COORDINATED HIGHWAYS ACTION RESPONSE TEAM
STATE HIGHWAY ADMINISTRATION

CHART Release 6
Mapping Release 5
Detailed Design

Contract SHA-06-CHART
Document # WO19-DS-001
Work Order 19, Deliverable 4

September 21, 2010
By
CSC

Revision	Description	Pages Affected	Date
0	Initial Release	All	09/21/2010

Table of Contents

1 Introduction 1-1

1.1 Purpose 1-1

1.2 Objectives 1-2

1.3 Scope 1-2

1.4 Design Process 1-2

1.5 Design Tools 1-2

1.6 Work Products 1-2

2 Architecture 2-1

2.1 Network/Hardware 2-1

2.1.1 Lane Configuration Web Service 2-1

2.2 Software 2-1

2.2.1 COTS Products 2-2

2.2.1.1 CHART 2-2

2.2.1.2 Mapping 2-5

2.2.2 Deployment /Interface Compatibility 2-5

2.2.2.1 External Interfaces 2-5

2.2.2.2 Internal Interfaces 2-9

2.3 Security 2-9

2.4 Data 2-10

2.4.1 Data Storage 2-10

2.4.1.1 Database 2-10

2.4.1.2 CHART Flat Files 2-30

2.4.2 Database Design 2-32

2.4.2.1 Lane Configuration 2-32

2.4.2.2 Integrated Map 2-33

2.4.2.3 Mapping 2-33

2.4.2.4 Archiving - Changes 2-33

2.4.3 Error Processing 2-34

3 Key Design Concepts 3-1

3.1 EORS Road Closure Permit Integration 3-1

3.2 Improved Map Lane Configuration Data 3-1

3.2.1 CHART 3-1

3.2.2 Intranet Mapping 3-3

3.3 Map Integration Part 2 – Locate Between Features 3-3

3.4 External TSSs on Intranet Map 3-4

3.4.1 CHART 3-4

3.4.2 Intranet Mapping 3-5

3.5 Camera Location Data Synchronization **3-5**
3.6 Error Processing **3-6**
3.7 Packaging **3-6**
3.7.1 CHART 3-6
3.7.2 Mapping 3-8
3.8 Assumptions and Constraints **3-9**
4 Use Cases – EORS Integration **4-1**
4.1 CHART **4-1**
4.1.1 ManageTrafficEvents (Use Case Diagram) 4-1
4.1.1.1 Add Text to Event History (Use Case) 4-1
4.1.1.2 Associate Event (Use Case) 4-1
4.1.1.3 Change Event Attributes (Use Case) 4-2
4.1.1.4 Change Event Type (Use Case) 4-2
4.1.1.5 Change Lane Direction (Use Case) 4-2
4.1.1.6 Close Event (Use Case) 4-2
4.1.1.7 Copy Traffic Event (Use Case) 4-2
4.1.1.8 Create Incident Event (Use Case) 4-2
4.1.1.9 Create Traffic Event (Use Case) 4-2
4.1.1.10 Edit Traffic Event Lane Configuration and Status (Use Case) 4-2
4.1.1.11 Get Event History Text (Use Case) 4-2
4.1.1.12 Merge Traffic Events (Use Case) 4-3
4.1.1.13 Modify Traffic Event (Use Case) 4-3
4.1.1.14 Record Lane Closure (Use Case) 4-3
4.1.1.15 Record Organization Notification And Arrival (Use Case) 4-3
4.1.1.16 Record Resource Notification And Arrival (Use Case) 4-3
4.1.1.17 Respond to Traffic Event (Use Case) 4-3
4.1.1.18 Search EORS Permits (Use Case) 4-3
4.1.1.19 Search Traffic Events (Use Case) 4-4
4.1.1.20 Set EORS Permit for Planned Roadway Closure Event (Use Case) 4-4
4.1.1.21 Specify Event Location (Use Case) 4-4
4.1.1.22 Specify Expected Duration (Use Case) 4-5
4.1.1.23 Specify WebsiteTraffic Alert Settings (Use Case) 4-5
4.1.1.24 Take Event Offline (Use Case) 4-5
4.1.1.25 View Lane Configuration and Status Textually (Use Case) 4-5
4.1.1.26 View Potential Duplicate Events (Use Case) 4-5
4.1.1.27 View Suggested EORS Permits (Use Case) 4-5
4.1.1.28 View Traffic Events (Use Case) 4-5
5 Detailed Design – EORS Integration **5-6**
5.1 Human-Machine Interface **5-6**
5.1.1 EORS Permit Searching 5-6
5.2 System Interfaces **5-17**
5.2.1 Class Diagrams 5-17
5.2.1.1 EORS (Class Diagram) 5-17
5.3 EORS Module **5-19**
5.3.1 Class Diagrams 5-19
5.3.1.1 EORSModuleClasses (Class Diagram) 5-19

5.3.2 Sequence Diagrams 5-21
5.3.2.1 EORSSystemImpl:getPermitsModifiedSince (Sequence Diagram) 5-21
5.4 EORS GUI 5-22
5.4.1 Class Diagrams 5-22
5.4.1.1 EORSDataClasses (Class Diagram) 5-22
5.4.1.2 EORSServletClasses (Class Diagram) 5-24
5.4.2 Sequence Diagrams 5-26
5.4.2.1 EORSReqHdlr:handleSearchEORSPermitsRequest (Sequence Diagram) 5-26
5.4.2.2 EORSReqHdlr:handleNewEORSPermitSearch (Sequence Diagram) 5-27
5.4.2.3 EORSReqHdlr:handleExistingEORSPermitSearch (Sequence Diagram) 5-28
5.4.2.4 EORSReqHdlr:handleSuggestEORSPermitsJSONRequest (Sequence Diagram) 5-29
6 Use Cases – Lane Configuration 6-1
6.1 CHART 6-1
6.1.1 LaneConfiguration (Use Case Diagram) 6-1
6.1.1.1 Change Lane Direction (Use Case) 6-1
6.1.1.2 Customize Lane Configuration (Use Case) 6-2
6.1.1.3 Edit Lane Configuration and Status (Use Case) 6-2
6.1.1.4 Edit Traffic Event Lane Configuration and Status (Use Case) 6-2
6.1.1.5 Override Lane Status Change Time (Use Case) 6-2
6.1.1.6 Record Lane Closure (Use Case) 6-2
6.1.1.7 Render Lane Configuration and Status (Use Case) 6-3
6.1.1.8 Select Lane Configuration (Use Case) 6-3
6.1.1.9 Set Nearby Lane Configuration Search Radius (Use Case) 6-3
6.1.1.10 Specify Lane Configuration (Use Case) 6-3
6.1.1.11 Store User Specified Lane Config (Use Case) 6-4
6.1.1.12 View Lane Configuration and Status Textually (Use Case) 6-4
6.1.1.13 View Lane Configuration and Status Graphically (Use Case) 6-4
6.1.1.14 View Nearby Lane Configurations (Use Case) 6-4
6.1.1.15 View Traffic Event Lane Status (Use Case) 6-4
6.1.2 ManageTrafficEvents (Use Case Diagram) 6-5
6.1.2.1 Add Text to Event History (Use Case) 6-5
6.1.2.2 Associate Event (Use Case) 6-5
6.1.2.3 Change Event Attributes (Use Case) 6-5
6.1.2.4 Change Event Type (Use Case) 6-5
6.1.2.5 Change Lane Direction (Use Case) 6-6
6.1.2.6 Close Event (Use Case) 6-6
6.1.2.7 Copy Traffic Event (Use Case) 6-6
6.1.2.8 Create Incident Event (Use Case) 6-6
6.1.2.9 Create Traffic Event (Use Case) 6-6
6.1.2.10 Edit Traffic Event Lane Configuration and Status (Use Case) 6-6
6.1.2.11 Get Event History Text (Use Case) 6-6
6.1.2.12 Merge Traffic Events (Use Case) 6-6
6.1.2.13 Modify Traffic Event (Use Case) 6-6
6.1.2.14 Record Lane Closure (Use Case) 6-7
6.1.2.15 Record Organization Notification And Arrival (Use Case) 6-7
6.1.2.16 Record Resource Notification And Arrival (Use Case) 6-7
6.1.2.17 Respond to Traffic Event (Use Case) 6-7
6.1.2.18 Search EORS Permits (Use Case) 6-7
6.1.2.19 Search Traffic Events (Use Case) 6-7
6.1.2.20 Set EORS Permit for Planned Roadway Closure Event (Use Case) 6-8
6.1.2.21 Specify Event Location (Use Case) 6-8

6.1.2.22 Specify Expected Duration (Use Case) 6-8
6.1.2.23 Specify WebsiteTraffic Alert Settings (Use Case) 6-9
6.1.2.24 Take Event Offline (Use Case) 6-9
6.1.2.25 View Lane Configuration and Status Textually (Use Case) 6-9
6.1.2.26 View Potential Duplicate Events (Use Case) 6-9
6.1.2.27 View Suggested EORS Permits (Use Case) 6-9
6.1.2.28 View Traffic Events (Use Case) 6-9

6.2 Mapping 6-10

6.2.1 GISLaneConfigWebService (Use Case Diagram) 6-10

6.2.1.1 Find Nearby Lane Configurations by Location (Use Case) 6-10
6.2.1.2 Store Lane Configuration for a Location & Route Information (Use Case) 6-10

7 Detailed Design – Lane Configuration 7-11

7.1 Human-Machine Interface 7-11

7.1.1 Lane Configuration 7-11

7.1.1.1 Traffic Event Details Page 7-11
7.1.1.2 Lane Editor Popup 7-12

7.2 System Interfaces 7-21

7.2.1 Class Diagrams 7-21

7.2.1.1 TrafficEventManagement (Class Diagram) 7-21

7.3 Lane Config Utility Package 7-27

7.3.1 Class Diagrams 7-27

7.3.1.1 LaneConfigModelClasses (Class Diagram) 7-27
7.3.1.2 LaneDisplayClasses (Class Diagram) 7-29
7.3.1.3 LaneConfigIDLUtilClasses (Class Diagram) 7-31

7.4 Utility Package 7-32

7.4.1 Class Diagrams 7-32

7.4.1.1 UtilityClasses3 (Class Diagram) 7-32

7.5 chartlite.servlet.trafficevents 7-33

7.5.1 Class Diagrams 7-33

7.5.1.1 chartlite.servlet.trafficevents_classes (Class Diagram) 7-33

7.5.2 Sequence Diagrams 7-35

7.5.2.1 LaneConfigReqHdlr:createInitLaneConfigEditorRequest2 (Sequence Diagram) 7-35
7.5.2.2 LaneConfigReqHdlr:initLaneConfigEditor (Sequence Diagram) 7-36
7.5.2.3 LaneConfigReqHdlr:laneEditorSubmitCallback (Sequence Diagram) 7-38

7.6 webservices.laneeditormodule 7-39

7.6.1 Class Diagrams 7-39

7.6.1.1 LaneEditorClassDiagram (Class Diagram) 7-39

7.6.2 Sequence Diagrams 7-42

7.6.2.1 LaneEditorClient:LaneEditorInitializationFromClient (Sequence Diagram) 7-42
7.6.2.2 LaneEditorClient:LaneEditorSubmitToClient (Sequence Diagram) 7-43
7.6.2.3 LaneEditorModule:getNearbyLaneConfigs (Sequence Diagram) 7-44
7.6.2.4 LaneEditorRequestHandler:addLane (Sequence Diagram) 7-45
7.6.2.5 LaneEditorRequestHandler:endLaneEditorSession (Sequence Diagram) 7-46
7.6.2.6 LaneEditorRequestHandler:initializeEditingSession (Sequence Diagram) 7-47
7.6.2.7 LaneEditorRequestHandler:marshall (Sequence Diagram) 7-48
7.6.2.8 LaneEditorRequestHandler:removeAllLanes (Sequence Diagram) 7-49

7.6.2.9 LaneEditorRequestHandler:removeLanes (Sequence Diagram) 7-50
7.6.2.10 LaneEditorRequestHandler:sendLaneConfigGraphicJSON (Sequence Diagram) 7-52
7.6.2.11 LaneEditorRequestHandler:sendLaneEditorCallbackRequest (Sequence Diagram) 7-53
7.6.2.12 LaneEditorRequestHandler:sendStoreLaneConfigRequest (Sequence Diagram) 7-54
7.6.2.13 LaneEditorRequestHandler:setAllLanesOpen (Sequence Diagram) 7-55
7.6.2.14 LaneEditorRequestHandler:setLanesState (Sequence Diagram) 7-56
7.6.2.15 LaneEditorRequestHandler:setLanesStateChangedTime (Sequence Diagram) 7-57
7.6.2.16 LaneEditorRequestHandler:setLanesTrafficFlowDir (Sequence Diagram) 7-58
7.6.2.17 LaneEditorRequestHandler:useNearbyLaneConfig (Sequence Diagram) 7-60
7.6.2.18 LaneEditorRequestHandler:useStandardLaneConfig (Sequence Diagram) 7-61
7.6.2.19 LaneEditorRequestHandler:viewLaneEditor (Sequence Diagram) 7-62

7.7 webservices.wsutil 7-63

7.7.1 Class Diagrams 7-63

7.7.1.1 webservices.util-classes (Class Diagram) 7-63

7.8 webservices.wsutil.jaxbutil 7-64

7.8.1 Class Diagrams 7-64

7.8.1.1 JAXBUtilClasses (Class Diagram) 7-64

7.8.2 Sequence Diagrams 7-65

7.8.2.1 JAXBWebServiceClient:get (Sequence Diagram) 7-65
7.8.2.2 JAXBWebServiceClient:post (Sequence Diagram) 7-66
7.8.2.3 JAXBXMLHTTPHelper:handleResponse (Sequence Diagram) 7-67
7.8.2.4 JAXBXMLHTTPHelper:marshall (Sequence Diagram) 7-67
7.8.2.5 JAXBXMLHTTPHelper:prepareRequest (Sequence Diagram) 7-68
7.8.2.6 JAXBXMLHTTPHelper:unmarshall (Sequence Diagram) 7-69

7.9 xsdutil.common 7-70

7.9.1 Class Diagrams 7-70

7.9.1.1 XSDUtilCommonClasses (Class Diagram) 7-70

7.10 xsdutil.laneconfig 7-71

7.10.1 Class Diagrams 7-71

7.10.1.1 XSDUtilLaneConfigClasses (Class Diagram) 7-71

7.11 Mapping - Lane Config Web Service 7-72

7.11.1 Class Diagrams 7-72

7.11.1.1 GISLaneConfig (Class Diagram) 7-72
7.11.1.2 GISLaneConfigHelper (Class Diagram) 7-74

7.11.2 Sequence Diagrams 7-77

7.11.2.1 LaneConfigGISWebService:getLaneConfiguration (Sequence Diagram) 7-77
7.11.2.2 LaneConfigGISWebService:getLaneConRslt (Sequence Diagram) 7-79
7.11.2.3 LaneConfigGISWebService:storeLaneConfiguration (Sequence Diagram) 7-80
7.11.2.4 LaneConfigGISWebService:storeLaneRecords (Sequence Diagram) 7-81

8 Use Cases – Map Between Features 8-1

8.1 CHART 8-1

8.1.1 R6HighLevel (Use Case Diagram) 8-1

8.1.1.1 Configure Duplicate Event Comparison Rules (Use Case) 8-1
8.1.1.2 Create Duplicate Event Alerts (Use Case) 8-2
8.1.1.3 Edit Traffic Event Lane Configuration and Status (Use Case) 8-2
8.1.1.4 Select First Intersecting Feature (Use Case) 8-2
8.1.1.5 Select Intersecting Feature (Use Case) 8-2

8.1.1.6 Select Second Intersecting Feature (Use Case) 8-2
8.1.1.7 Specify Intersecting Feature Pair (Use Case) 8-2
8.1.1.8 Specify Lane Configuration (Use Case) 8-3
8.1.1.9 Specify Object Location (Use Case) 8-3

9 Detailed Design – Map Between Features 9-4

9.1 Human-Machine Interface 9-4

9.1.1 "Between" as a Location Proximity 9-4
9.1.1.1 New Location Proximity Values 9-4
9.1.1.2 Specifying a Second Intersecting Feature 9-4
9.1.1.3 Feature Markers on the Map 9-8
9.1.2 Set Traffic Event Duplicate Comparison Rules 9-9

9.2 CHART Common 9-10

9.2.1 Class Diagrams 9-10
9.2.1.1 Common2 (Class Diagram) 9-10

9.3 GUI – Flex – Edit Location (chartlite/Flex/editlocation) 9-13

9.3.1 Class Diagrams 9-13
9.3.1.1 FlexLocationClasses (Class Diagram) 9-13
9.3.1.2 GUIFlexComponentsClasses (Class Diagram) 9-16
9.3.2 Sequence Diagrams 9-17
9.3.2.1 flex.shared.components:SpecifyLocation.updateIntFeatures (Sequence Diagram) 9-17

9.4 GUI – Javascript – Open Layers (chartlite/javascript/OpenLayers) 9-18

9.4.1 Class Diagrams 9-18
9.4.1.1 MapViewSpecificClasses (Class Diagram) 9-18
9.4.2 Sequence Diagrams 9-20
9.4.2.1 SpecifyLocation:jsUpdateMapGeoLocInfo (Sequence Diagram) 9-20

9.5 Utility Package 9-21

9.5.1 Class Diagrams 9-21
9.5.1.1 UtilityClasses2 (Class Diagram) 9-21
9.5.1.2 LocationRelatedProxyClasses (Class Diagram) 9-25
9.5.2 Sequence Diagrams 9-27
9.5.2.1 ProxyBasicTrafficEvent2:areLocationsTheSame (Sequence Diagram) 9-27

10 Use Cases – External TSS 10-1

10.1 CHART 10-1

10.1.1 Provide Data to External Systems (Use Case Diagram) 10-1
10.1.1.1 Add External Client (Use Case) 10-2
10.1.1.2 Authenticate External System (Use Case) 10-3
10.1.1.3 Edit External Client (Use Case) 10-3
10.1.1.4 Generate Key Pair (Use Case) 10-3
10.1.1.5 Manage External Clients (Use Case) 10-3
10.1.1.6 Provide Camera Data to External Systems (Use Case) 10-3
10.1.1.7 Provide Data to External Systems (Use Case) 10-3
10.1.1.8 Provide Detector Data to External Systems (Use Case) 10-4
10.1.1.9 Provide DMS Data to External Systems (Use Case) 10-4
10.1.1.10 Provide HAR Data to External Systems (Use Case) 10-4
10.1.1.11 Provide SHAZAM Data to External Systems (Use Case) 10-4

10.1.1.12 Provide Traffic Event Data to External Systems (Use Case) 10-5
10.1.1.13 Remove External Client (Use Case) 10-5
10.1.1.14 View External Clients (Use Case) 10-5

10.2 Mapping 10-5

10.2.1 CHART Intranet Map User Login (Use Case Diagram) 10-5

10.2.1.1 Not Login (Use Case) 10-6
10.2.1.2 Login (Use Case) 10-6
10.2.1.3 Display TSS (Use Case) 10-6

11 Detailed Design – External TSS 11-7

11.1 Human-Machine Interface 11-7

11.1.1 CHART Intranet Mapping GUI 11-7

11.1.1.1 Login to CHART 11-7
11.1.1.2 Display External TSS 11-9

11.2 Mapping 11-11

11.2.1 Class Diagrams 11-11

11.2.1.1 Configuration - Intranet Map (Class Diagram) 11-11
11.2.1.2 User Validation - Intranet Map (Class Diagram) 11-11
11.2.1.3 User Rights – Intranet Map (Class Diagram) 11-12
11.2.1.4 Response Data – Intranet Map (Class Diagram) 11-13
11.2.1.5 Client Request - Intranet Map (Class Diagram) 11-15

11.2.2 Sequence Diagrams 11-16

11.2.2.1 Login to CHART – Intranet Map (Sequence Diagram) 11-16
11.2.2.2 Keep Session Alive (Sequence Diagram) 11-18

11.3 CHART Data Exporter Web Service 11-19

11.3.1 Class Diagrams 11-19

11.3.1.1 WSUserManagementModule (Class Diagram) 11-19
11.3.1.2 WSDMSExportModule (Class Diagram) 11-21

11.3.2 Sequence Diagrams 11-26

11.3.2.1 DMSExportHandler:getDMSInventoryList (Sequence Diagram) 11-26
11.3.2.2 DMSRequestHandler:processRequest (Sequence Diagram) 11-28
11.3.2.3 DMSSubscriptionRequestHandler:processRequest (Sequence Diagram) 11-29
11.3.2.4 DMSExportHandler:getDMSStatusList (Sequence Diagram) 11-30

11.4 CHART User Management Web Service 11-31

11.4.1 Sequence Diagrams 11-31

11.4.1.1 UserManagerRequestHandler:processRequest (Sequence Diagram) 11-31
11.4.1.2 UserManagerRequestHandler:handleException (Sequence Diagram) 11-32

12 Use Cases – Camera Synchronization 12-1

12.1 CHART Data Exporter 12-1

12.1.1 Provide Data to External Systems (Use Case Diagram) 12-1

12.1.1.1 Add External Client (Use Case) 12-2
12.1.1.2 Authenticate External System (Use Case) 12-3
12.1.1.3 Edit External Client (Use Case) 12-3
12.1.1.4 Generate Key Pair (Use Case) 12-3
12.1.1.5 Manage External Clients (Use Case) 12-3
12.1.1.6 Provide Camera Data to External Systems (Use Case) 12-3
12.1.1.7 Provide Data to External Systems (Use Case) 12-3

12.1.1.8 Provide Detector Data to External Systems (Use Case) 12-4
12.1.1.9 Provide DMS Data to External Systems (Use Case) 12-4
12.1.1.10 Provide HAR Data to External Systems (Use Case)....12-4
12.1.1.11 Provide SHAZAM Data to External Systems (Use Case) 12-4
12.1.1.12 Provide Traffic Event Data to External Systems (Use Case)....12-5
12.1.1.13 Remove External Client (Use Case) 12-5
12.1.1.14 View External Clients (Use Case) 12-5

