "ailures occurred." << endl;
    return !anyFail;
}
```

A. TestSpecification.import

```
Sample Output Specification Table
This is a sample output specification table.
OUTSPECSAMP1
Output Specification
CREATE TABLE OUTSPECSAMP1 (
  PROCESSOR VARCHAR(50),
  ROOT VARCHAR(50),
  NAME VARCHAR(50),
  TIMEMIN NUMBER(10),
  TIMEMAX NUMBER(10),
  TIMESTP NUMBER(10),
  TIMESMP NUMBER(10),
  BOXLEN FLOAT,
  ABSCISSAMIN FLOAT,
  ABSCISSAMAX FLOAT,
  ORDINATEMIN FLOAT,
  ORDINATEMAX FLOAT,
  PRIMARY KEY (NAME)
);
PROCESSOR, ROOT, NAME, TIMEMIN, TIMEMAX, TIMESTP, TIMESMP, BOXLEN, ABSCISSAMIN,
ABSCISSAMAX, ORDINATEMIN, ORDINATEMAX
'Evolution', '/home/projects/transims/test/NET','sample',60,120,5,10,150,1,1000,2,2000

Sample Output Node Specification Table
This is a sample output node specification table.
OUTNODESPECSAMP1
Output Node Specification
CREATE TABLE OUTNODESPECSAMP1 (
  NAME VARCHAR(50),
  NODE NUMBER(10),
  PRIMARY KEY (NAME,NODE)
);
NAME, NODE
'sample',1
'sample',2

Sample Output Link Specification Table
This is a sample output link specification table.
OUTLINKSPECSAMP1
Output Link Specification
CREATE TABLE OUTLINKSPECSAMP1 (
  NAME VARCHAR(50),
  LINK NUMBER(10),
  PRIMARY KEY (NAME,LINK)
);
NAME, LINK
'sample',11
'sample',12
```

X. APPENDIX: Source Creation Utility

This appendix contains the complete C++ source code for the simulation output subsystem data source creation utility.

A. CreateSources.C

```
//  Project: TRANSIMS
//  Subsystem: Simulation Output
//  $RCSfile: CreateSources.C,v $
//  $Revision: 1.5 $
//  $Date: 1996/06/19 17:05:42 $
//  $State: Exp $
//  $Author: bwb $
//  U.S. Government Copyright 1995
```

```
//  All rights reserved


//  Include Standard C++ header files.
#include <iostream.h>


//  Include TRANSIMS header files.
#include <GBL/Globals.h>
#include <DBS/Directory.h>
#include <DBS/Exception.h>


//  Main program.
int main(int, char*[])
{
    try {

        TDbDirectory directory(TDbDirectoryDescription("IOC-1"));

        if (!directory.HasSource("Output Specification"))
            directory.CreateSource(TDbSourceDescription("Output Specification",
                    "This data source contains simulation output "
                    "specifications."));

        if (!directory.HasSource("Output Node Specification"))
            directory.CreateSource(TDbSourceDescription("Output Node "
                    "Specification", "This data source contains simulation "
                    "output node specifications."));

        if (!directory.HasSource("Output Link Specification"))
            directory.CreateSource(TDbSourceDescription("Output Link "
                    "Specification", "This data source contains simulation "
                    "output link specifications."));

    } catch(const TDbException& dbException) {

        cerr << "Database exception: " << dbException.GetMessage() << endl;

    } catch(...) {

        cerr << "Unknown exception." << endl;

    }

    return 0;
}
```

XI. APPENDIX: Storage Dump Utility

This appendix contains the complete C++ source code for the simulation output subsystem storage dump utility.

A. DumpStorage.C

```
//  Project: TRANSIMS
//  Subsystem: Simulation Output
//  $RCSfile: DumpStorage.C,v $
//  $Revision: 0.1 $
//  $Date: 1996/06/19 17:10:24 $
//  $State: Stab $
//  $Author: bwb $
//  U.S. Government Copyright 1995
//  All rights reserved


//  Include Standard C++ header files.
#include <iostream.h>


//  Include TRANSIMS header files.
#include <OUT/Record.h>
#include <OUT/Storage.h>
```

```
#include <OUT/TextWriter.h>


//  Main program.
int main(int argc, char* argv[])
{
    //  Check arguments.
    if (argc < 4) {
        cerr << "Usage: DumpStorage outname root name hosts..." << endl;
        return -1;
    }

    //  Create the output file.
    TOutTextWriter writer(argv[1], "\t", TRUE);

    //  Make the host list.
    TOutStorage::HostSet hosts(HashValue);
    for (int h = 4; h < argc; ++h)
        hosts.Add(argv[h]);

    //  Open the storage.
    TOutStorage storage(hosts, argv[2], argv[3], TOutStorage::kRead);

    //  Setup input.
    TOutRecord record;
    TOutStorage::HostSetIterator i(storage.GetHosts());
    for (i.Reset(); !i.IsDone(); i.Next()) {
        const string& host = *i.CurrentItem();
        storage.Seek(host, TOutStorage::kBegin);
        storage.ReadHeader(host, record);
    }

    //  Setup output.
    TOutWriter::FieldCollection fields;
    for (TOutRecord::FieldMapIterator f = record.GetIterator(); !f.IsDone();
            f.Next())
        fields.Append(*f.CurrentItem());

    //  Transfer data.
    for (i.Reset(); !i.IsDone(); i.Next()) {
        const TOutStorage::HostHandle host =
                storage.GetHostHandle(*i.CurrentItem());
        for (storage.Read(host, record); !storage.AtEnd(host);
                storage.Read(host, record))
            writer.Write(record, fields);
    }

    return 0;
}
```