12.2 Data Exporter Client 12-5

12.2.1 R6ExportClientImportDataFromDataExporter (Use Case Diagram) 12-5

12.2.1.1 Create new Detector record in database (Use Case) 12-6
12.2.1.2 Create new DMS record in database (Use Case) 12-7
12.2.1.3 Create new HARrecord in database (Use Case) 12-7
12.2.1.4 Create new SHAZAM record in database (Use Case) 12-7
12.2.1.5 Create new Traffic Event record in database (Use Case)....12-7
12.2.1.6 Create new Camera record in database (Use Case)....12-7
12.2.1.7 Import Camera Data (Use Case) 12-7
12.2.1.8 Import Data from Data Exporter (Use Case) 12-7
12.2.1.9 Import Detector Data (Use Case)....12-7
12.2.1.10 Import DMS Data (Use Case)....12-7
12.2.1.11 Import HAR Data (Use Case) 12-7
12.2.1.12 Import SHAZAM Data (Use Case)....12-7
12.2.1.13 Import Traffic Data (Use Case) 12-8
12.2.1.14 Send HTTP Request to Map application (Use Case) 12-8
12.2.1.15 Send on_demand requset to DataExporter (Use Case) 12-8
12.2.1.16 Send subscriptions request to DataExporter (Use Case) 12-8
12.2.1.17 Update Camera record in database (Use Case) 12-8
12.2.1.18 Update Detector record in database (Use Case)....12-8
12.2.1.19 Update DMS record in database (Use Case)....12-8
12.2.1.20 Update HAR record in database (Use Case) 12-8
12.2.1.21 Update SHAZAM record in database (Use Case)....12-8
12.2.1.22 Update Traffic Event record in database (Use Case) 12-8

12.2.2 R6ExportClientPostRequestToMapApplication (Use Case Diagram)....12-8

12.2.2.1 Camera config updated (Use Case)....12-9
12.2.2.2 DMS config updated (Use Case) 12-9
12.2.2.3 HAR config updated (Use Case)....12-9
12.2.2.4 SendH TTP To Map Application (Use Case) 12-9
12.2.2.5 SHAZAM config updated (Use Case) 12-9
12.2.2.6 Traffic event config updated (Use Case) 12-10
12.2.2.7 TSSS config updated (Use Case)....12-10

12.3 Mapping Synchronization Application 12-10

12.3.1 Intranet Map Synchronization (Use Case Diagram) 12-10

12.3.1.1 Add CHART Events & Devices (Use Case)....12-10
12.3.1.2 Update CHART Events & Devices (Use Case) 12-11
12.3.1.3 Remove CHART Devices (Use Case) 12-11

13 Detailed Design – Camera Synchronization 13-12

13.1 Human-Machine Interface.... 13-12

13.1.1 Data Exporter Server and Client 13-12

13.1.2 CHART Intranet Mapping GUI 13-12

13.2 CHART Data Exporter 13-12

13.2.1 Class Diagrams 13-12

13.2.1.1 WebServicesBaseClasses (Class Diagram) 13-12
13.2.1.2 WSTrafficEventExportModuleClasses (Class Diagram) 13-16
13.2.1.3 DataExporterUtilityClasses (Class Diagram) 13-21
13.2.1.4 DMSSubscriptionSupportClasses (Class Diagram) 13-23
13.2.1.5 WSDMSExportModuleClasses (Class Diagram) 13-25
13.2.2 Sequence Diagrams 13-29
13.2.2.1 DMSExportHandler:getDMSInventoryList (Sequence Diagram) 13-29
13.2.2.2 DMSExportHandler:getDMSStatusList (Sequence Diagram) 13-30
13.2.2.3 DMSExportHandler:initialize (Sequence Diagram) 13-31
13.2.2.4 DMSRequestHandler:handleExceptions (Sequence Diagram) 13-32
13.2.2.5 DMSRequestHandler:processRequest (Sequence Diagram) 13-33
13.2.2.6 DMSSubscription:doPush (Sequence Diagram) 13-34
13.2.2.7 DMSSubscriptionMgr:creation (Sequence Diagram) 13-35
13.2.2.8 DMSSubscriptionMgr:initialize (Sequence Diagram) 13-36
13.2.2.9 DMSSubscriptionMgr:modelObserverUpdate (Sequence Diagram) 13-37
13.2.2.10 DMSSubscriptionMgr:updateSubscription (Sequence Diagram) 13-38
13.2.2.11 DMSSubscriptionRequestHandler:processRequest (Sequence Diagram) 13-39
13.2.2.12 WSDMSExportModule:initialize (Sequence Diagram) 13-40
13.2.2.13 WSDMSExportModule:shutdown (Sequence Diagram) 13-41
13.3 Data Exporter Client 13-42
13.3.1 Class Diagrams 13-42
13.3.1.1 ExportListenerModuleClasses (Class Diagram) 13-42
13.3.2 Sequence Diagrams 13-45
13.3.2.1 ExportListener:ProcessCameraInventory (Sequence Diagram) 13-45
13.3.2.2 ExportListener:ProcessCameraStatus (Sequence Diagram) 13-46
13.4 Mapping Synchronization Application 13-47
13.4.1 Class Diagrams 13-47
13.4.1.1 Data Exporter Synchronization Intranet Map (Class Diagram) 13-47
13.4.2 Sequence Diagrams 13-49
13.4.2.1 CHART Data Exporter Synchronization – (Sequence Diagram) 13-49
14 Deprecated Functionalities 14-1
14.1.1 EORS V2 14-1
14.1.1.1 Camera Hacking Page 14-1
14.1.2 CHART Device Editor 14-1
14.1.2.1 View, Add, Update, Remove CHART Devices 14-1
15 Mapping To Requirements 15-1
16 Acronyms/Glossary 16-1

Table of Figures

Figure 2-1 Lane Configuration Architecture Diagram 2-1
Figure 2-2 CHART and External Interfaces 2-6
Figure 2-3 R6 Server Deployment 2-8
Figure 2-4 R6 GUI Deployment 2-9
Figure 2-5 R6 ERD 2-20
Figure 3-1 Export TMDD Devices 3-6
Figure 4-1 ManageTrafficEvents (Use Case Diagram) 4-1
Figure 5-1 Planned Closure With No Permit Selected 5-6
Figure 5-2 EORS Permit Search Page 5-7
Figure 5-3 EORS Permit Suggestions 5-8
Figure 5-4 EORS Permit Suggestion Selection 5-9
Figure 5-5 No Permit Suggestions 5-10
Figure 5-6 Permit Search Results 5-11
Figure 5-7 More Permit Search Results 5-12
Figure 5-8 Show/Hide Permit Details 5-13
Figure 5-9 Use Search Result 5-14
Figure 5-10 Planned Closure Event with Associated EORS Permit 5-15
Figure 5-11 Clear Permit Confirmation 5-16
Figure 5-12 EORS (Class Diagram) 5-17
Figure 5-13 EORSModuleClasses (Class Diagram) 5-19
Figure 5-14 EORSSystemImpl:getPermitsModifiedSince (Sequence Diagram) 5-21
Figure 5-15 EORSDataClasses (Class Diagram) 5-22
Figure 5-16 EORSServletClasses (Class Diagram) 5-24
Figure 5-17 EORSReqHdlr:handleSearchEORSPermitsRequest (Sequence Diagram) 5-26
Figure 5-18 EORSReqHdlr:handleNewEORSPermitSearch (Sequence Diagram) 5-27
Figure 5-19 EORSReqHdlr:handleExistingEORSPermitSearch (Sequence Diagram) 5-28
Figure 5-20 EORSReqHdlr:handleSuggestEORSPermitsJSONRequest (Sequence Diagram) 5-30
Figure 6-1 LaneConfiguration (Use Case Diagram) 6-1
Figure 6-2 ManageTrafficEvents (Use Case Diagram) 6-5
Figure 6-3 GISLaneConfigWebService (Use Case Diagram) 6-10
Figure 7-1 Roadway Condtions Section in Traffic Event 7-11
Figure 7-2 Roadway Conditions Popup 7-11
Figure 7-3 Lane Editor Popup 7-12
Figure 7-4 Lane Editor Popup 7-13
Figure 7-5 Selecting a Lane Configuration 7-14
Figure 7-6 Lane Insertion Points 7-15
Figure 7-7 Lane Types for New Lane 7-15
Figure 7-8 Selecting Multiple Lanes 7-16
Figure 7-9 Empty Lane Configuration 7-17
Figure 7-10 CHART Default Status for Lane Editor 7-18
Figure 7-11 Setting Lane State Changed Time 7-18
Figure 7-12 Setting Lane State Changed Time (Advanced) 7-19
Figure 7-13 Textual Lane Status 7-20
Figure 7-14 Changed Not Saved Warning 7-20
Figure 7-15 TrafficEventManagement (Class Diagram) 7-22
Figure 7-16 LaneConfigModelClasses (Class Diagram) 7-28
Figure 7-17 LaneDisplayClasses (Class Diagram) 7-30
Figure 7-18 LaneConfigIDLUtilClasses (Class Diagram) 7-32
Figure 7-19 UtilityClasses3 (Class Diagram) 7-32
Figure 7-20 chartlite.servlet.trafficevents_classes (Class Diagram) 7-34
Figure 7-21 LaneConfigReqHdlr:createInitLaneConfigEditorRequest2 (Sequence Diagram) 7-36
Figure 7-22 LaneConfigReqHdlr:initLaneConfigEditor (Sequence Diagram) 7-37
Figure 7-23 LaneConfigReqHdlr:laneEditorSubmitCallback (Sequence Diagram) 7-39
Figure 7-24 LaneEditorClassDiagram (Class Diagram) 7-40

Figure 7-25 LaneEditorClient:LaneEditorInitializationFromClient (Sequence Diagram) 7-43
Figure 7-26 LaneEditorClient:LaneEditorSubmitToClient (Sequence Diagram) 7-44
Figure 7-27 LaneEditorModule:getNearbyLaneConfigs (Sequence Diagram) 7-45
Figure 7-28 LaneEditorRequestHandler:addLane (Sequence Diagram) 7-46
Figure 7-29 LaneEditorRequestHandler:endLaneEditorSession (Sequence Diagram) 7-47
Figure 7-30 LaneEditorRequestHandler:initializeEditingSession (Sequence Diagram) 7-48
Figure 7-31 LaneEditorRequestHandler:marshall (Sequence Diagram) 7-49
Figure 7-32 LaneEditorRequestHandler:removeAllLanes (Sequence Diagram) 7-50
Figure 7-33 LaneEditorRequestHandler:removeLanes (Sequence Diagram) 7-51
Figure 7-34 LaneEditorRequestHandler:renderLaneConfig (Sequence Diagram) 7-52
Figure 7-35 LaneEditorRequestHandler:sendLaneConfigGraphicJSON (Sequence Diagram) 7-53
Figure 7-36 LaneEditorRequestHandler:sendLaneEditorCallbackRequest (Sequence Diagram) 7-54
Figure 7-37 LaneEditorRequestHandler:sendStoreLaneConfigRequest (Sequence Diagram) 7-55
Figure 7-38 LaneEditorRequestHandler:setAllLanesOpen (Sequence Diagram) 7-56
Figure 7-39 LaneEditorRequestHandler:setLanesState (Sequence Diagram) 7-57
Figure 7-40 LaneEditorRequestHandler:setLanesStateChangedTime (Sequence Diagram) 7-58
Figure 7-41 LaneEditorRequestHandler:setLanesTrafficFlowDir (Sequence Diagram) 7-59
Figure 7-42 LaneEditorRequestHandler:unmarshall (Sequence Diagram) 7-60
Figure 7-43 LaneEditorRequestHandler:useNearbyLaneConfig (Sequence Diagram) 7-61
Figure 7-44 LaneEditorRequestHandler:useStandardLaneConfig (Sequence Diagram) 7-62
Figure 7-45 LaneEditorRequestHandler:viewLaneEditor (Sequence Diagram) 7-63
Figure 7-46 webservices.util-classes (Class Diagram) 7-64
Figure 7-47 JAXBUtilClasses (Class Diagram) 7-65
Figure 7-48 JAXBWebServiceClient:get (Sequence Diagram) 7-66
Figure 7-49 JAXBWebServiceClient:post (Sequence Diagram) 7-66
Figure 7-50 JAXBXMLHTTPHelper:handleResponse (Sequence Diagram) 7-67
Figure 7-51 JAXBXMLHTTPHelper:marshall (Sequence Diagram) 7-68
Figure 7-52 JAXBXMLHTTPHelper:prepareRequest (Sequence Diagram) 7-69
Figure 7-53 JAXBXMLHTTPHelper:unmarshall (Sequence Diagram) 7-70
Figure 7-54 XSDUtilCommonClasses (Class Diagram) 7-70
Figure 7-55 XSDUtilLaneConfigClasses (Class Diagram) 7-71
Figure 7-56 GISLaneConfig (Class Diagram) 7-72
Figure 7-57 GISLaneConfigHelper (Class Diagram) 7-75
Figure 7-58 LaneConfigGISWebService:getLaneConfiguration (Sequence Diagram) 7-78
Figure 7-59 LaneConfigGISWebService:getLaneConRslt (Sequence Diagram) 7-79
Figure 7-60 LaneConfigGISWebService:storeLaneConfiguration (Sequence Diagram) 7-80
Figure 7-61 LaneConfigGISWebService:storeLaneRecords (Sequence Diagram) 7-81
Figure 8-1 R6HighLevel (Use Case Diagram) 8-1
Figure 9-1 Location Form Showing Default Behavior 9-5
Figure 9-2 Location Form Showing New Values for Proximity 9-6
Figure 9-3 Location Form Showing Tabs for Feature 1 and Feature 2 9-6
Figure 9-4 Location Form Showing Feature 1 as an Exit and Feature 2 as a Road 9-7
Figure 9-5 Location Form Showing Error Message Due to Identical Intersecting Features 9-7
Figure 9-6 Location Form and Map Showing Default Behavior 9-8
Figure 9-7 Location Form and Map Showing Cross-Hair Marker and Feature Markers 9-8
Figure 9-8 Set Traffic Event Duplicate Comparison Rules Page 9-9
Figure 9-9 The Duplicate Event Proximity Value 9-9
Figure 9-10 Common2 (Class Diagram) 9-10
Figure 9-11 FlexLocationClasses (Class Diagram) 9-14
Figure 9-12 GUIFlexComponentsClasses (Class Diagram) 9-16
Figure 9-13 flex.shared.components:SpecifyLocation.updateIntFeatures (Sequence Diagram) 9-18
Figure 9-14 MapViewSpecificClasses (Class Diagram) 9-19
Figure 9-15 SpecifyLocation:jsUpdateMapGeoLocInfo (Sequence Diagram) 9-20
Figure 9-16 UtilityClasses2 (Class Diagram) 9-21
Figure 9-17 LocationRelatedProxyClasses (Class Diagram) 9-26
Figure 9-18 ProxyBasicTrafficEvent2:areLocationsTheSame (Sequence Diagram) 9-28

Figure 10-1 R6 ProvideDataToExternalSystems (Use Case Diagram) 10-2
Figure 10-2 Intranet Map User Login (Use Case Diagram) 10-5
Figure 11-1 Login to CHART 11-7
Figure 11-2 Login Screen 11-8
Figure 11-3 Login Screen Error 11-8
Figure 11-4 New Road Speed Sensor Legend 11-9
Figure 11-5 Turn on Road Speed Sensor layer 11-9
Figure 11-6 Filter TSS Display by Owning Organization and Speed Range 11-10
Figure 11-7 Intranet Map Configuration (Class Diagram) 11-11
Figure 11-8 Intranet Map User Validation (Class Diagram) 11-12
Figure 11-9 Intranet Map User Rights (Class Diagram) 11-13
Figure 11-10 Intranet Map Response Data (Class Diagram) 11-14
Figure 11-11 Intranet Map Client Request (Class Diagram) 11-15
Figure 11-12 Intranet Map Login (Sequence Diagram) 11-17
Figure 11-13 Intranet Map Keep Session Alive (Sequence Diagram) 11-19
Figure 11-14 WSUserManagerModuleClasses (Class Diagram) 11-19
Figure 11-15 WSDMSExportModuleClasses (Class Diagram) 11-22
Figure 11-16 DMSExportHandler:getDMSInventoryList (Sequence Diagram) 11-27
Figure 11-17 DMSRequestHandler:processRequest (Sequence Diagram) 11-28
Figure 11-18 DMSSubscriptionRequestHandler:processRequest (Sequence Diagram) 11-29
Figure 11-19 DMSExportHandler:getDMSStatusList (Sequence Diagram) 11-30
Figure 11-20 UserManagerRequestHandler:processRequest (Sequence Diagram) 11-31
Figure 11-21 UserManagerRequestHandler:handleException (Sequence Diagram) 11-32
Figure 12-1 R5ProvideDataToExternalSystems (Use Case Diagram) 12-2
Figure 12-2 R6ExportClientImportDataFromDataExporter (Use Case Diagram) 12-6
Figure 12-3 R6ExportClientPostRequestToMapApplication (Use Case Diagram) 12-9
Figure 12-4 Intranet Map Synchronization (Use Case Diagram) 12-10
Figure 13-1 WebServicesBaseClasses (Class Diagram) 13-13
Figure 13-2 WSTrafficEventExportModuleClasses (Class Diagram) 13-17
Figure 13-3 DataExporterUtilityClasses (Class Diagram) 13-21
Figure 13-4 DMSSubscriptionSupportClasses (Class Diagram) 13-23
Figure 13-5 WSDMSExportModuleClasses (Class Diagram) 13-26
Figure 13-6 DMSExportHandler:getDMSInventoryList (Sequence Diagram) 13-30
Figure 13-7 DMSExportHandler:getDMSStatusList (Sequence Diagram) 13-31
Figure 13-8 DMSExportHandler:initialize (Sequence Diagram) 13-32
Figure 13-9 DMSRequestHandler:handleExceptions (Sequence Diagram) 13-33
Figure 13-10 DMSRequestHandler:processRequest (Sequence Diagram) 13-34
Figure 13-11 DMSSubscription:doPush (Sequence Diagram) 13-35
Figure 13-12 DMSSubscriptionMgr:creation (Sequence Diagram) 13-36
Figure 13-13 DMSSubscriptionMgr:initialize (Sequence Diagram) 13-37
Figure 13-14 DMSSubscriptionMgr:modelObserverUpdate (Sequence Diagram) 13-38
Figure 13-15 DMSSubscriptionMgr:updateSubscription (Sequence Diagram) 13-39
Figure 13-16 DMSSubscriptionRequestHandler:processRequest (Sequence Diagram) 13-40
Figure 13-17 WSDMSExportModule:initialize (Sequence Diagram) 13-41
Figure 13-18 WSDMSExportModule:shutdown (Sequence Diagram) 13-42
Figure 13-19 ExportListenerModuleClasses (Class Diagram) 13-43
Figure 13-21 ExportListener:ProcessCameraInventory (Sequence Diagram) 13-45
Figure 13-22 ExportListener:ProcessCameraStatus (Sequence Diagram) 13-46
Figure 13-23 Intranet Map Exporter Synchonization (Class Diagram) 13-47
Figure 13-24 Intranet Map Exporter Synchronization (Sequence Diagram) 13-49

1 Introduction

1.1 Purpose

This document describes the design of the software for Release 6 of the CHART system. This build provides:

- EORS Road Closure Permit Integration: CHART R6 will include changes that help users more easily associate an EORS permit with a CHART planned closure event. The existing feature that allows the list of permit tracking numbers to be refreshed will be analyzed and corrected if needed to assure that it works properly. A type-ahead search feature will be added to allow the user to select a tracking number by typing in the first few characters of the number. A search by main route feature will be added to allow the user to locate permits based on the main route affected by the permit.
- Improve Map Lane Configuration Data: CHART R6 will enhance the ability to specify lane configurations for traffic events. A list of potential lane configurations will be presented to the user for selection based on an event's location utilizing lane configuration data from the State's mapping database. These GIS based lane configurations will be presented in addition to the existing list of the system's pre-defined lane configurations. The user will be permitted to edit any selected lane configuration to add and remove lanes as needed to more accurately depict the roadway at the location of the event. Edited lane configurations will be stored with the event, and also be stored as a candidate lane configuration for future events that occur near that location. The lane configuration feature will be re-designed to permit re-use of the lane configuration dialog from other applications on the CHART network, such as EORS.
- Map Integration Part 2 – Locate Between Features: CHART R6 will allow users to specify that a traffic event or device is located between two geographic features in addition to the existing at/past/prior a single geographic feature. The supported geographic feature types will remain the same as they are in CHART R5 (intersecting route, milepost, exit). The system will assign a point between the two selected features as the initial coordinates for the traffic event or device and will zoom the map on the location editor to the selected location. The user will then be able to click on the map to alter the coordinates as necessary. The method the system will use for the initial placement of the object was made during the requirements/JAD portion of the release.
- External TSSs on Intranet Map: As of R3B3 CHART imports many detectors and types of detectors from RITIS. However, CHART does not export this data for use by the Intranet Map, Internet Map, or the Public Web site. CHART R6 will allow suitably privileged operators to view detector data from RITIS on the Intranet Map.
- Camera Location Data Synchronization: CHART Mapping R5 will have the ability to view the Camera Locations. Camera information will be exported through the Exporter and Exporter Client in order for the user to view. This feature will work similar to Events, HARS and SHAZAM in CHART Mapping R4. The location information and description will be saved into CHARTWEB database. The location information and description will

be edited/created in the CHART GUI and will be transported via the CHART R6 Exporter to the CHARTWEB database, the Synchronization application expands its capability by importing CCTV information from the CHART System, and the camera data will be presented on R5 Intranet/Internet maps.

- Internet Explorer and Java upgrades: CHART R6 will include an upgrade of Internet Explorer to IE 8 and an upgrade to Java 6. There are no design changes for Internet Explorer and Java upgrades, and this feature is not considered any further in this design document.

1.2 Objectives

The main objective of this detailed design document is to provide software developers with a framework in which to implement the requirements identified in the CHART R6 Requirements document. A matrix mapping requirements to the design is presented in Section 15 (Mapping to Requirements).

1.3 Scope

This design is limited to Release 6 of the CHART System. It addresses both the design of the server components of CHART and the Graphical User Interface (GUI) components of CHART to support the new features being added. Design changes for the Intranet Map are included. This design does not include designs for components implemented in earlier releases of the CHART system.

1.4 Design Process

The design was created by capturing the requirements of the system in UML Use Case diagrams. Class diagrams were generated showing the high level objects that address the Use Cases. Sequence diagrams were generated to show how each piece of major functionality will be achieved. This process was iterative in nature – the creation of sequence diagrams sometimes caused re-engineering of the class diagrams, and vice versa.

1.5 Design Tools

The work products contained within this design will be extracted from the Tau Unified Modeling Language (UML) Suite design tool. Within this tool, the design will be contained in the CHART project, Release 6, Analysis phase and System Design phase. And also in the CHART Mapping project, Release 6, Analysis phase and System Design phase.

1.6 Work Products

The final R6 design consists of the following work products:

- Use Case diagrams that capture the requirements of the system
- Human-Machine Interface section which provides descriptions of the screens that are changing or being added in order to allow the user to perform the described uses.

- UML Class diagrams, showing the software objects which allow the system to accommodate the uses of the system described in the Use Case diagrams
- UML Sequence diagrams showing how the classes interact to accomplish major functions of the system
- Requirement Verification Traceability Matrix that shows how this design meets the documented requirements for this feature

This document incorporates the five features by providing a Use Cases and Detailed Design section for each feature. For instance, for EORS Integration, Use Cases are in Section 4, and Detailed Design (including Human-Machine Interface, Class Diagrams, and Sequence Diagrams) are in Section 5. Sections 6 & 7 cover Lane Configuration, and so on.

2 Architecture

The sections below discuss specific elements of the architecture and software components that are created, changed, or used in R6.

2.1 Network/Hardware

Except where noted, CHART R6 features do not impact the network or hardware architecture of the CHART System.

2.1.1 Lane Configuration Web Service

The following figure shows the network layout for the Lane Configuration feature. The Web Services shown on this diagram will be deployed with web services similar to those created for R5. The CHART Lane Editor Web Service will be deployed on the same server as the existing CHART GIS Web Service, and the Mapping Lane Config Web Service will be deployed on the same server as the existing Mapping GIS Web Service.

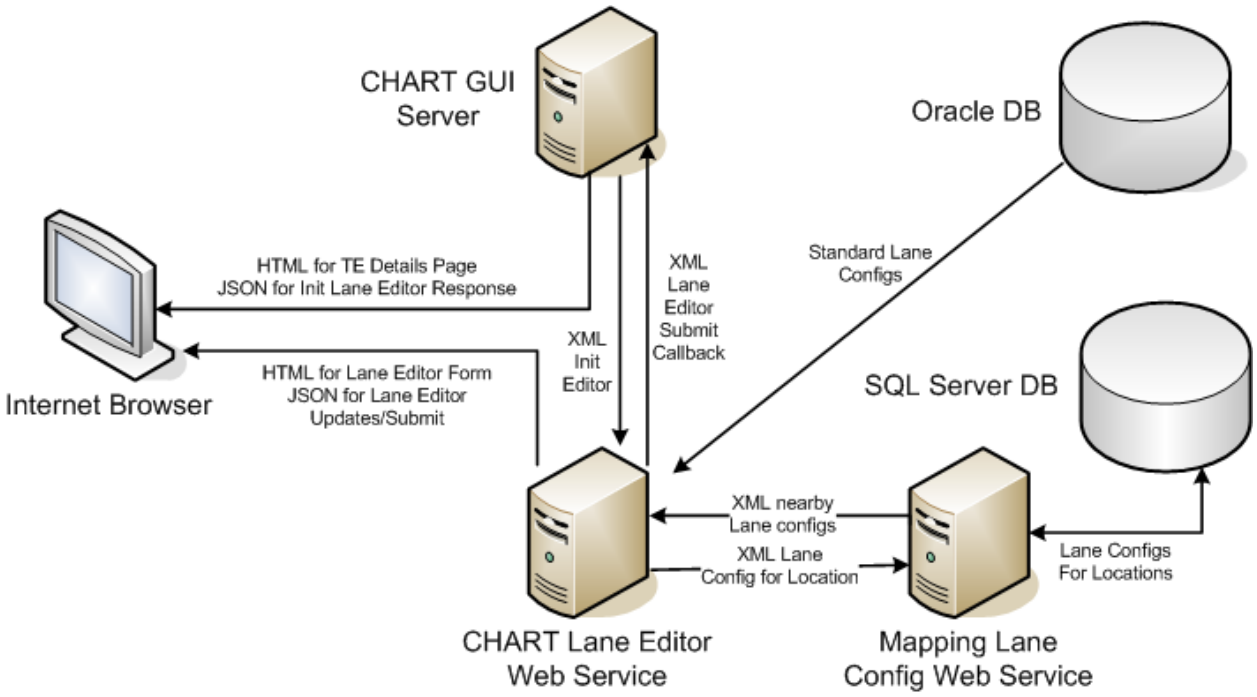


Figure 2-1 Lane Configuration Architecture Diagram

2.2 Software

CHART uses the Common Object Request Broker Architecture (CORBA) as the base architecture, with custom built software objects made available on the network allowing their data to be accessed via well defined CORBA interfaces. Communications to remote devices use

the Field Management Server (FMS) architecture. Newer external interfaces such as the User Management web service, Data Exporter, and GIS service employ a web services architecture combining an HTTP request/response structure to pass XML messages.

Except where noted in the subsections below, CHART R6 features do not impact the software architecture of the CHART System.

The Lane Configuration feature uses a web service interface to provide an open interface to lane editing functionality. This web service is utilized by the CHART GUI and the service is also available for use in the future by other SHA applications, such as EORS.

A User Management web service will be added to support user validation to clients external to CHART. The R5 Mapping Application will use this facility to validate Intranet Mapping users for the purpose of identifying what level of detail of External Detector data can be viewed by users logged into the R5 Intranet Mapping application.

2.2.1 COTS Products

2.2.1.1 CHART

CHART uses numerous COTS products for both run-time and development. New products being added in Release 6 are as follows:

Product Name	Description
bsn.autosuggest	The EORS integration feature uses version 2.1.3 of the bsn.autosuggest JavaScript code from brandspankingnew.net. This tool is freely available and is included as source code in the CHART GUI. It provides a simple JavaScript tool that can be associated with a text entry field. When the user types characters in the field, the tool waits until there has been no typing for a configurable number of milliseconds (to make sure the user is done typing) then places an AJAX call to a web server which can return suggested results that match the user entered text. The bsn.autosuggest tool then parses the results (XML or JSON) and displays a UI element that shows the user the suggestions and lets them select one of them by clicking on it. If a suggested element is selected by the user, a configurable JS method is invoked to allow the application to use the selected suggestion.

The following table contains existing COTS products that have not changed for R6:

Product Name	Description
Apache ActiveMQ	CHART uses this to connect to RITIS JMS queues

Product Name	Description
Apache Jakarta Ant	CHART uses Apache Jakarta Ant 1.6.5 to build CHART applications and deployment jars.
Apache Tomcat	CHART uses Apache Tomcat 6.0.29 as the GUI web server.
Apache XML-RPC	CHART uses the apache xmlrpc java library 3.1.2 protocol that uses XML over HTTP to implement remote procedure calls. The video Flash streaming "red button" ("kill switch") API uses XML over HTTP remote procedure calls.
Attention! CC	CHART uses Attention! CC Version 2.1 to provide notification services.
Attention! CC API	CHART uses Attention! CC API Version 2.1 to interface with Attention! CC.
Attention! NS	CHART uses Attention! NS Version 7.0 to provide notification services.
Bison/Flex	CHART uses Bison and Flex as part of the process of compiling binary macro files used for performing camera menu operations on Vicon Surveyor VFT cameras.
CoreTec Decoder Control	CHART uses a CoreTec supplied decoder control API for commanding CoreTec decoders.
Dialogic API	CHART uses the Dialogic API for sending and receiving Dual Tone Multi Frequency (DTMF) tones for HAR communications.
ESRI's ArcGIS Sever	CHART uses version 9.3 to serve maps over the Internet.
ESRI's MapObjects	CHART uses the Map Objects 2.4 for spatial algorithms.
Flex2 SDK	The CHART GUI will use the Flex2 SDK, version 3.1 to provide the Flex compiler, the standard Flex libraries, and examples for building Flex applications.
GIF89 Encoder	Utility classes that can create .gif files with optional animation. This utility is used for the creation of DMS True Display windows.
JAXB	CHART uses the jaxb java library to automate the tedious task of hand-coding field-by-field XML translation and validation for exported data.
JDOM	CHART uses JDOM b7 (beta-7) dated 2001-07-07. JDOM provides a way to represent an XML document for easy and efficient reading, manipulation, and writing.
JacORB	CHART uses a compiled, patched version of JacORB 2.2.4. The JacORB source code, including the patched code, is kept in the CHART source repository.
Java Run-Time (JRE)	CHART uses 1.6.0_21
JavaService	CHART uses JavaService to install the server side Java software components as Windows services.

Product Name	Description
JAXEN	CHART uses JAXEN 1.0-beta-8 dated 2002-01-09. The Jaxen project is a Java XPath Engine. Jaxen is a universal object model walker, capable of evaluating XPath expressions across multiple models.
JoeSNMP	CHART uses JoeSNMP version 0.2.6 dated 2001-11-11. JoeSNMP is a Java based implementation of the SNMP protocol. CHART uses for commanding iMPath MPEG-2 decoders and for communications with NTCIP DMSs.
JSON-simple	CHART uses the JSON-simple java library to encode/decode strings that use JSON (JavaScript Object Notation).
JTS	CHART uses the Java Topology Suite (JTS) version 1.8.0 for geographical utility classes.
Log4J	CHART uses the log4J version 1.2.15 for logging purposes.
NSIS	CHART uses the Nullsoft Scriptable Installation System (NSIS), version 2.20, as the server side installation package.
Nuance Text To Speech	For text-to-speech (TTS) conversion CHART uses a TTS engine that integrates with Microsoft Speech Application Programming Interface (MSSAPI), version 5.1. CHART uses Nuance Vocalizer 4.0 with Nuance SAPI 5.1 Integration for Nuance Vocalizer 4.0.
OpenLayers	The Integrated Map feature uses the Open Layers JavaScript API 2.8 (http://openlayers.org/) in order to render interactive maps within a web application without relying on vendor specific software. Open Layers is an open source product released under a BSD style license which can be found at (http://svn.openlayers.org/trunk/openlayers/license.txt).
Oracle	CHART uses Oracle 10.1.0.5 as its database and uses the Oracle 10G JDBC libraries (ojdbc1.4.jar) for all database transactions.
O'Reilly Servlet	Provides classes that allow the CHART GUI to handle file uploads via multi-part form submission.
Prototype Javascript Library	The CHART GUI uses the Prototype JavaScript library, version 1.6.0.3, a cross-browser compatible JavaScript library provides many features (including easy Ajax support).
SAXPath	CHART uses SAXPath 1.0-beta-6 dated 2001-09-27. SAXPath is an event-based API for XPath parsers, that is,

Product Name	Description
	for parsers which parse XPath expressions.
SQLServer JDBC Driver	CHART uses this driver to lookup GIS related data and also to store Location Aliases in SQL Server databases.
Velocity Template Engine	Provides classes that CHART GUI uses in order to create dynamic web pages using velocity templates, CHART uses Velocity version 1.6.1 and tools version 1.4.
Vicon V1500 API	CHART uses a Vicon supplied API for commanding the ViconV1500 CPU to switch video on the Vicon V1500 switch

2.2.1.2 Mapping

The ESRI COTS (MapObjects API v 2.4 and ArcSDE v 9.3) will be used. The ArcSDE command line is used to export the original (shape file) spatial table to the CHARTBG (ArcSDE enterprise SDE) database.

2.2.2 Deployment /Interface Compatibility

2.2.2.1 External Interfaces

This section describes the external interfaces being added in Release 6 of the CHART system and release 5 of the Mapping application.

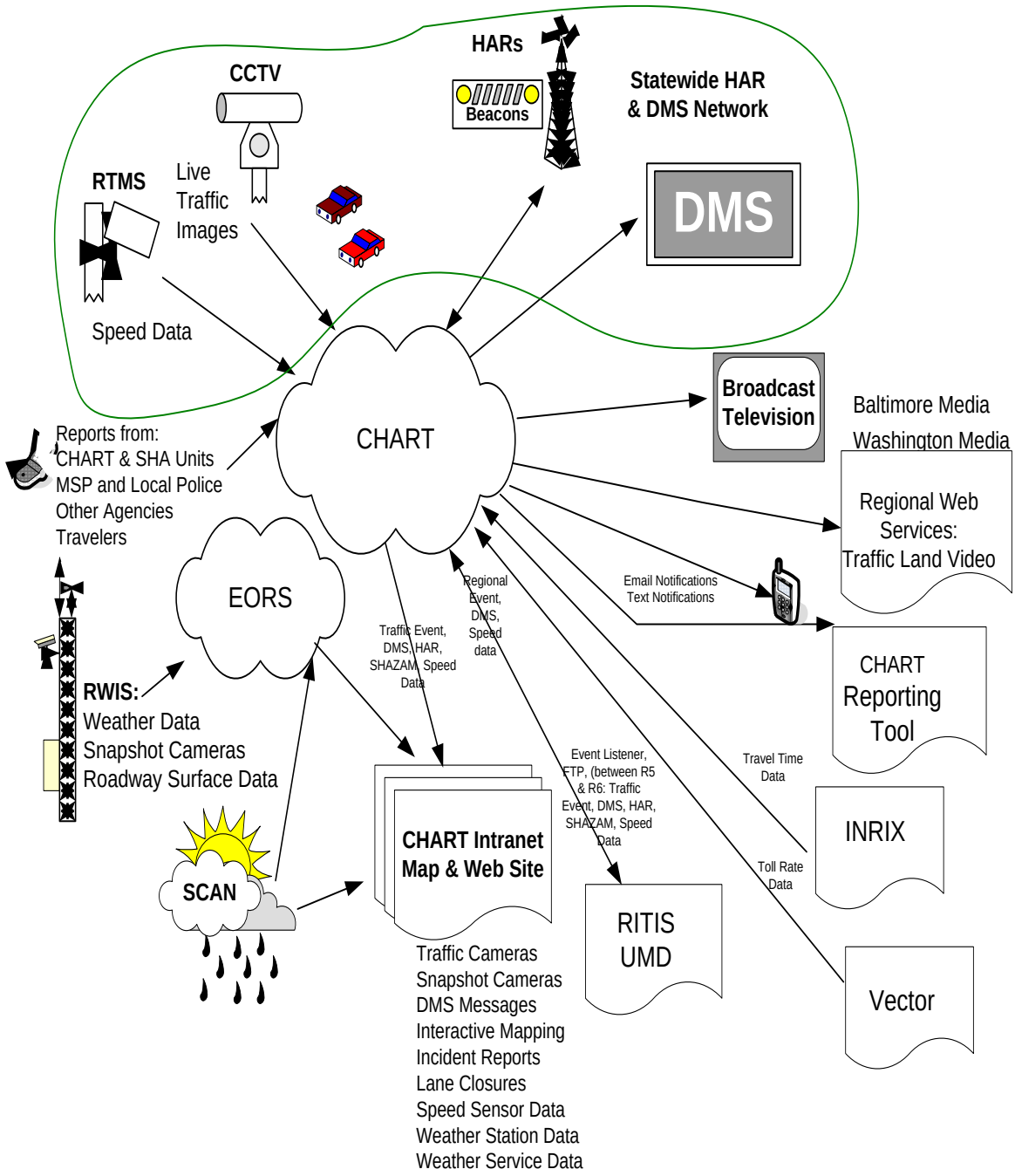


Figure 2-2 CHART and External Interfaces

The external interfaces included are:

1. The R6 Integrated map feature changes the ways that an object's location can be described. Thus it impacts the CHART Exporter and all clients of the CHART exporter by altering the exported XML that describes object locations.
2. The EORS Integration feature provides a list of EORS permits that are in the queued state as well as the active state. The SQLServer view used by the CHART R5 system to query EORS permits does not include this information. This view has been updated as part of the design for the CHART EORS Integration feature. Additionally, the view has been modified to allow the CHART system to query a last modified date for each EORS permit. This allows the CHART system to be much more efficient by querying only those permits that have changed since the last time synchronization was performed, rather than always sending all EORS permits across the network each time.
3. The Lane Configuration Editor uses a GIS web service API provided by the CHART Mapping team for retrieving lane configurations nearby traffic events. The lane configuration editor passes lane configurations created/edited by users to the GIS web service to allow it to augment the lane configuration data from its mapping database with lane configuration data as specified by users.
4. The External TSS feature provides for exporting of external TSSs, together with an indication for each TSS of the owning organization for that TSS. Consumers of this data can use the owning organization to filter what their users see. For instance, the Intranet Map application will use CHART user rights to determine the level of detail (summary or detailed volume/speed/occupancy data) that each user can see. RITIS can use owning organization to avoid bringing in detectors from CHART which was originally sourced by RITIS.
5. The External TSS feature adds a User Management web service which can be used by the R5 Mapping Application for authenticating Intranet Mapping Users, and could also be used in the future for other related applications, such as EORS.

Server and GUI deployment diagrams are shown in the next two figures.

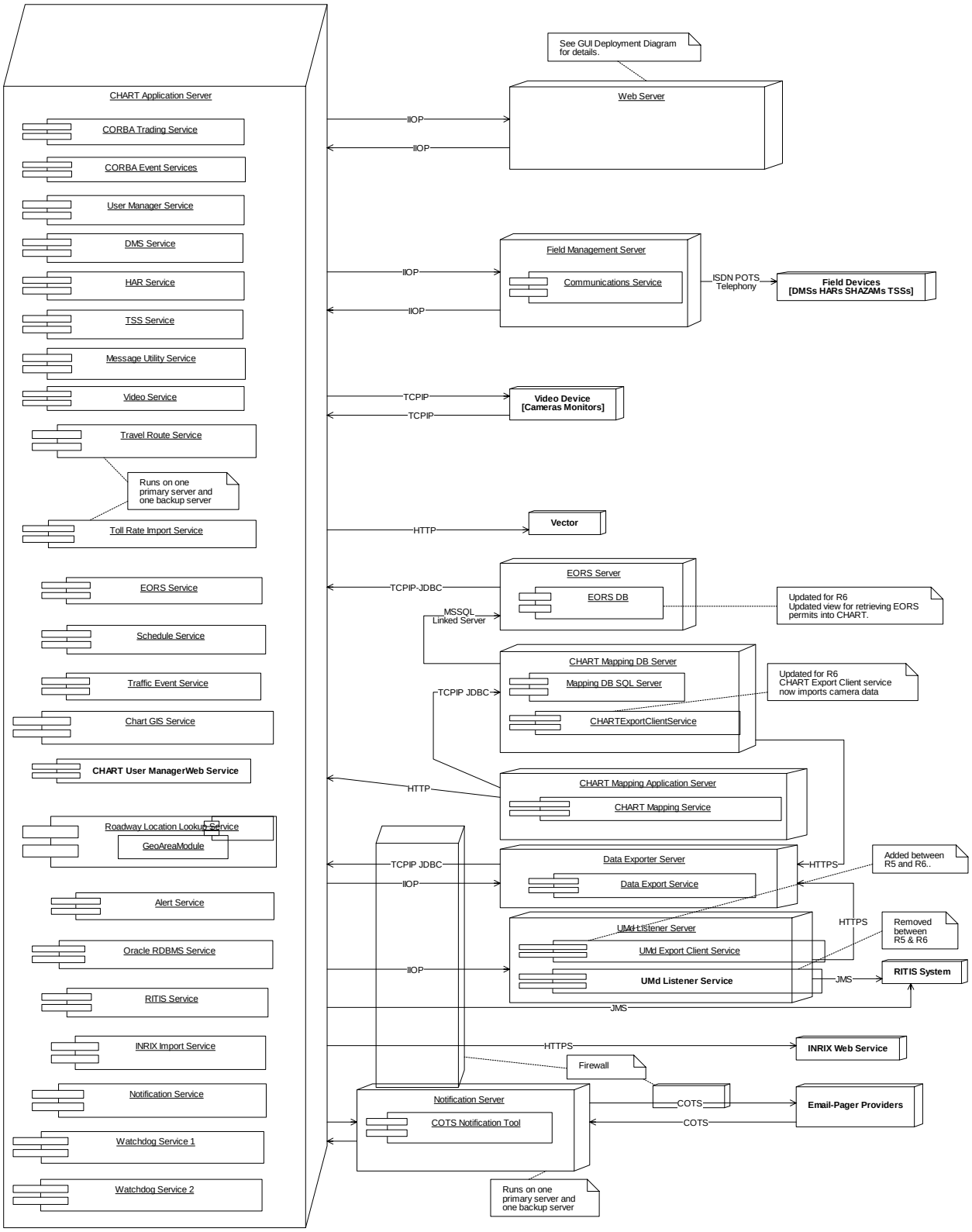


Figure 2-3 R6 Server Deployment

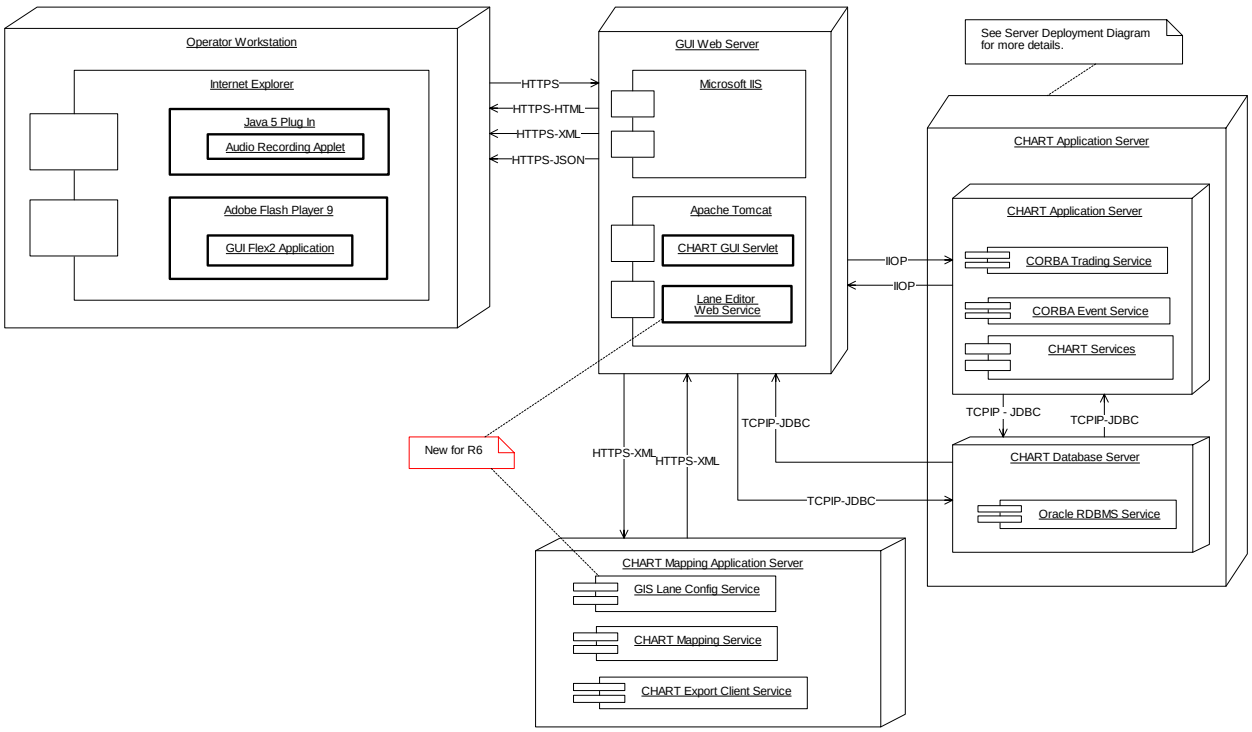


Figure 2-4 R6 GUI Deployment

2.2.2.2 Internal Interfaces

This section describes the internal interfaces being added or modified in Release 6 of the CHART system.

1. The Integrated Map feature utilizes the existing GUI interface. It changes only the forms that are used by operators to describe object locations. CHART services that utilize object locations and persist/depersist them in the CHART database are updated to utilize the new location data.
2. The Lane Configuration Editor is a web service that is used internally within CHART, and can be made available to other SHA applications, provided they are given a CHART private key and client ID to access the service. The CHART GUI interfaces to the Lane Editor Web service via XML over HTTP.

2.3 Security

This section describes the security being added or modified in Release 6 of the CHART system. Unless otherwise noted, features being added for R6 do not change security aspects of the CHART system.

1. The Lane Configuration Editor web service utilizes public/private key pairs and digital signatures to authenticate requests to initialize a lane editing session. This service will allow any client with the edit lane configuration functional right or the CHART2System functional right to perform this operation. The CHART GUI utilizes the

CHART2System functional right to make the call, but only if the currently logged in user possesses the Manage Traffic Events functional right. When the Lane Configuration Editor calls back to the CHART GUI when the user has submitted the form, the CHART GUI uses the Lane Configuration Editor public key and client ID to authenticate the request. It validates the signature provided with the request using the public key, and ensures the clientID has the Manage Traffic Events functional right.

2. The Intranet Mapping application will use CHART user rights, obtained from a new User Management Web Service, to determine which users can see what type of volume/speed/occupancy data (summary (speed range) or detailed (actual speed)).

3. Starting with deployment of CHART R6, if not before, it is anticipated that the CORBA interface to CHART from external systems (such as RITIS) will be completely retired. (It is anticipated that during the lifespan of CHART R5, RITIS will convert to using the CHART Data Exporter web service.)

2.4 Data

CHART R6 will be tested with the Oracle database patches that are available and will be deployed in the field at the time of CHART R6 deployment. The database patches may possibly be applied in the field before CHART R6 deployment.

2.4.1 Data Storage

The CHART System stores most of its data in an Oracle database. Additionally the Integrated Map feature adds the ability to store location aliases to the spatial SQL Server database. Some data is stored in flat files on the CHART servers. This section describes both types of data.

2.4.1.1 Database

2.4.1.1.1 Database Architecture

Except as noted CHART R6 features do not impact the overall architecture of the CHART database.

2.4.1.1.2 Logical Design

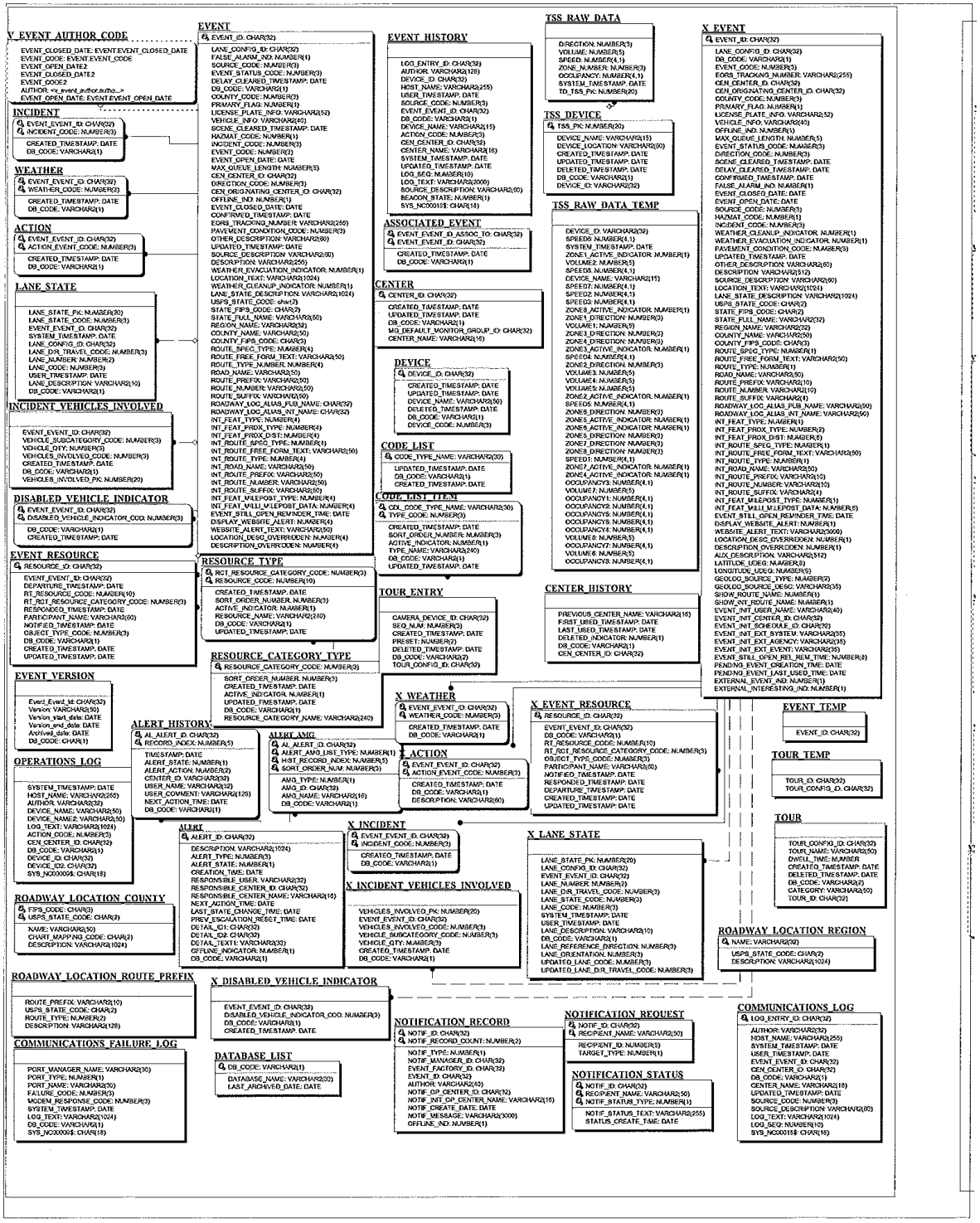
2.4.1.1.2.1 Entity Relationship Diagram (ERD)

For the Lane Configuration feature, the STANDARD_LANE_CONFIG table is no longer used by CHART. As a result, the STANDARD_LANE table will have a column name changed and a foreign key constraint removed.

The Integrated map feature modifies the OBJECT_LOCATION to support 3 new location proximity values (NONE, BETWEEN and FROM/TO), and the OBJECT_LOCATION table now supports a set of attributes to describe an optional second intersecting feature. These attribute default to null values and are only populated if the proximity for the feature location is BETWEEN or FROM/TO.

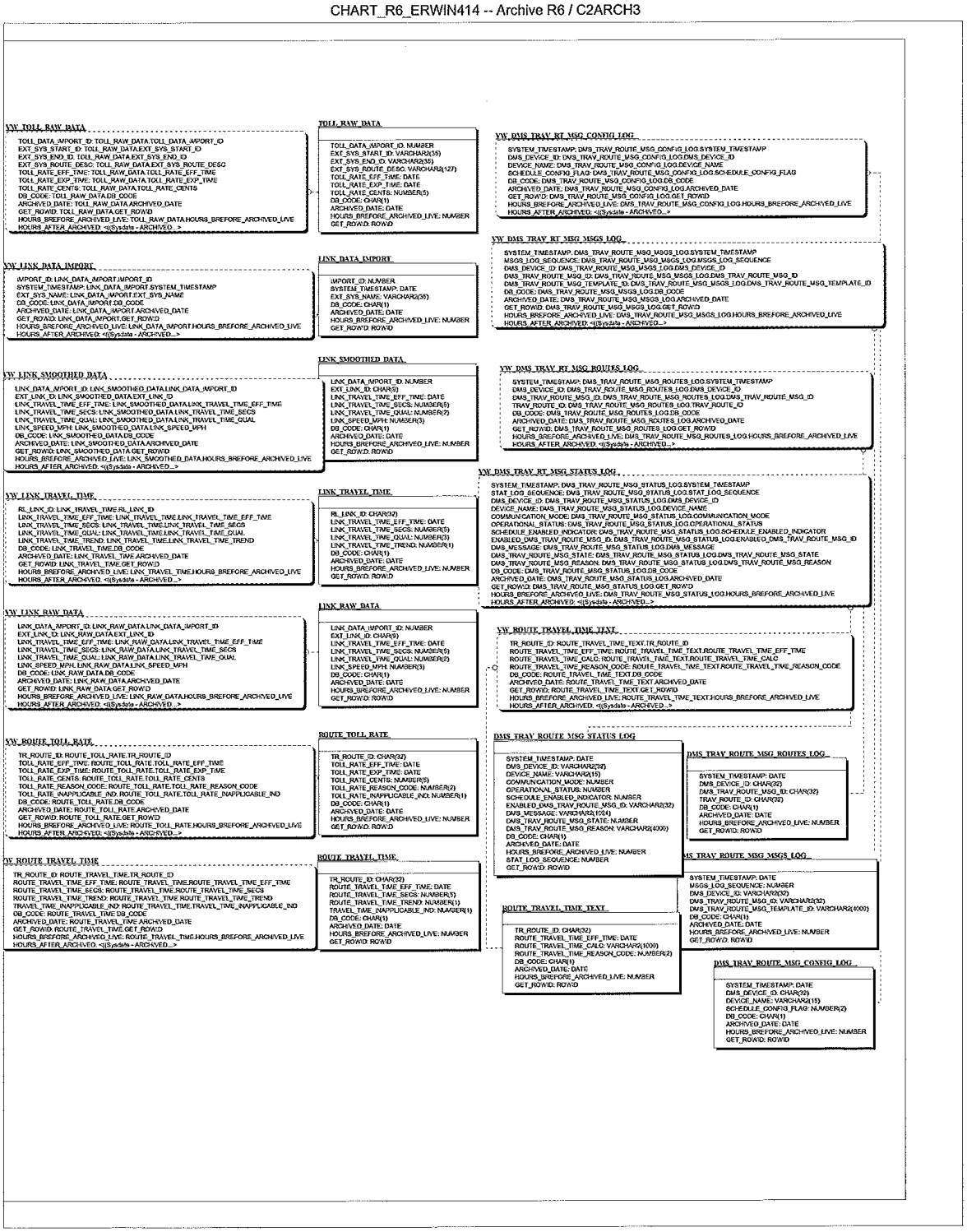
CHART Database entity relationship diagrams are shown below in the multiple pages of figures labeled collectively as Figure 2-5.

CHART_R6_ERWIN414 -- Archive R6 / C2ARCH3



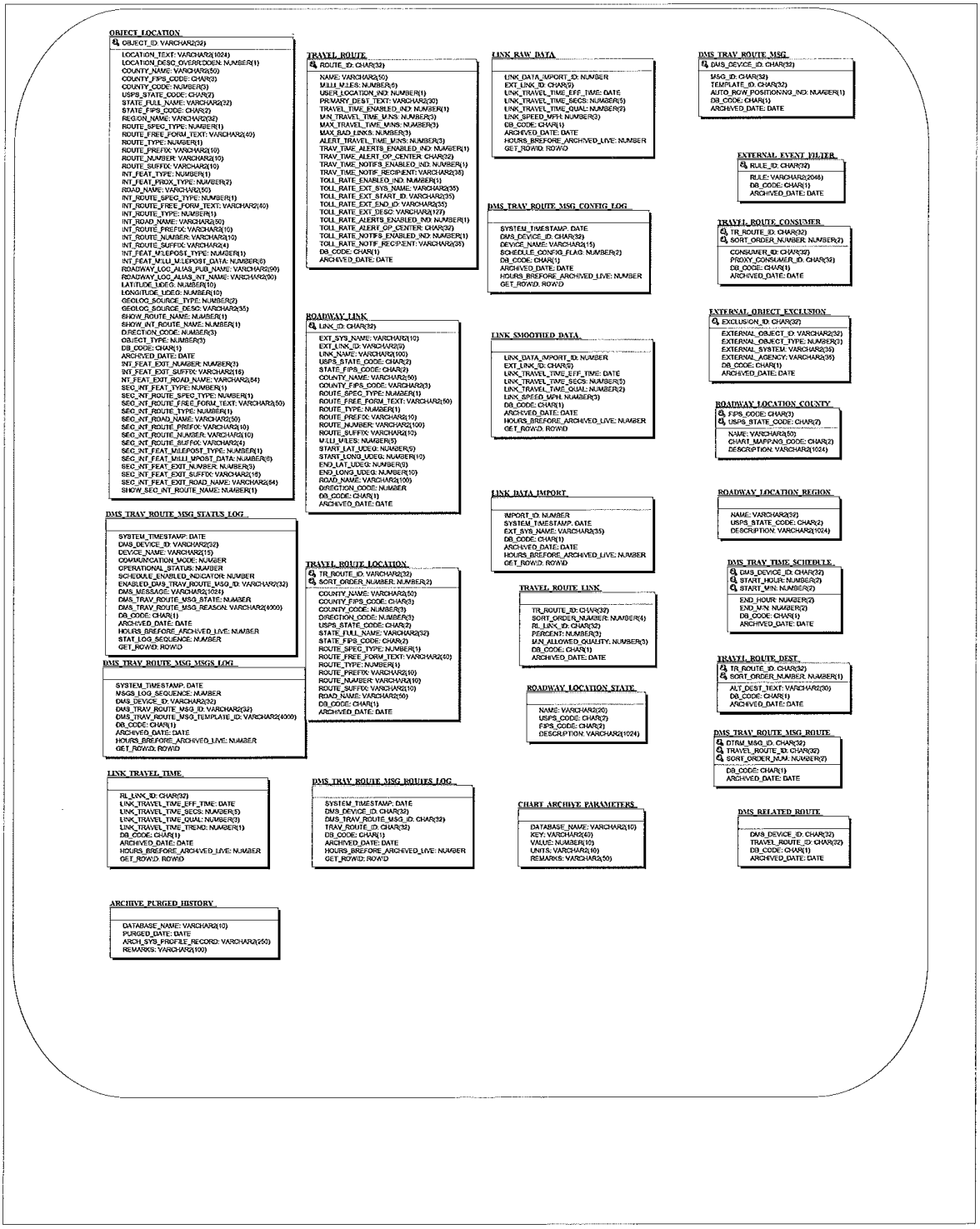
1, 1 / 1, 3 -- 2:04:41 PM , 9/21/2010

CHART_R6_ERWIN414 -- Archive R6 / C2ARCH3



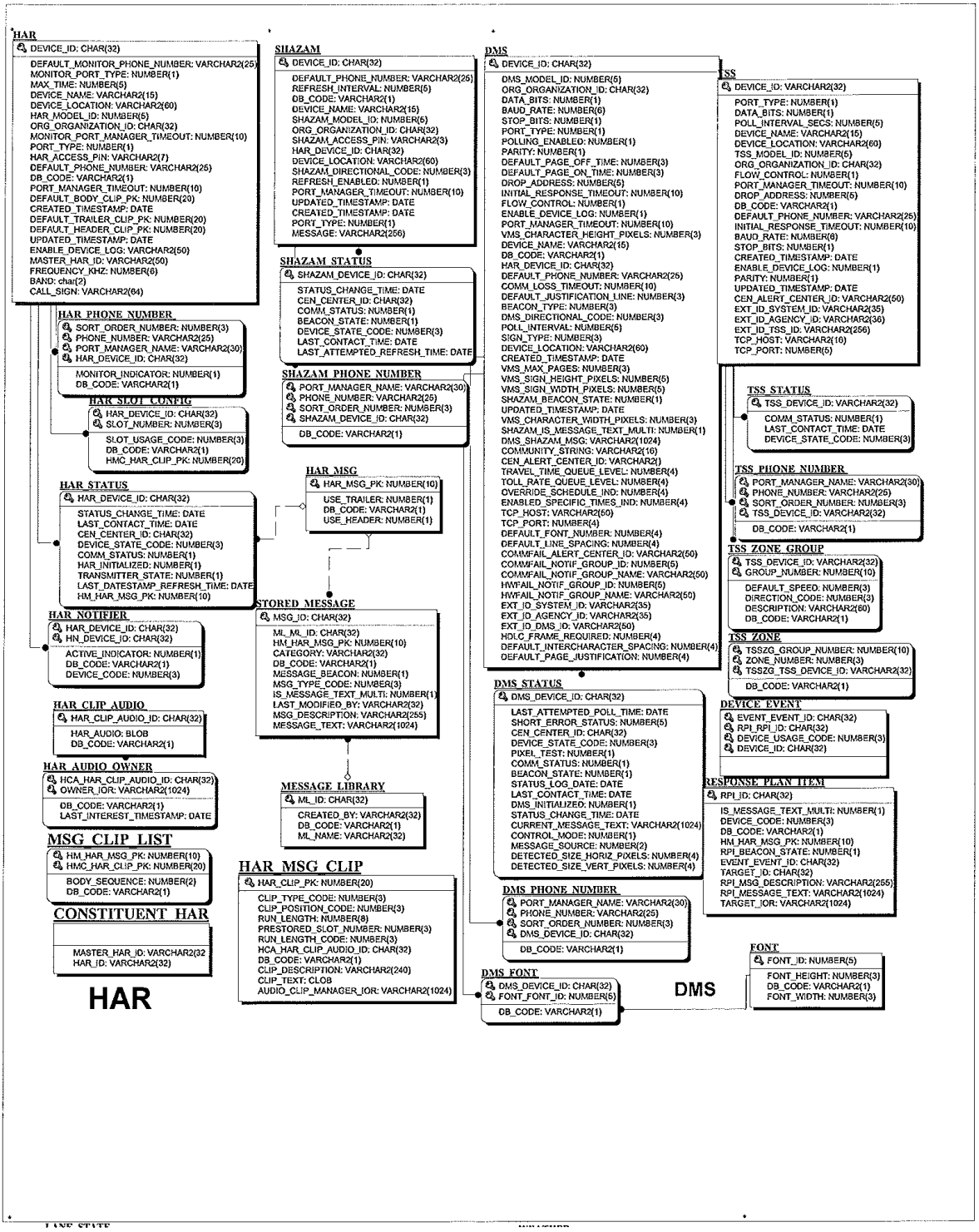
1, 2 / 1, 3 -- 2:04:41 PM , 9/21/2010

CHART_R6_ERWIN414 -- Archive R6 / C2ARCH3



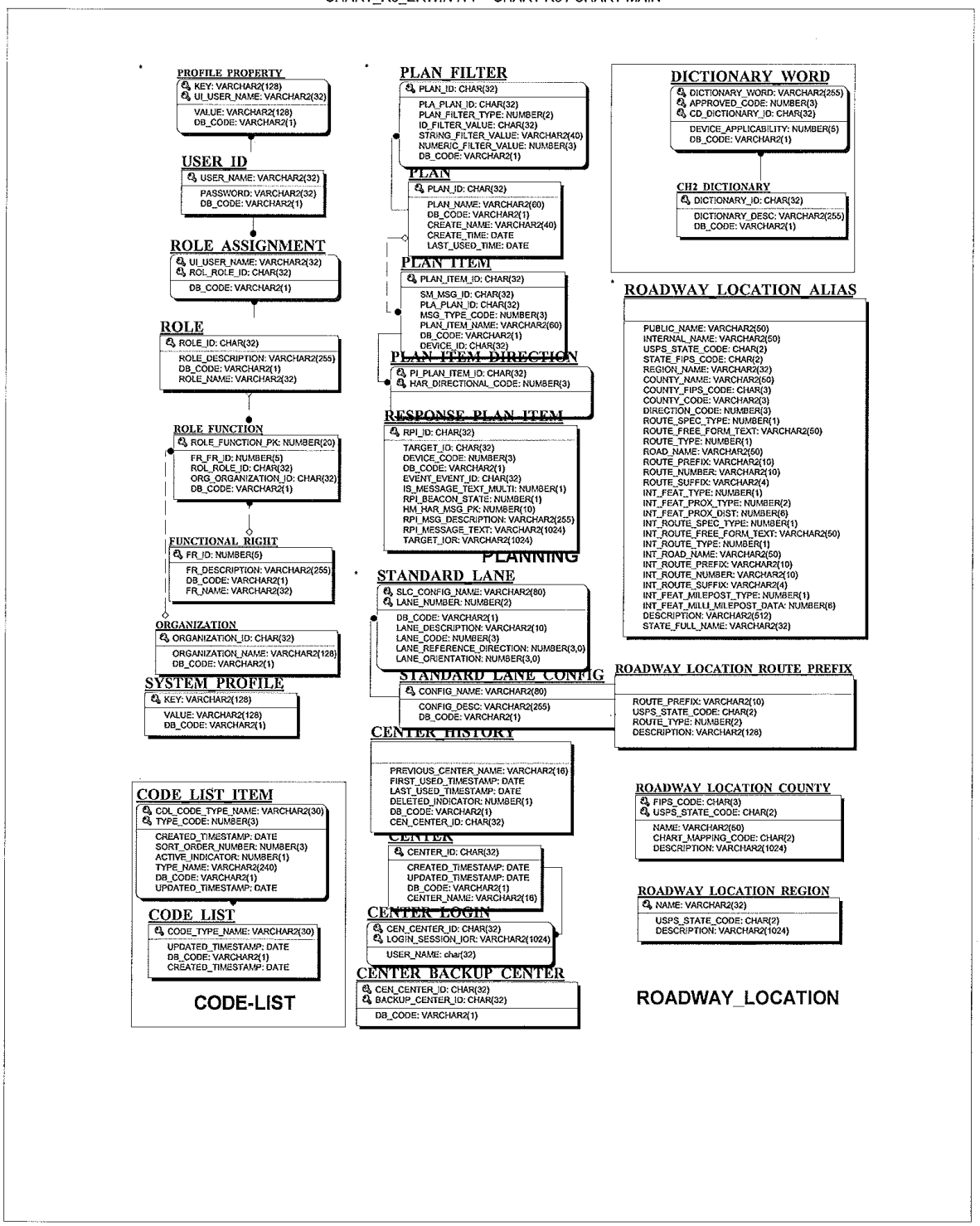
1, 3 / 1, 3 -- 2:04:41 PM , 9/21/2010

CHART_R6_ERWIN414 -- CHART R6 / CHART MAIN



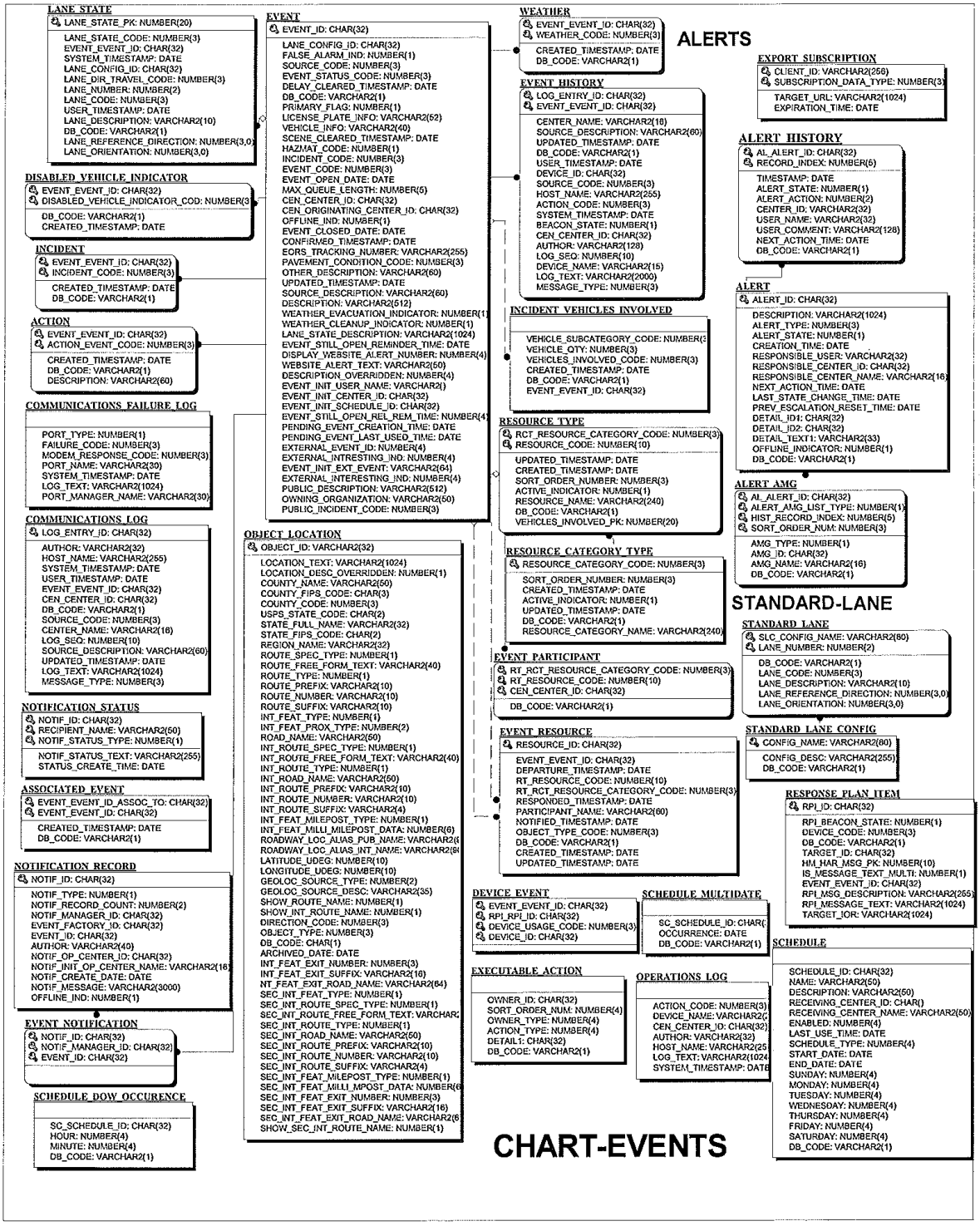
1, 1 / 2, 2 -- 2:04:50 PM , 9/21/2010

CHART_R6_ERWIN414 -- CHART R6 / CHART MAIN



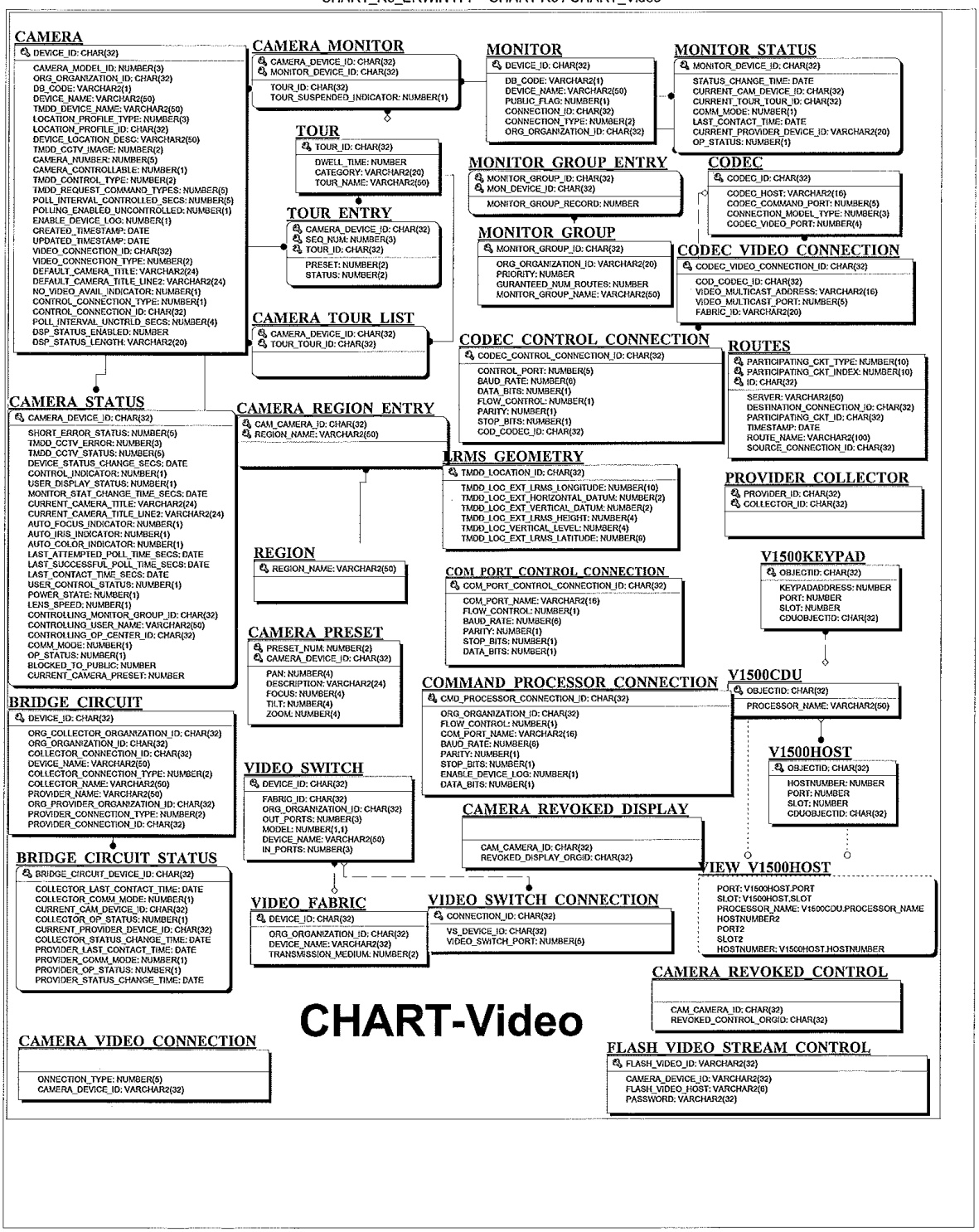
1, 2 / 2, 2 -- 2:04:50 PM , 9/21/2010

CHART_R6_ERWIN414 -- CHART R6 / CHART MAIN



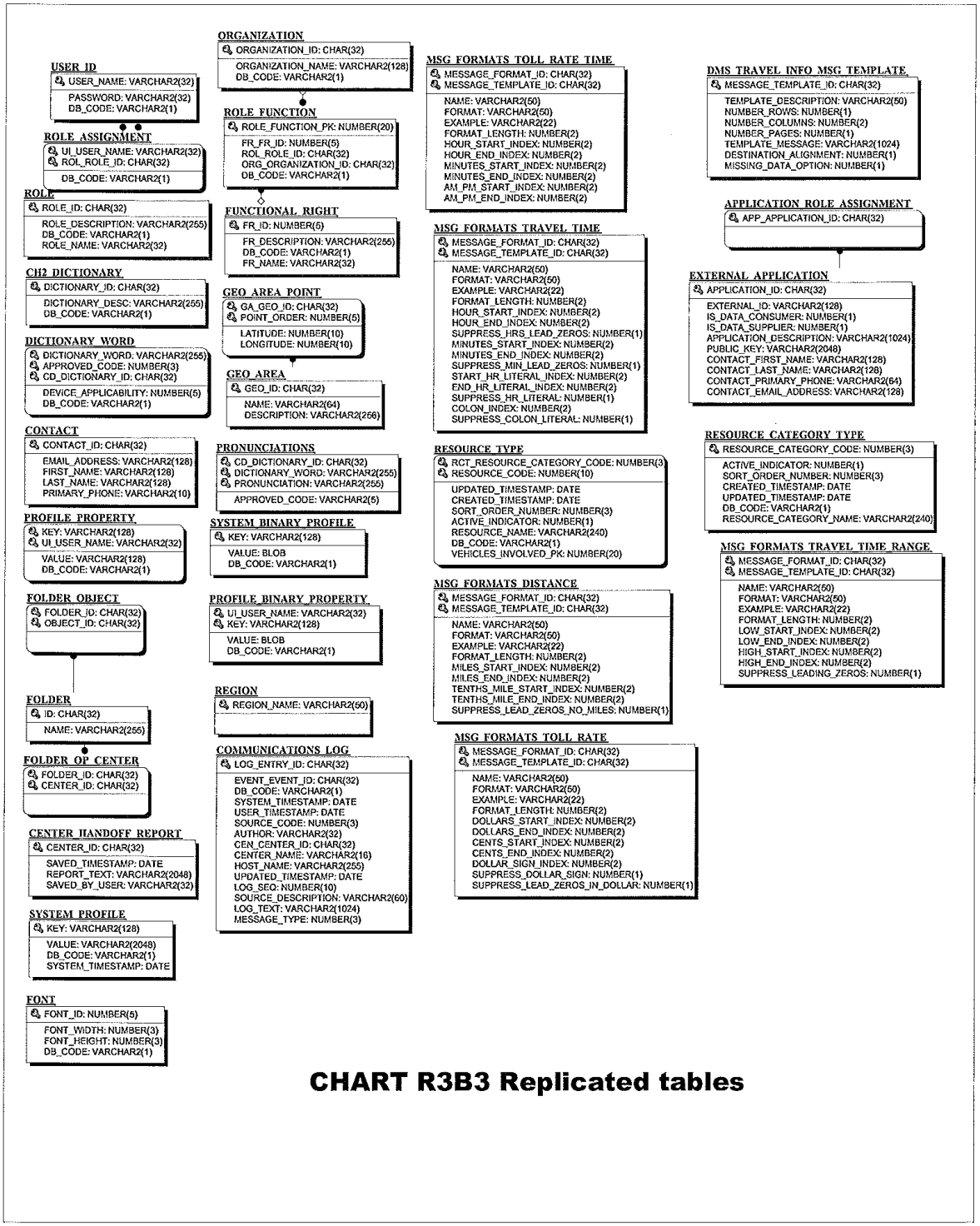
2, 1 / 2, 2 -- 2:04:50 PM , 9/21/2010

CHART_R6_ERWIN414 -- CHART R6 / CHART_Video



1, 1 / 1, 1 -- 2:04:58 PM , 9/21/2010

CHART_R6_ERWIN414 -- CHART R6 / Replicated Tables



1, 1 / 1, 1 -- 2:05:06 PM , 9/21/2010

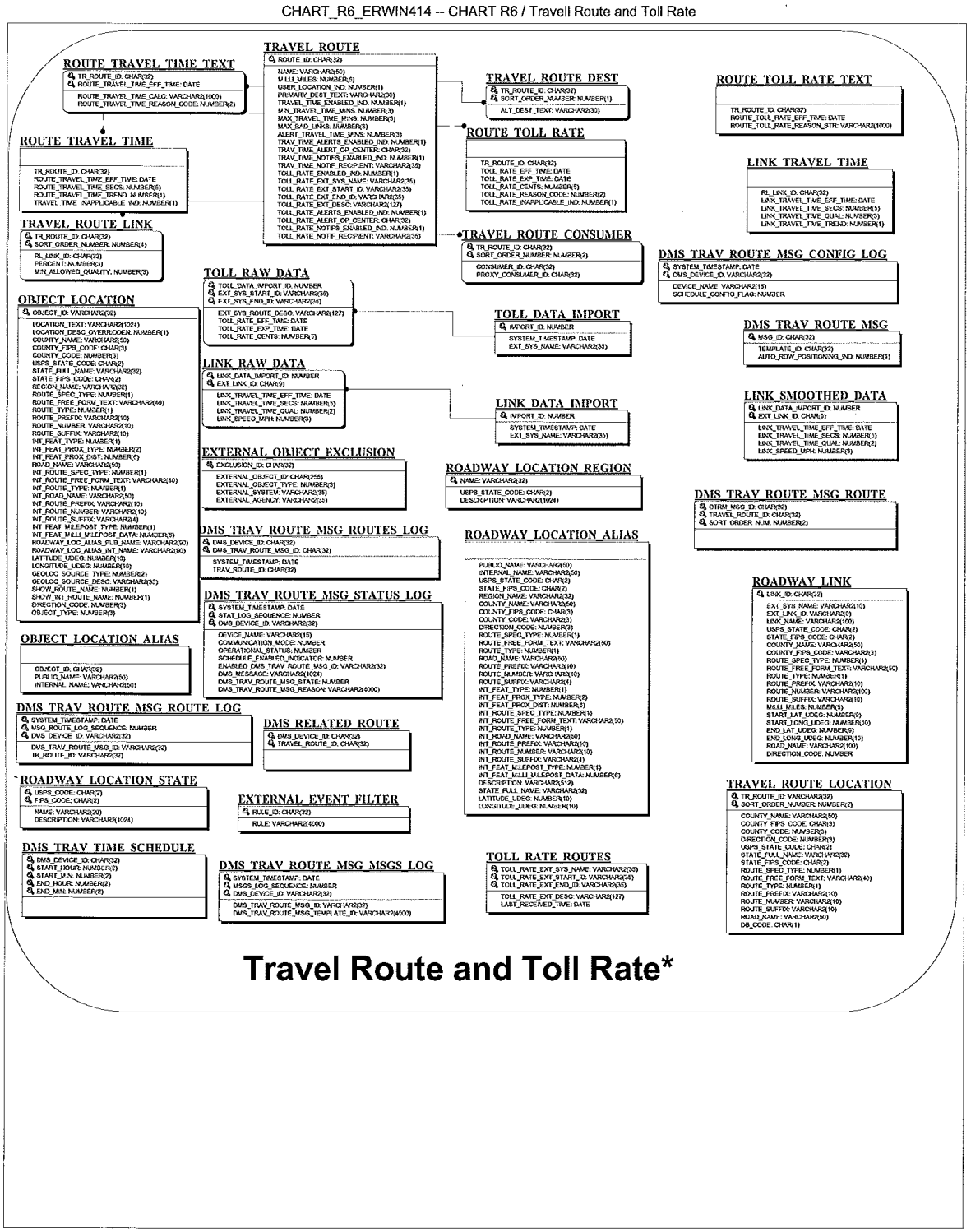


Figure 2-5 R6 ERD

2.4.1.1.2.2 Function to Entity Matrix Report

The Create, Retrieve, Update, Delete (CRUD) matrix cross-references business functions to entities and shows the use of the entities by those functions. This report will be generated as part of the CHART O&M Guide.

2.4.1.1.2.3 Table Definition Report –

In existing tables shown below:

- Deleted columns marked with a minus sign ("-")
- Modified columns marked with an asterisk ("*")
- New columns marked with a plus sign ("+")

2.4.1.1.2.3.1 Modifications for EORS Integration in the EORS Database

The EORS integration feature requires changes to the SQL Server view that is used to view the data in the EORS database in order to allow the CHART system to search permits that are in states other than ACTIVE. Additionally, the view has been modified to allow the CHART system to determine the last date/time that each permit was modified in order to allow for synchronization of the permits cached in the GUI without frequently pulling all EORS permit data across the network. The altered view definition is shown below under Logical Design and the changed portions are highlighted.

```
USE [eors]
GO
/***** Object: View [dbo].[chart_permit_view] Script Date: 08/23/2010 12:58:51 *****/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER OFF
GO

ALTER view [dbo].[chart_permit_view]

as

-- coalesce: ensure no null values are returned
select   top 100 percent
         coalesce(a.tracking_num,'None') 'trackingNo',
         coalesce
         (
            (
               select chart_id
               from permit_type
               where permit_type = a.permit_type
            ),0
         ) 'pt_type',
         coalesce
         (
            (
               select full_txt
               from permit_type
               where permit_type = a.permit_type
```

```
      ), 'None'
   ) 'pt_typeDecsription',
   coalesce(a.date_submitted,0) 'submissionDate',
   (
      case
         when b.status = '85D7D8D9-79C2-11D4-8E52-0000F878B97A' -- Active
         then 1
         else 0
      end
   ) 'isActive',
   (
      case
         when b.status = 'acdab1b9-f43b-4174-ba35-f47b1bbd144a' -- Queued
         then 1
         else 0
      end
   ) 'isQueued',
   coalesce
   (
      (
         select max(mod_date)
         from permit_history
            where permit = a.permit
      ),0
   ) 'lastModifiedDate',
   coalesce(a.permit_num,'None') 'contractNumber',
   coalesce
   (
      (
         select name
         from county
         where county = a.county_from
      ), 'None'
   ) 'startCountyName',
   coalesce
   (
      (
         select name
         from county
         where county = a.county_to
      ), 'None'
   ) 'endCountyName',
   coalesce
   (
      (
         select chart_id
         from route_type
         where route_type = a.route_type
      ), 0
   ) 'rt_type',
   coalesce
   (
      (
         select full_txt
         from route_type
         where route_type = a.route_type
      ), 'None'
   ) 'rt_typeDecsription',
   coalesce(a.route_num,0) 'routeNo',
   coalesce(c.intersection,'None') 'location',
   coalesce(c.exit_num,'None') 'exitNo',
   coalesce(c.limits_from,'None') 'startOfWorkZone',
```

```
         coalesce(c.limits_to,'None') 'endOfWorkZone',
         coalesce(cast(c.mp_from as float),0) 'startingMilePost',
         coalesce(cast(c.mp_to as float),0) 'endingMilePost',
         coalesce
         (
            (
               select chart_id
               from direction
               where direction = c.direction
            ), 0
         ) 'directionOfTravel',
         coalesce(cast(g.lanes as int),0) 'lanesClosed',
         coalesce(a.adc_map_num,'None') 'mapNo',
         coalesce(a.adc_map_coord,'None') 'coordinates',
         coalesce
         (
            (
               select tcs_num
               from tcs_number
               where tcs_number = a.tcs_number
            ), 'None'
         ) 'shaTCS',
         coalesce
         (
            (
               select reason_text
               from reason
               where reason = a.reason
            ), 'None'
         ) 'workDesc',
         coalesce(a.date_from + c.time_from,0) 'startDateTime',
         coalesce(a.date_to + c.time_to,0) 'endDateTime',
         coalesce(cast((d.sun|d.mon|d.tue|d.wed|d.thu|d.fri|d.sat) as tinyint),0)
'daysOfWeek',
         coalesce(e.contact,'None') 'fieldContactName',
         coalesce(e.phone,'None') 'sha_contactPhoneNo',
         coalesce(e.pager,'None') 'sha_contactPagerNo',
         coalesce(e.call_num,'None') 'sha_contactCellPhoneNo',
         coalesce(e.fax,'None') 'sha_contactFaxNo',
         coalesce(f.name,'None') 'permiteeName',
         coalesce(f.address,'None') 'permiteeAddress',
         coalesce(f.contact,'None') 'permiteeContactPerson',
         coalesce(f.phone,'None') 'permitee_contactPhoneNo',
         coalesce(f.pager,'None') 'permitee_contactPagerNo',
         coalesce(f.call_num,'None') 'permitee_contactCellPhoneNo',
         coalesce(f.fax,'None') 'permitee_contactFaxNo',
         (case when b.approval_name is null then 0 else 1
end)'districtApprovalGranted',
         coalesce(b.approval_name,'None') 'districtApprovalName',
         coalesce(b.approval_date,0) 'districtApprovalDate'

from permit a
inner join permit_history b
   on a.permit = b.permit
   and b.mod_date =
   (
      -- get most recent entry in history table
      select max(mod_date)
      from permit_history
      where permit = a.permit
   )
inner join lane_schedule c
```

```
   on a.permit = c.permit
   and c.lane_schedule =
   (
      /* get a single lane */
      select top 1 lane_schedule
      from lane_schedule
      where permit = c.permit
   )
inner join
(
   -- cross-tab existing permit dates into bitmasks
   select top 100 percent
      permit,
      max
      (
         (
            case when datepart(weekday,date) = 1 then 0x01 else 0 end
         )
      ) 'sun',
      max
      (
         (
            case when datepart(weekday,date) = 2 then 0x02 else 0 end
         )
      ) 'mon',
      max
      (
         (
            case when datepart(weekday,date) = 3 then 0x04 else 0 end
         )
      ) 'tue',
      max
      (
         (
            case when datepart(weekday,date) = 4 then 0x08 else 0 end
         )
      ) 'wed',
      max
      (
         (
            case when datepart(weekday,date) = 5 then 0x10 else 0 end
         )
      ) 'thu',
      max
      (
         (
            case when datepart(weekday,date) = 6 then 0x20 else 0 end
         )
      ) 'fri',
      max
      (
         (
            case when datepart(weekday,date) = 7 then 0x40 else 0 end
         )
      ) 'sat'
   from permit_date
   group by permit
) d
   on a.permit = d.permit
-- left outer join: contact_info may be null
left outer join contact_info e
   on a.contact_info = e.contact_info
```

```
-- left outer join: permitee_info may be null
left outer join contact_info f
   on a.permitee_info = f.contact_info
inner join
(
   -- get 'lanes' as a bitfield
   select top 100 percent
      a.permit,
      -- must sum distinct; no aggregate bitwise OR function
      sum(distinct b.chart_id) 'lanes'
   from lane_schedule a
   inner join lane b
      on a.lane = b.lane
   group by permit
) g
   on a.permit = g.permit
-- filter to avoid excess rows; calling process does not filter
where (b.status = '85D7D8D9-79C2-11D4-8E52-0000F878B97A' -- Active
OR b.status='85D7D8D8-79C2-11D4-8E52-0000F878B97A'
OR b.status='85D7D8D7-79C2-11D4-8E52-0000F878B97A'
OR b.status='ACDAB1B9-F43B-4174-BA35-F47B1BBD144A')

order by a.tracking_num
```

2.4.1.1.2.3.2 Modifications for Lane Configuration in the CHART R6 Live Database

The Lane Configuration feature includes the following changes with regard to the CHART R6 live database:

The STANDARD_LANE_CONFIG table is no longer used by CHART. The STANDARD_LANE table will have a column name changed and a foreign key constraint removed. The changes are effected by the following commands:

```
ALTER TABLE STANDARD_LANE DROP CONSTRAINT SL_SLC_FK;

ALTER TABLE STANDARD_LANE  RENAME COLUMN SLC_CONFIG_NAME TO CONFIG_NAME;

DROP TABLE STANDARD_LANE_CONFIG;
```

2.4.1.1.2.3.3 Modifications for Lane Configuration in the R5 Mapping Database

The Lane Configuration feature uses a CENTERLINE_UNIV2008_M table in the Intranet Mapping database. The table is defined as:

```
[dbo].[CENTERLINE_UNIV2008_M](
        [OBJECTID] [int] NULL,
        [COUNTY] [numeric](30, 11) NULL,
        [MUN_SORT] [numeric](30, 11) NULL,
        [ID_PREFIX] [varchar](2) NULL,
        [ID_RTE_NO] [numeric](30, 11) NULL,
        [MP_SUFFIX] [varchar](2) NULL,
        [ID_MP] [numeric](24, 11) NULL,
        [MAIN_LINE] [numeric](30, 11) NULL,
        [PROPOSED] [numeric](30, 11) NULL,
```

```
[AREA_TYPE] [numeric](30, 11) NULL,
[STATE_MUN_] [numeric](30, 11) NULL,
[ACCESS_TYP] [numeric](30, 11) NULL,
[LT_OUT_SHL] [numeric](30, 11) NULL,
[LT_OUT_S_1] [numeric](30, 11) NULL,
[LT_ROADWAY] [numeric](30, 11) NULL,
[LT_ROADW_1] [numeric](30, 11) NULL,
[LT_THRU_LA] [numeric](30, 11) NULL,
[LT_IN_SHLD] [numeric](30, 11) NULL,
[LT_IN_SH_1] [numeric](30, 11) NULL,
[MEDIAN_WD] [numeric](30, 11) NULL,
[MEDIAN_TY] [numeric](30, 11) NULL,
[RT_IN_SHLD] [numeric](30, 11) NULL,
[RT_IN_SH_1] [numeric](30, 11) NULL,
[RT_ROADWAY] [numeric](30, 11) NULL,
[RT_ROADW_1] [numeric](30, 11) NULL,
[RT_THRU_LA] [numeric](30, 11) NULL,
[RT_OUT_SHL] [numeric](30, 11) NULL,
[RT_OUT_S_1] [numeric](30, 11) NULL,
[STATE_MP] [numeric](24, 11) NULL,
[ROAD_NAME] [varchar](40) NULL,
[AADT_STATI] [varchar](12) NULL,
[ADMIN_CLAS] [numeric](30, 11) NULL,
[TRANS_YR] [varchar](4) NULL,
[STMP_SEQ] [numeric](30, 11) NULL,
[AADT_YR] [varchar](4) NULL,
[DISTRICT] [varchar](1) NULL,
[MAINT_SHOP] [numeric](30, 11) NULL,
[SPEED_LIMI] [numeric](30, 11) NULL,
[LRS_ID] [varchar](12) NULL,
[RURURB] [numeric](30, 11) NULL,
[URB_TECH_A] [numeric](30, 11) NULL,
[NAAQS_AREA] [numeric](30, 11) NULL,
[URBAN_AREA] [numeric](30, 11) NULL,
[FUNC_CL] [numeric](30, 11) NULL,
[NHS_CODE] [numeric](30, 11) NULL,
[UNBUILT_FA] [numeric](30, 11) NULL,
[GOVT_CONTR] [numeric](30, 11) NULL,
[SPECIAL_SY] [numeric](30, 11) NULL,
[FACILITY_T] [numeric](30, 11) NULL,
[TRUCKS] [numeric](30, 11) NULL,
[TOLL] [numeric](30, 11) NULL,
[SECTION_LE] [numeric](24, 11) NULL,
[AADT] [numeric](30, 11) NULL,
[THROUGH_LA] [numeric](30, 11) NULL,
[SYS_MEDIAN] [numeric](30, 11) NULL,
[ROUGHNESS] [numeric](30, 11) NULL,
[PSR] [numeric](24, 11) NULL,
[ROUTE_CONT] [numeric](30, 11) NULL,
[REVERSIBLE] [numeric](30, 11) NULL,
[LT_OUT_SID] [numeric](30, 11) NULL,
[LT_IN_SIDE] [numeric](30, 11) NULL,
[RT_OUT_SID] [numeric](30, 11) NULL,
[LT_OUT_AUX] [numeric](30, 11) NULL,
[LT_IN_AUX_] [numeric](30, 11) NULL,
```

```
[RT_IN_AUX_] [numeric](30, 11) NULL,
[RT_OUT_AUX] [numeric](30, 11) NULL,
[LT_OUT_A_1] [numeric](30, 11) NULL,
[LT_IN_AUX1] [numeric](30, 11) NULL,
[RT_IN_AUX1] [numeric](30, 11) NULL,
[RT_OUT_A_1] [numeric](30, 11) NULL,
[LT_OUT_A_2] [numeric](30, 11) NULL,
[LT_IN_AU_1] [numeric](30, 11) NULL,
[RT_IN_AU_1] [numeric](30, 11) NULL,
[RT_OUT_A_2] [numeric](30, 11) NULL,
[IS_DONUT] [numeric](30, 11) NULL,
[IS_SAMPLE] [numeric](30, 11) NULL,
[GENERIC_FI] [varchar](254) NULL,
[GENERIC__1] [varchar](254) NULL,
[GENERIC__2] [varchar](254) NULL,
[GENERIC__3] [varchar](254) NULL,
[GENERIC__4] [varchar](254) NULL,
[FITM] [smallint] NULL,
[HURRICANE_] [smallint] NULL,
[FLOOD_EVAC] [smallint] NULL,
[SNOW_EMERG] [smallint] NULL,
[BICYCLE] [smallint] NULL,
[CIVIL_WAR] [smallint] NULL,
[SCENIC_BYW] [smallint] NULL,
[I38] [numeric](30, 11) NULL,
[I39] [numeric](30, 11) NULL,
[I40] [numeric](30, 11) NULL,
[I41] [numeric](30, 11) NULL,
[I42] [numeric](30, 11) NULL,
[I43] [numeric](30, 11) NULL,
[I44] [numeric](30, 11) NULL,
[I45] [numeric](30, 11) NULL,
[I46] [numeric](30, 11) NULL,
[NLFID] [varchar](13) NULL,
[RT_IN_SIDE] [numeric](30, 11) NULL,
[IS_HOV] [numeric](30, 11) NULL,
[MAP_NUMBER] [varchar](12) NULL,
[RT_IN_SH_2] [numeric](30, 11) NULL,
[RT_OUT_S_2] [numeric](30, 11) NULL,
[LT_IN_SH_2] [numeric](30, 11) NULL,
[LT_OUT_S_2] [numeric](30, 11) NULL,
[LT_LA_TY] [numeric](30, 11) NULL,
[RT_LA_TY] [numeric](30, 11) NULL,
[RECORD_STA] [numeric](30, 11) NULL,
[NHS_ID] [numeric](30, 11) NULL,
[DATE_LAST_] [datetime] NULL,
[EXIT_NUMBE] [varchar](10) NULL,
[RAMP_NUMBE] [numeric](30, 11) NULL,
[ASSOC_ID_P] [varchar](10) NULL,
[VALIDATION] [varchar](250) NULL,
[HISTORY] [varchar](254) NULL,
[IS_COMMENT] [numeric](30, 11) NULL,
[IS_ADMIN] [numeric](30, 11) NULL,
[SAMPLE_NUM] [varchar](12) NULL,
[ROUTEID] [varchar](32) NULL,
```

```
[end_mp] [numeric](30, 11) NULL,
[Shape_Leng] [numeric](30, 11) NULL,
[Shape] [int] NULL,
[LnConfigSource] [varchar] (32) NULL,
[DATE_Ln_POST] [datetime] NULL
```

2.4.1.1.2.3.4 Tables for Modified for Integrated Map in the CHART R6 Live Database

The Integrated map feature includes the following changes with regard to database.

- The OBJECT_LOCATION table now supports 3 new location proximity values (NONE, BETWEEN and FROM/TO).
- The OBJECT_LOCATION table now supports a set of attributes to describe an optional second intersecting feature. These attribute default to null values and are only populated if the proximity for the feature location is BETWEEN or FROM/TO.

OBJECT_LOCATION

```
OBJECT_ID                                VARCHAR2(32) NOT NULL
LOCATION_TEXT                            VARCHAR2(1024)
LOCATION_DESC_OVERRIDDEN                 NUMBER(1)
COUNTY_NAME                              VARCHAR2(50)
COUNTY_FIPS_CODE                         CHAR(3)
COUNTY_CODE                              VARCHAR2(3)
USPS_STATE_CODE                          CHAR(2)
STATE_FULL_NAME                          VARCHAR2(32)
STATE_FIPS_CODE                          CHAR(2)
REGION_NAME                              VARCHAR2(32)
ROUTE_SPEC_TYPE                          NUMBER(1)
ROUTE_FREE_FORM_TEXT                     VARCHAR2(50)
ROUTE_TYPE                               NUMBER(1)
ROUTE_PREFIX                             VARCHAR2(10)
ROUTE_NUMBER                             VARCHAR2(10)
ROUTE_SUFFIX                             VARCHAR2(10)
INT_FEAT_TYPE                            NUMBER(1)
INT_FEAT_PROX_TYPE                       NUMBER(2)
- INT_FEAT_PROX_DIST                     NUMBER(6)
ROAD_NAME                                VARCHAR2(50)
INT_ROUTE_SPEC_TYPE                      NUMBER(1)
INT_ROUTE_FREE_FORM_TEXT                 VARCHAR2(50)
INT_ROUTE_TYPE                           NUMBER(1)
INT_ROAD_NAME                            VARCHAR2(50)
INT_ROUTE_PREFIX                         VARCHAR2(10)
INT_ROUTE_NUMBER                         VARCHAR2(10)
INT_ROUTE_SUFFIX                         VARCHAR2(4)
INT_FEAT_MILEPOST_TYPE                   NUMBER(1)
INT_FEAT_MILLI_MPOST_DATA                NUMBER(6)
ROADWAY_LOC_ALIAS_PUB_NAME               VARCHAR2(90)
ROADWAY_LOC_ALIAS_INT_NAME               VARCHAR2(90)
LATITUDE_UDEG                            NUMBER(10)
```

```
LONGITUDE_UDEG                              NUMBER(10)
GEOLOC_SOURCE_TYPE                          NUMBER(2)
GEOLOC_SOURCE_DESC                          VARCHAR2(35)
SHOW_ROUTE_NAME                             NUMBER(1)
SHOW_INT_ROUTE_NAME                         NUMBER(1)
DIRECTION_CODE                              NUMBER(3)
OBJECT_TYPE                                 NUMBER(3)
INT_FEAT_EXIT_NUMBER                        NUMBER(3)
INT_FEAT_EXIT_SUFFIX                        VARCHAR2(16)
INT_FEAT_EXIT_ROAD_NAME                     VARCHAR2(64)
+ SEC_INT_FEAT_TYPE                         NUMBER(1)
+ SEC_INT_ROUTE_SPEC_TYPE                   NUMBER(1)
+ SEC_INT_ROUTE_FREE_FORM_TEXT              VARCHAR(50)
+ SEC_INT_ROUTE_TYPE                        NUMBER(1)
+ SEC_INT_ROAD_NAME                         VARCHAR(50)
+ SEC_INT_ROUTE_PREFIX                      VARCHAR(10)
+ SEC_INT_ROUTE_NUMBER                      VARCHAR(10)
+ SEC_INT_ROUTE_SUFFIX                      VARCHAR(4)
+ SEC_INT_FEAT_MILEPOST_TYPE                NUMBER(1)
+ SEC_INT_FEAT_MILLI_MILEPOST_DATA          NUMBER(6)
+ SEC_INT_FEAT_EXIT_NUMBER                  NUMBER(3)
+ SEC_INT_FEAT_EXIT_SUFFIX                  VARCHAR(16)
+ SEC_INT_FEAT_EXIT_ROAD_NAME               VARCHAR(64)
+ SHOW_SEC_INT_ROUTE_NAME                   NUMBER(1)
```

2.4.1.1.2.3.5 Modifications for Camera Locations in the R5 Mapping Database

The following changes are necessary for CCTV location synchronization in the R5 Mapping Application database:

In the Camera table, additional columns such as “county”, “region”, “latitude”, “longitude” and “block to public” are added to the camera table. The data will be generated by the exporter client instead of EORS. Instead of using “cameraID” as the identifier, a unique 32 characters device id which is used by CHART will be used as the identifier. Both “enc_id” and ”G_CCTVID” fields are dropped. Also the existing “Camera” table in the CHARTWeb Database is updated to accommodate CCTV information in R6. Following two stored procedures were added to the CHARTWeb Database in R6 “camera_add_update_config_wl” and “camera_add_update_status_wl”.

In the Encoder_trafficCams table, instead of using “enc_id” as the identifier, a unique 32 characters device id which is used by CHART will be used as the identifier.

Data for the G_CCTV table is generated by the synchronization application instead of the Device Editor.

2.4.1.1.2.4 PL/SQL Module Definition and Database Trigger Reports

There are no new PL/SQL modules for the Data Exporter, Maintenance GUI and the Integrated Map. For the Intranet Map the following new stored procedures are added.

Intranet Map

`+ camera_add_update_config_wl` – To add or update a Camera device configuration.
`+ camera_add_update_status_wl` - To add or update a Camera's status.

2.4.1.1.2.5 Database Size Estimate - provides size estimate of current design

There are no changes for any significance to the database size for R6.

2.4.1.1.2.6 Data Distribution

There are no changes to data distribution for R6.

2.4.1.1.2.7 Database Replication

There are no changes to database replication for R6.

2.4.1.1.2.8 Archival Migration

There are no changes to archival migration for R6.

2.4.1.1.2.9 Database Failover Strategy

There are no changes to the database failover strategy for R6.

2.4.1.1.2.10 Reports

No reports will be added or updated for R6. Starting with R6, the CHART reporting function has been transferred to University of Maryland.

2.4.1.2 CHART Flat Files

The following describes the use of flat files in CHART.

2.4.1.2.1 Service Registration Files

There are no new Java services and therefore no new service registration files for CHART R6.

2.4.1.2.2 Service Property Files

Except as noted, there are no new service property files for CHART R6.

The CHART Lane Configuration Editor web service obtains configuration values from a properties file that it reads from its WEB-INF directory. The CHART User Management web service obtains configuration values from a properties file that it reads from its WEB-INF directory.

2.4.1.2.3 GUI Property Files

There are only minor updates to the GUI properties file in its WEB-INF directory for CHART R6.

2.4.1.2.4 Arbitration Queue Storage Files

There are no changes to Arbitration Queue Storage Files for R6.

2.4.1.2.5 Device Logs

There are no changes to Device Log Files for R6.

2.4.1.2.6 Traffic Sensor Raw Data Logs

There are no changes to Traffic Sensor Raw Data Log Files for R6.

2.4.1.2.7 Service Process Logs

All CHART services write to a process log, used to provide a historical record of activity undertaken by the services. These logs are occasionally referenced by software engineering personnel to diagnose a problem or reconstruct a sequence of events leading to a particular anomalous situation. These logs are automatically deleted by the system after a set period of time defined by the service's properties file, so they do not accumulate infinitely. These files are stored in the individual service directories and are named by the service name and date, plus a "`.txt`" extension. These logs are typically read only by software engineering personnel. Except where noted, there are no changes for service process logs for R6 features.

- The Lane Configuration Editor web service maintains a rolling application log file in its WEB-INF directory.
- The User Management web service maintains a rolling application log file in its WEB-INF directory.

2.4.1.2.8 Service Error Logs

All CHART services write to an error log, used to provide detail on certain errors encountered by the services. Most messages, including most errors, are captured by the CHART software and written to the process logs, but certain messages (typically produced by the Java Virtual Machine itself, by COTS, or DLLs) cannot be captured by CHART Software and instead are captured in these "catch-all" logs. Errors stored in these logs are typically problems resulting from a bad installation; once the system is up and running, errors rarely appear in these error logs. Debugging information from the JacORB COTS, which is not usually indicative of errors, can routinely be found in these error logs, as well. These log files can be reviewed by software engineering personnel to diagnose an installation problem or other type of problem. These logs are automatically deleted by the system after a set period of time defined by the service's properties file, so they do not accumulate infinitely. These files are stored in the individual service directories and are named by the service name and date, plus an ".err" extension. These logs are typically read only by software engineering personnel. Except where noted, there are no changes for service error logs for R6 features.

- The Lane Configuration Editor web service redirects stderr to an application error log file in its WEB-INF directory.
- The User Management web service redirects stderr to an application error log file in its WEB-INF directory.

2.4.1.2.9 GUI Process Logs

Like the CHART background services, the CHART GUI service also writes to a process log file, used to provide a historical record of activity undertaken by the process. These GUI process logs are occasionally referenced by software engineering personnel to diagnose a problem or reconstruct a sequence of events leading to a particular anomalous situation. These logs are automatically deleted by the system after a set period of time defined by the GUI service's properties file, so they do not accumulate infinitely. These files are stored in the `chartlite/LogFiles/` directory under the `WebApps/` directory in the Apache Tomcat installation area. They are named by the service name ("`chartlite`") and date, plus a "`.txt`" extension. These logs are typically read only by software engineering personnel. Additional log files written by the Apache Tomcat system itself are stored in the `log/` directory in the Apache Tomcat installation area.

- R6 GUI changes do not change the way the GUI process logs operate.

2.4.1.2.10 FMS Port Configuration Files

The CHART Communications Services read a Port Configuration file, typically named `PortConfig.xml`, upon startup, which indicates which ports are to be used by the service and how they are to be initialized. A Port Configuration Utility is provided which allows for addition, removal of ports and editing of initialization parameters. As indicated by the extension, these files are in XML format. This means these files are hand-editable, although the Port Configuration Utility allows for safer, more controlled editing. The Port Configuration files are typically modified only by software engineers or telecommunications engineers.

- There are no changes to this section for the any of the R6 features.

2.4.1.2.11 Watchdog Configuration Files

The watchdog configuration files are updated to provide monitoring and restarting of the CHART Lane Configuration Editor Web Service and User Management Web Service.

2.4.2 Database Design

Changes made to the CHART database design for Release 6 features are described below.

2.4.2.1 Lane Configuration

2.4.2.1.1 CHART

The STANDARD_LANE_CONFIG database table is no longer needed by the CHART software and will be removed. This table was previously used to associate a config name with a config description, however config name is no longer needed. The STANDARD_LANE table will be used to obtain the lane config description from the renamed CONFIG_NAME column.

2.4.2.1.2 Intranet Mapping

The UNIV2008 database table, provided by the MDOT HISD organization, will be used for the Mapping Lane Configuration Web Service. This table contains information for road segments

within Maryland, including lane types, lane directions, and the number of lanes on each side of the road. This table is copied into the existing CHARTBG database using the name CENTERLINE_UNIV2008_M, and is modified as follows.

- To allow the table to queried using the ESRI MapObjects tool, the Z value must be removed from the ***Shape*** column using the ESRI ArcMap EtGeoWizard. The Z values are not needed so this work around is not an issue.
- The type of the ***Shape*** column (LINE M) must be changed to LINE because the LINE M type is not allowed in the Spatial Object query.
- Columns are added to store the lane configuration time stamp value and Lane configuration source.

The source table, UNIV2008, will not be altered.

2.4.2.2 Integrated Map

The Integrated map feature includes the following changes with regard to database.

- The OBJECT_LOCATION table now supports 3 new location proximity values (NONE, BETWEEN and FROM/TO).
- The OBJECT_LOCATION table now supports a set of attributes to describe an optional second intersecting feature. These attribute default to null values and are only populated if the proximity for the feature location is BETWEEN or FROM/TO.

2.4.2.3 Mapping

In the Camera table, additional columns such as “county”, “region”, “latitude”, “longitude” and “block to public” are added to the camera table. Both “enc_id” and ”G_CCTVID” fields are dropped. Also the existing “Camera” table in the CHARTWeb Database is updated to accommodate CCTV information in R6. Two stored procedures were added: “camera_add_update_config_wl” and “camera_add_update_status_wl”.

2.4.2.4 Archiving - Changes

The CHART Archive database stores data from the CHART operational system as part of a permanent archive. The CHART Archive database design is a copy of the CHART operational system for those tables containing system, alert, traveler information messages and their underlying data, and event log information. In addition, the CHART Archive database stores detector data. In CHART R6 archiving process will archive event location from the OBJECT_LOCATION table for CHART events.

- R6 changes will not invalidate pre-R6 events and devices in the archive database., and it will enhance the object location archive process for CHART external events. As a part of this release, old event data of existing event and external event tables will use to populate the archived OBJECT_LOCATION table.

2.4.3 Error Processing

There are no changes for error processing for CHART R6.

3 Key Design Concepts

3.1 EORS Road Closure Permit Integration

The EORS integration feature simplifies the process of associating a CHART Planned Roadway Closure Event with an EORS permit by replacing the drop down list of currently active EORS permit tracking numbers with more advanced user interface tools. The user is presented with a text entry field where they can type search criteria. If the user types part of an EORS permit tracking number, the system will suggest permits for the user to select from. If the user does not see the permit they are looking for they may click a search button which will search all active and queued permits from the EORS system. The searching feature is not limited to tracking numbers. Rather, it will search the following fields of each permit: permit tracking number, start county name, end county name, permit type, route location, route type, route number, work order description, permittee name, contract number and days of week. Thus a user may search for permits using search text such as "Bridge Montgomery Monday". Returned permits are ranked according to their relevance to the search text specified.

The EORS integration feature also improves the usability of the EORS permit searching feature by removing reliance on the user to manually update the list of EORS permits that are available for searching. This is accomplished by adding a synchronization feature that keeps the list of EORS permits available in the GUI current without requiring all data for all permits to be passed across the LAN on every refresh. This is accomplished by determining when each EORS permit was last modified in the EORS system and making this data available to the CHART system via the SQL Server view.

3.2 Improved Map Lane Configuration Data

3.2.1 CHART

The existing lane configuration features of the CHART GUI are enhanced and moved to the Lane Configuration Editor web service to make these features reusable in other applications. The CHART GUI accesses the lane configuration features via the web service's http / XML interface. In addition to providing an http / XML interface, the web service also serves the lane editor web page (HTML) and supports requests used by the web page as the user interacts with the form. The requests performed during user interaction with the form use AJAX techniques to perform these requests asynchronously to prevent form refreshes and provide a better user experience. All AJAX requests elicit a response that uses JSON (JavaScript Object Notation) to allow the responses to be more easily handled via JavaScript on the lane editor web page.

The general processing flow for editing the lane configuration and status for a traffic event in CHART is as follows:

- The CHART GUI calls the Lane Configuration Editor web service via http / XML to initialize an editing session. The initialization provides information to the service to allow it to call the Mapping Lane Configuration web service to find lane configurations nearby the traffic event location. The initialization process also allows an existing lane

configuration and status to be passed in to initialize the editor when the user is editing an existing lane configuration. The web service returns a unique identifier for the lane editing session.

- The CHART GUI creates a popup window and sets its URL to the address of the Lane Configuration Editor web service's request to view a lane editor, passing the lane editing session ID as a parameter. The response is HTML for the lane editor web page.
- The user interacts with the lane editor web page. The lane editor allows the user to choose a configuration from a list of configurations that includes the existing standard lane configurations in addition to lane configurations for the roadway that are nearby the traffic event location. The user can set the state (open, closed, unknown) for each lane and can also set the travel direction for each lane. Lanes can be removed from the lane configuration (thereby removing the need for the "non-existent" lane state) and lanes can be added. As the user interacts with the form, the web page sends requests to the Lane Configuration Editor web service which keeps track of the lane configuration and status and generates the corresponding lane image. The Lane Configuration Editor web service responds with JSON that allows the web page to update its image as well as the image map that is used to allow the user to select lanes and perform actions on them.
- The user submits the lane editor form. When the user clicks the submit button, a standard HTTP submit is not used, and instead a request is sent to the Lane Configuration Editor web service via AJAX. The Lane Configuration Editor web service calls back to the CHART GUI using http / XML to indicate the lane editing session has been submitted, and the CHART GUI updates the lane configuration and status for the traffic event. The CHART GUI responds back to the Lane Configuration Editor web service to indicate if its processing was successful, and in turn, the Lane Configuration Editor web service returns a JSON response to the lane editor web page to indicate success or failure. Additionally, if the lane editing session was initialized with a location that includes a lat/long and main route, the Lane Configuration Editor web service sends the lane configuration to the Mapping Lane Configuration web service via http / XML to allow the user specified lane configuration to be stored at the specified location so it is available for use the next time a traffic event occurs near that location.
- The lane editor web page processes the response from the submit request. If an error is indicated, it will be displayed to the user and the lane editing form remains open. If the submit was successful, the lane editing form calls a JavaScript function in its parent window (if present) to let the parent window know the lane editing session has ended. The parent window can then update if needed to show the current lane configuration and status as specified by the user. Finally, the popup window containing the lane editor is closed.

In addition to lane editor related functionality, the Lane Configuration Editor web service also supports a request to allow a lane configuration to be rendered into a GIF image. This feature allows the client application to retrieve an image for a specified lane configuration and status for use within the client application. The returned data includes metadata for the image such as lane boundaries to allow clients to create an image map for the image, making features such as lane selection possible to implement.

3.2.2 Intranet Mapping

The Mapping Lane Configuration web service is a new web service provided as part of the Intranet Mapping application suite. The purpose of this web service is to provide access to lane configuration data available from the State's mapping database, and to allow this data to be augmented by lane configurations specified by users. The lane configuration data in the mapping database is stored per road segment and the user specified lane configurations will utilize this same scheme.

The Mapping Lane Configuration web service supports two requests, a query and a post. The query is used by client applications (e.g. the CHART Lane Configuration Editor web service) to locate lane configurations nearby a given point (lat/long). A radius is provided to define the search area as a circle around the point, and a route can be provided to narrow the results to include lane configurations on a specific roadway. This request is processed by finding all roadway segments that fall within the given search circle, narrowing it to include only those on the specified route (if any), and returning lane configurations for those roadway segments (both those defined in the mapping database and those that are user specified).

The post request provides the ability for user specified lane configurations to augment the data that exists in the mapping database. The client application (e.g. the CHART Lane Configuration Editor web service) can post the lane configuration specified by the user for a specific lat/long and route, and the Mapping Lane Configuration web service will locate the roadway segment on the specified route that is closest to the specified lat/long and store the specified lane configuration as the user specified configuration for that roadway segment. The next time a query is done that includes that roadway segment, this configuration specified by the user will be returned as part of the query results. Each user specified lane configuration will include a timestamp of the time it was stored, allowing for cleanup of the user specified lane configuration data in the future if needed (by a DBA).

Because the post request alters data in the database, the Mapping Lane Configuration web service will only allow this request to be performed by authorized clients. The post request will require two parameters, a clientID and signature, which will be used by the web service to determine if the request is from an authorized client. The web service will keep a list of each authorized client and its associated public key. When a post request is received, it will use the specified clientID to look up the public key and will use the public key to verify the signature that was generated by the client using its private key. Only requests with a verified signature will be processed – others will result in an error being returned to the client.

3.3 Map Integration Part 2 – Locate Between Features

Changes have been made to the object location forms (when adding/editing devices and traffic events) to allow a user to specify that the location of the object is BETWEEN two features, or FROM an intersecting feature and TO a second intersecting feature. When the user selects either of these proximity values, they are provided with additional UI input fields that allow them to specify the second intersecting feature for the location. If a location can be determined for the two intersecting features the system will display markers at each of the locations to visually indicate to the user where the first and second intersecting feature being used in the object location are located. The system will set the location of the object initially to equal the location

of the first intersecting feature, if it has a defined lat/lon coordinate. If no coordinate can be determined for the first intersecting feature, the system will attempt to use the lat/lon coordinate of the second intersecting feature as the initial object location. If the second intersecting feature has no defined coordinate, the system will not be able to set an initial location for the object whose location is being specified. Regardless of the initial location set, the user may double click any location on the map to specify that actual desired coordinate that should be used as the point location of the object.

3.4 External TSSs on Intranet Map

3.4.1 CHART

There are three mapping components to the CHART system:

- Intranet Map – internal map for CHART operators and other agencies connected to the CHART network.
- Internet Map – displayed by CHART-on-the-Web for external users
- Integrated Map – tactical maps to aid CHART operators in geo-locating events and devices

Currently only internal objects are displayed on the Intranet and Internet Maps. For R6, there is a need to display external detectors on the Intranet Map as well; specifically NAVTEQ detectors. The word 'external' in this context refers to CHART and not the State of Maryland. That is, an external detector is defined as one that is not controlled by the CHART system regardless of whether it is located within the State of Maryland or not. Neither the Integrated Map nor the Internet Maps display external detectors.

The display of external detector data comes with complexity. Due to contract restrictions, only actual MDSHA employees are permitted to view detailed speeds from some of the external detectors; specifically NAVTEQ detectors. Other users may view which range the speed falls in (e.g. under 30 mph, 30-50 mph, 50+ mph) however they must be prevented from seeing precise speeds from these detectors.

Because the existing CHART system already supports the granting of detailed rights to manage the display of detector data to its users, the same ability to manage user rights will be made available to the Intranet Map so that it may apply those same rights to its users. This will be accomplished by creating a new User Management Web Service that the Intranet Map can use to access back-end User Management functions. Only a few of the back-end service's features need to be exposed for this release such as the validation of a username and password and the retrieval of rights. The design will support the easy addition of the remaining features to the User Management Web Service as they are needed in future releases. This design also makes it easier for other applications (e.g. EORS) to use the same User Management rights in the future possibly leading to a single sign-on for all CHART applications.

3.4.2 Intranet Mapping

When a user first opens the Intranet Map they will be given rights associated with a default user. This default user will be defined by the CHART administrator and is expected to be configured so they can view all the events and devices they could view in previous releases including detailed speeds for detectors. For R6, however there will be new external detectors on the Intranet Map including some that require special rights to view (e.g. NAVTEQ). For these the administrator will either see no speed or only a speed range depending on how the CHART administrator configures the default user's rights. To view more information for these detectors, the user will have to log in to the Intranet Map using their normal CHART username and password. Then, based on any rights they gain by logging in, they might be able to view more or better information for these detectors (see Section 5). Because all detectors (internal and external) will be using the existing CHART rights mechanism, the CHART administrator will be able to control how all users may view all detectors on the Internet Map – even internal detectors. This feature could be expanded in future releases to provide additional restricted features on the Intranet Map.

It is conceivable that the act of logging in could actually remove rights if it happens that the user's CHART login provides fewer rights than the default user. This unlikely scenario would be in error in setting up the user's rights, and could be quickly remedied by the CHART administrator.

3.5 Camera Location Data Synchronization

R6 introduces the export of CCTV information. The Internet/Intranet Map and Public Web are special clients of the Exporter in that they do not communicate directly with the Data Exporter. Instead a separate Export Client application acts as an intermediary. The Export Client's prime responsibility is to gather data from the Data Exporter and update the database shared by the Internet/Intranet Map. When the Export Client detects a configuration change it notifies a synchronization application via a HTTP message. This triggers the maps to recache their data.

The following figure shows the common ground between the CHART device data and TMDD standard, along with the exported data that is available to the external entities.

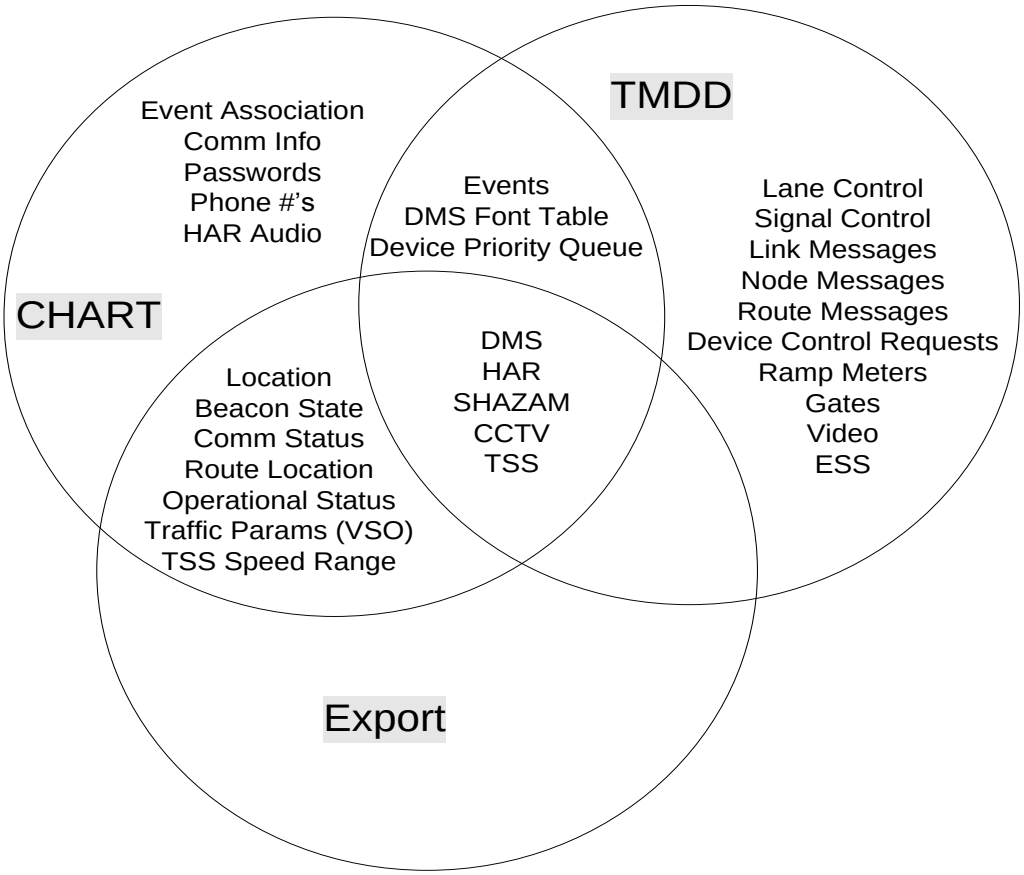


Figure 3-1 Export TMDD Devices

Please refer to the CHART CCTV Export ICD for additional details on how each field is exported.

3.6 Error Processing

In general, CHART traps conditions at both the GUI and at the server. User errors that are trapped by the GUI are reported immediately back to the user. The GUI will also report communications problems with the server back to the user. The server may also trap user errors and those messages will be written to a server log file and returned back to the GUI for display to the user. Additionally, server errors due to network errors or internal server problems will be written to log files and returned back to the GUI.

3.7 Packaging

3.7.1 CHART

This software design is broken into packages of related classes. The table below shows each package that is new or changed to support the Release 6 features.

Package Name	Package Description
CHART2.Common	This CORBA system interface package was changed to add the new proximity values (NONE, BETWEEN, and FROM/TO. It was also changed to allow an object location to have any number of intersecting features that describe the location rather than only supporting 0 or 1. As of R6 the system will utilize only 0 (NONE), 1 (AT, PAST or PRIOR) or 2 (BETWEEN, FROM/TO) intersecting features.
CHART2.EORS	This CORBA system interface package was changed to add support for a queued status and last modified time for each EORS permit.
CHART2.LaneConfigUtil	The package is new for R6 and contains utility classes related to lane configurations, including a Model sub package for modeling lane configurations, a Display sub package for rendering images, and an IDL utility sub package for conversion to/from IDL.
CHART2.TrafficEventManagement	This CORBA system interface package is changed to remove some fields related to lane configuration that are no longer needed.
CHART2.Utility	This package contains generic utility classes that are used by chart services. This ObjectLocationUtility class in this package has been updated to handle optional secondary intersecting features and the new Intersecting Feature Proximity Type values mentioned above. A new Direction enumeration will be added, and classes will be added to assist in the cleanup of temporary files.
CHART2.Utility.ObjectCache	This package contains classes related to the ObjectCache and Proxy objects that are stored in the ObjectCache. Proxy objects related to optional secondary intersecting features and new Intersecting Feature Proximity Type values mentioned above. Most notably the ProxyBasicTrafficEvent class was changes to simplify the event duplication comparison logic used to drive the creation of DuplicateEventAlert creation.
CHART2.webservices.dataexporter. cctvexportmodule	This package is new for R6 and contains classes that comprise the CCTV export.
CHART2.webservices.dataexporter. tssexportmodule	This package is changed for R6 to support the export of external detectors.
CHART2.webservices. dataexporter.utility	This package contains generic utility class used by the dataexporter modules. The ObjectLocationUtility class was modified to handle the export of the new optional secondary intersecting features and new Proximity Type values mentioned above.
CHART2.webservices. exportlistenermodule.camera	New package that supports updating the CHARTWeb database with exported CCTV inventory and status updates.
CHART2.webservices. exportlistenermodule.camera	This package is new for R6 and contains classes that implement the ExportClient for CCTV export.
CHART2.webservices. exportlistenermodule.tss	This package is changed for R6 to support the export of external detectors.
CHART2.webservices. exportlistenermodule.Utility	This package contains generic utility classes used by the exportlistenermodule. The ExportListenerUtility class has been modified to handle new location related elements updated in the Exporter XSD. XSD changes were done to support changes in the CHART2.Common package.
CHART2.webservices. laneeditormodule	This package is new for R6 and contains the classes for the Lane Configuration Editor web service.
CHART2.webservices. usermanagementmodule	This package is new for R6 and contains classes that comprise the initial web-based version of the CORBA User Management Service.
CHART2.webservices. usermanagementmodule.utility	This package is new for R6 and contains classes that support the User Management Web Service.

Package Name	Package Description
CHART2.webservices.util.jaxbutil	This package is new for R6 and contains generic functionality for converting between XML and JAXB objects, and for dealing with requests and responses using JAXB objects.
CHART2.webservices. wsclientmodule	This package is changed for R6 to support a more orderly shutdown of the ExportClient.
CHART2.xsdutil	This package is new for R6 and contains functionality for converting between XML and JAXB objects that are generated from the XSD files. This will have at least two subpackages: CHART2.xsdutil.common (corresponding to CHART2.xsd.common) and CHART2.xsdutil.laneconfig (corresponding to CHART2.xsd.laneconfig)
chartlite.data	This package contains the GUI data elements that are used by the chartlite.servlet package to create web pages and XML/JSON data responses to satisfy requests. This package has been changed to add an EORSDataManager class that manages a cache of EORS permits and synchronizes that data in that cache as necessary. Classes have also been added to return scored results of searches performed on EORS permits.
chartlite.data.trafficevents	This package will be refactored to move much of the lane configuration functionality to the CHART2.LaneConfigUtil.Model package, to make it available for use in the Lane Configuration Editor and other CHART applications.
chartlite.flex	This package contains the Abobe Flex applications that are used for creating traffic events, setting object locations, and viewing the home page traffic events and alerts lists. This package is changed to allow the operator to select the new proximity values and to specify a second intersecting feature when appropriate.
chartlite.lanedisplay	This package will be removed, and its contents will be refactored into the CHART2.LaneConfigUtil.Model and CHART2.LaneConfigUtil.Display packages.
chartlite.servlet	This package contains the GUI application code that handles incoming http requests for pages or data. This package is changed to add an EORS permit request handler which handles AJAX request for permit suggestion as the user types as well as full EORS permit searching.
chartlite.servlet.trafficevents	This package contains GUI classes related to traffic event web pages. The existing functionality for editing lane configurations will be removed, and functionality for interacting with the Lane Configuration Editor will be added.

3.7.2 Mapping

The table below shows each package (Namespace in .Net) that is new or changed to support the Release 6 Camera Locations Data Synchronization feature.

Namespace Name	Namespace Description
CHARTMap.Handlers	Existing namespace and extended for the class “cameraInvetoryHandler” that comprise the Camera Location Data Synchronization in R6.
CHARTMap.Lib.CHARTConfig	Existing namespace modified for External TSS feature
CHARTMap.Lib.CHARTRole	Added new namespace for External TSS feature

3.8 Assumptions and Constraints

1. Assumption: The State GIS mapping data defines each roadway segment such that only a single lane configuration applies to that segment of roadway in the real world. (If scenarios exist where the lane configuration actually changes at different points along the segment, this could lead to a scenario where one user specified lane configuration overwrites another that was previously stored for that segment, both of which are correct). We feel this is not likely since the State's mapping data defines only 1 lane configuration per roadway segment.
2. Assumption: There is no requirement for the number of simultaneous external clients the Exporter must support however enough parameters have been exposed in the service's property file to allow for run-time tuning in case load becomes a problem. Examples include the ability to collect multiple changes before exporting them to a client, limiting the size of export queues, reducing how often a client (e.g. ExportClient) requests information, limiting the number of TrafficEvent history log entries returned, etc.
3. Constraint: The Intranet/Internet mapping application will not be able to differentiate between cameras with flash streaming server configuration information and those without, since flash streaming server configuration information is not exported in R6.
4. Assumption: System administrator only imports a subset of all external detectors (e.g., only detectors in or near Maryland). Importing every detector from all RITIS sources (e.g., across the entire state of Virginia) and distributing them to the Intranet/Internet map may cause resource problems.

4 Use Cases – EORS Integration

The use case diagrams depict new functionality for the CHART EORS Integration functionality and also identify existing features that will be enhanced. The use case diagrams for this feature exist in the Tau design tool in the Release6 area. The sections below indicate the title of the use case diagrams that apply to this feature.

4.1 CHART

4.1.1 ManageTraffic Events (Use Case Diagram)

This diagram models the actions that an operator may take that relate to traffic events.

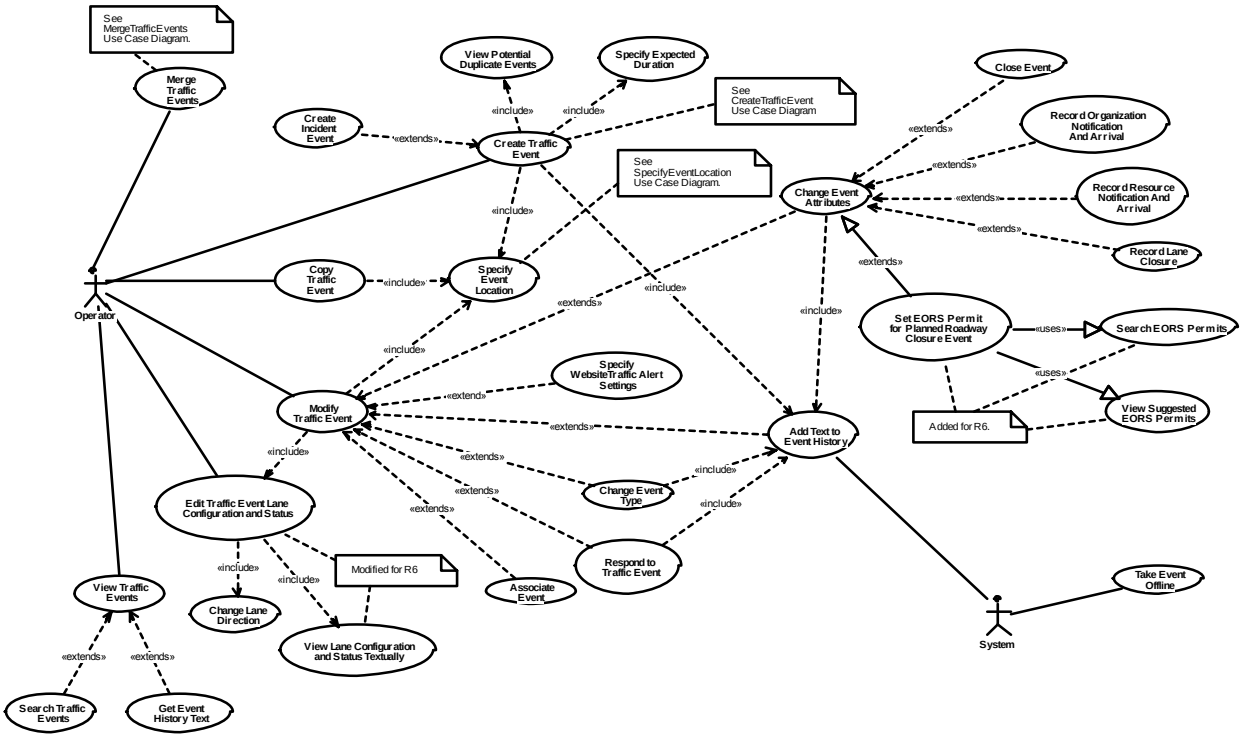


Figure 4-1 Manage Traffic Events (Use Case Diagram)

4.1.1.1 Add Text to Event History (Use Case)

An operator with the proper functional rights may add text to a traffic event's event history.

4.1.1.2 Associate Event (Use Case)

An operator with the proper functional rights may associate related traffic events.

4.1.1.3 Change Event Attributes (Use Case)

An operator with the proper functional rights may edit traffic event information after the event has been created. This includes closing the event, adding text to the event history, recording lane closures, and recording organization and resource arrivals.

4.1.1.4 Change Event Type (Use Case)

An operator with the proper functional rights may change the traffic event type.

4.1.1.5 Change Lane Direction (Use Case)

The lane configuration editor shall allow the user to change the traffic flow direction for a particular lane. This includes setting a lane to be multi-directional to indicate alternating use of a single lane. The editor shall allow the user to use hot keys to set the traffic flow direction. The ability to use "multi-directional" can be disabled as part of the initialization of the lane configuration editor.

4.1.1.6 Close Event (Use Case)

An operator with the proper functional rights may close an event.

4.1.1.7 Copy Traffic Event (Use Case)

The user with the correct functional rights will be able to create a copy of an existing traffic event.

4.1.1.8 Create Incident Event (Use Case)

An operator with the proper functional rights may create a new incident event.

4.1.1.9 Create Traffic Event (Use Case)

The user with the correct functional rights may add a new traffic event. When creating a traffic event, the system will show the user a list of existing traffic events that may be duplicates of the new event being created based on the user's selections for the new event's location. External and pending events do not appear as possible duplicate events.

4.1.1.10 Edit Traffic Event Lane Configuration and Status (Use Case)

An operator with the manage traffic events user right may edit the lane status of a traffic event, including changing direction for a particular lane. This only applies to open Planned Roadway Closures, Incidents, and Special Events.

4.1.1.11 Get Event History Text (Use Case)

An operator with the correct functional rights may view the text entries that have been added to an event.

4.1.1.12 Merge Traffic Events (Use Case)

This use case represents the merge traffic event operation. A user with manage traffic event right merges the data of two traffic events. See MergeTraffic Events use case diagram.

4.1.1.13 Modify Traffic Event (Use Case)

An operator with the proper functional rights may edit traffic event information after the event has been created. This includes responding to the event, editing lane status, editing location, associating with another event, and specifying other event attributes.

4.1.1.14 Record Lane Closure (Use Case)

The lane configuration editor shall allow the user to specify the status of each lane in the configuration as being open, closed, or unknown. A feature shall also exist to allow the user to set all lanes to open without having to set the status individually for each lane. Hot keys for setting lane status will be supported. The system will record the date/time a lane is opened or closed. When the lane configuration is initialized to include (not disable) the feature that sets the initial lane status in the main direction to unknown, then When a user makes the first change to the status of a lane in the main direction whose status was initially defaulted to "unknown", the system will set the status of all other lanes in the main direction that have a status of "unknown" to "open".

4.1.1.15 Record Organization Notification And Arrival (Use Case)

An operator with the proper functional rights may record the participation of various organizations in the event resolution.

4.1.1.16 Record Resource Notification And Arrival (Use Case)

An operator with the proper functional rights may record the participation of various resources in the event resolution.

4.1.1.17 Respond to Traffic Event (Use Case)

The system allows an operator to control devices in response to an event through the use of a response plan. The user may add devices to the plan, select the desired state of the devices, then activate the plan. Any of the devices used by the event response plan may be deactivated while the event is open by removing the item for that device from the plan. When the event is closed, if the response plan is active, it will be deactivated automatically.

4.1.1.18 Search EORS Permits (Use Case)

The system will allow the user to perform a text search on the following fields of an EORS permit: permit tracking number, start county name, end county name, permit type, route location, route type, route number, work order description, permittee name, contract number and days of week. The system will score each matching permit based on the

percentage of the user entered search terms were found in these fields and will present the results in order of relevance. The user will be shown a summary of each matching permit and will be able to show/hide additional details about each permit.

4.1.1.19 Search Traffic Events (Use Case)

An operator with the proper functional rights may search the CHART system for traffic events.

4.1.1.20 Set EORS Permit for Planned Roadway Closure Event (Use Case)

A user may set the EORS permit associated with a planned roadway closure event. Doing so will set the EORS permit tracking number into the planned roadway closure event details. The system will assist the user in finding the correct EORS permit by suggesting potential matching permits as the user types a permit tracking number. In the event that the user does not see the permit they are looking for in the list of suggested permits, the system will provide a more advanced searching capability that will search on other fields of the permit (in addition to the tracking number). The user will be able to specify if the list of permits considered for suggestion or searching should be limited to only the active and queued permits, or if all permits currently available in the EORS system should be considered.

The user will be able to indicate if permit tracking numbers should be considered to match their search terms if the permit number starts with the user entered text (only) or contains the user entered text anywhere within the permit number. Additionally the user may indicate that the permit may start with or end with (last 4 digits) the user entered text.

4.1.1.21 Specify Event Location (Use Case)

The event location choices will be populated using data from the CHART Mapping application database.

By default, MD will be selected as the state.

If the selected state is MD, the user will be required to select a predefined MD county/region. If a route is specified, the user will first select a route type from a pick list ("I", "US", or "MD") and the route type will be used to populate the list of predefined routes. To specify a route, the user will be required to select one of the predefined routes if the state is MD. If the state is not MD, the user will be able to enter a county name / region name and route number as freeform text.

If a route number is specified, the user will be able to select intersecting roads by route number or route name, or specify the state or county milepost. Additionally the user will be able to specify whether the traffic event is at, prior, or past the intersecting feature ("at" will be selected by default).

If the state is MD, the list of intersecting route numbers and names will be populated for the

user as suggestions; however, the user can still specify freeform text for an intersecting route number, route name, county milepost, and state milepost even if the state is MD.

4.1.1.22 Specify Expected Duration (Use Case)

An operator with the proper functional rights may specify the expected duration of an event.

4.1.1.23 Specify Website Traffic Alert Settings (Use Case)

An operator with the proper functional rights may specify whether or not the traffic event warrants a "Traffic Alert" on the public CHART web site, and may optionally provide specific alert text to be associated with the Alert.

4.1.1.24 Take Event Offline (Use Case)

The system periodically checks for closed events and takes them offline provided that a configured interval of time has elapsed since the event was closed.

4.1.1.25 View Lane Configuration and Status Textually (Use Case)

The system shall provide a read-only textual description of the specified lane configuration and status.

4.1.1.26 View Potential Duplicate Events (Use Case)

An operator with the correct functional rights will be presented with a list of potential duplicate traffic events based on the event location. The operator will have the option to then merge these events.

4.1.1.27 View Suggested EORS Permits (Use Case)

As the user types in the search field the system will present a set of suggested permits based on the permit tracking number. The suggested permits will each display the tracking number and some summary information about the permit as well as a link/button that allows the user to associate the permit to the planned closure directly from the suggestion.

4.1.1.28 View Traffic Events (Use Case)

An operator with the correct functional rights may view a traffic event.

5 Detailed Design – EORS Integration

5.1 Human-Machine Interface

5.1.1 EORS Permit Searching

The CHART GUI includes the tools that assist operators in identifying the correct EORS permit to associate with a CHART Planned Closure event. This capability is available only after the CHART Planned Closure event has been opened. The user starts the process of selecting an EORS permit from the details page of a CHART Planned Closure Event as shown in figure 4.1.

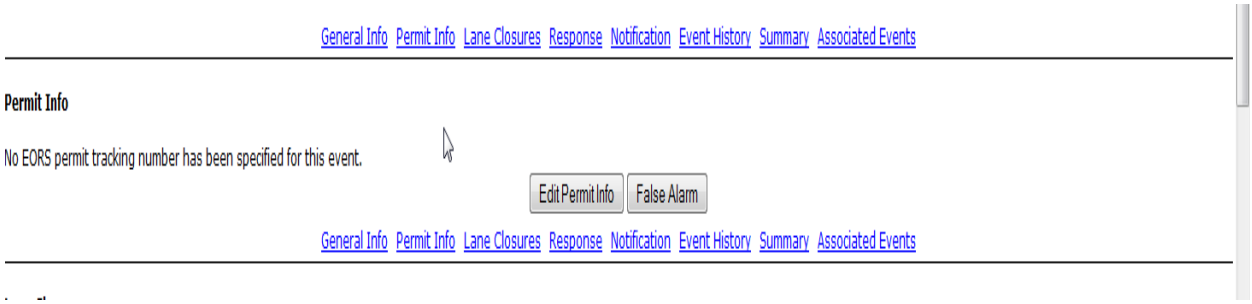


Figure 5-1 Planned Closure With No Permit Selected

When the user clicks the “Edit Permit Info” button, they are taken to the Search EORS Permits page where they may type either a tracking number or other permit information into the search field. The EORS permit search page is shown in figure 4-2. The user may use the checkboxes provided to limit the search to permits that are in the active and queued states in EORS only (default), or to search all available EORS permits (by un-checking the box). The user may also control whether tracking number matching is limited to the last four (4) digits of the EORS permit tracking number (default) or checks both the start of the tracking number and the last four digits (by un-checking the box).

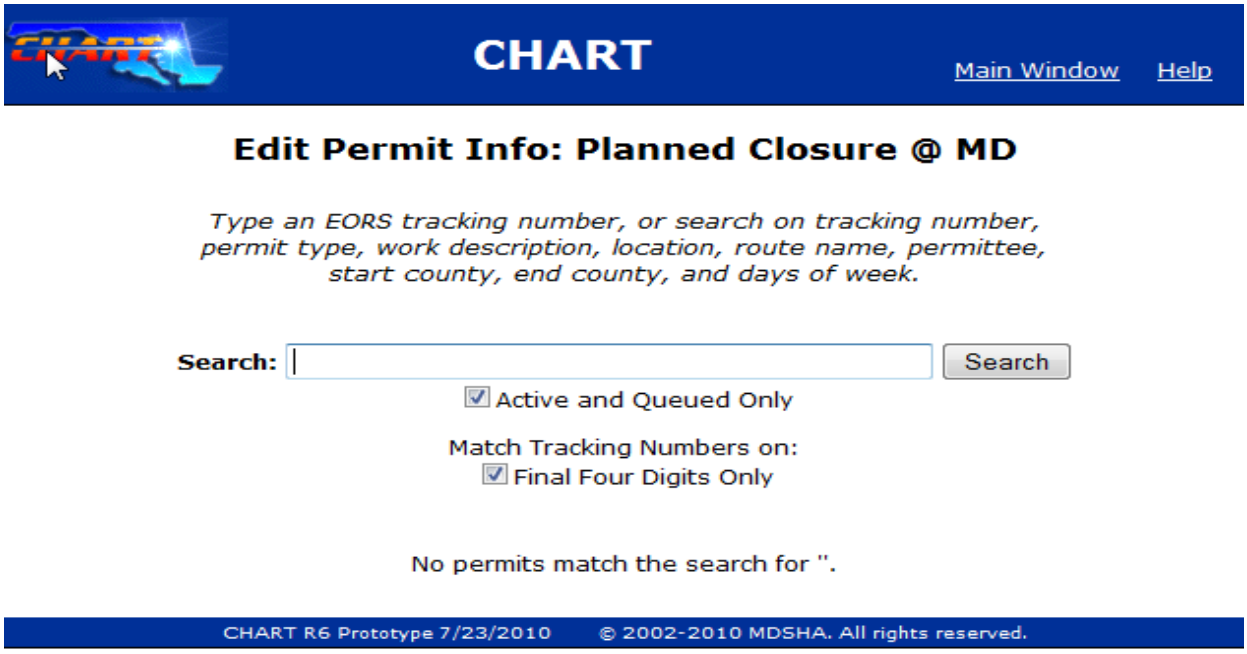


Figure 5-2 EORS Permit Search Page

As the user types, the system will search for tracking numbers that match the text the user has entered according to the tracking number match method selected. Matches are displayed below the text field as shown in figure 4-3.

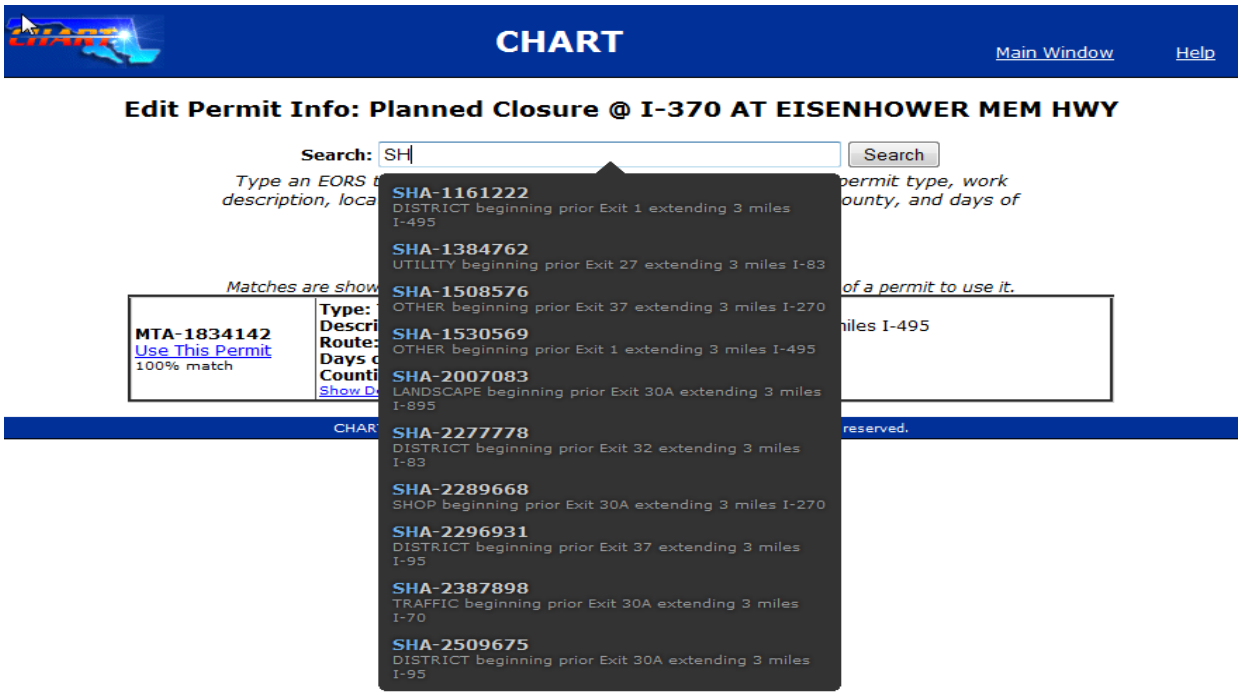


Figure 5-3 EORS Permit Suggestions

As the user continues to type the list of suggestions will narrow. When the user has found the permit they are interested in, they may select it from the list as shown in figure 4-4. Doing so will cause the selected EORS permit to be associated with the CHART Planned Closure and will return the user to the Planned Closure Details Page where they will see the full details of the selected permit.

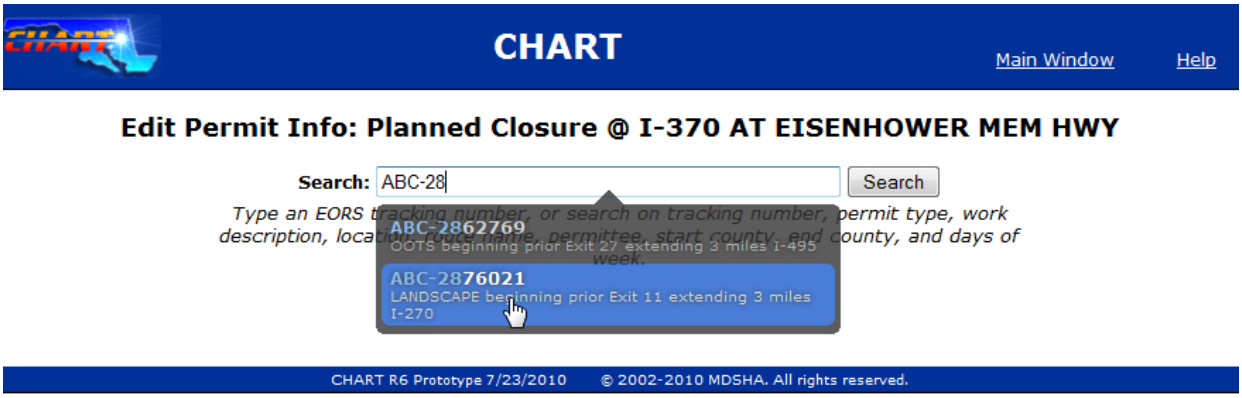


Figure 5-4 EORS Permit Suggestion Selection

If there are no suggestions that match the text the user has entered, the system will display a no suggestions message below the text field as shown in figure 4-5. The user may then use the "Search" button to the right of the text entry field to perform a full search of the permits checking each of the key fields for matches on the text that the user has provided.

Figure 5-5 No Permit Suggestions

When a search is performed in this manner, there may be many permits that match the search criteria, so the system returns the results in order of relevance to the search criteria and paginated as shown in figure 4-6.

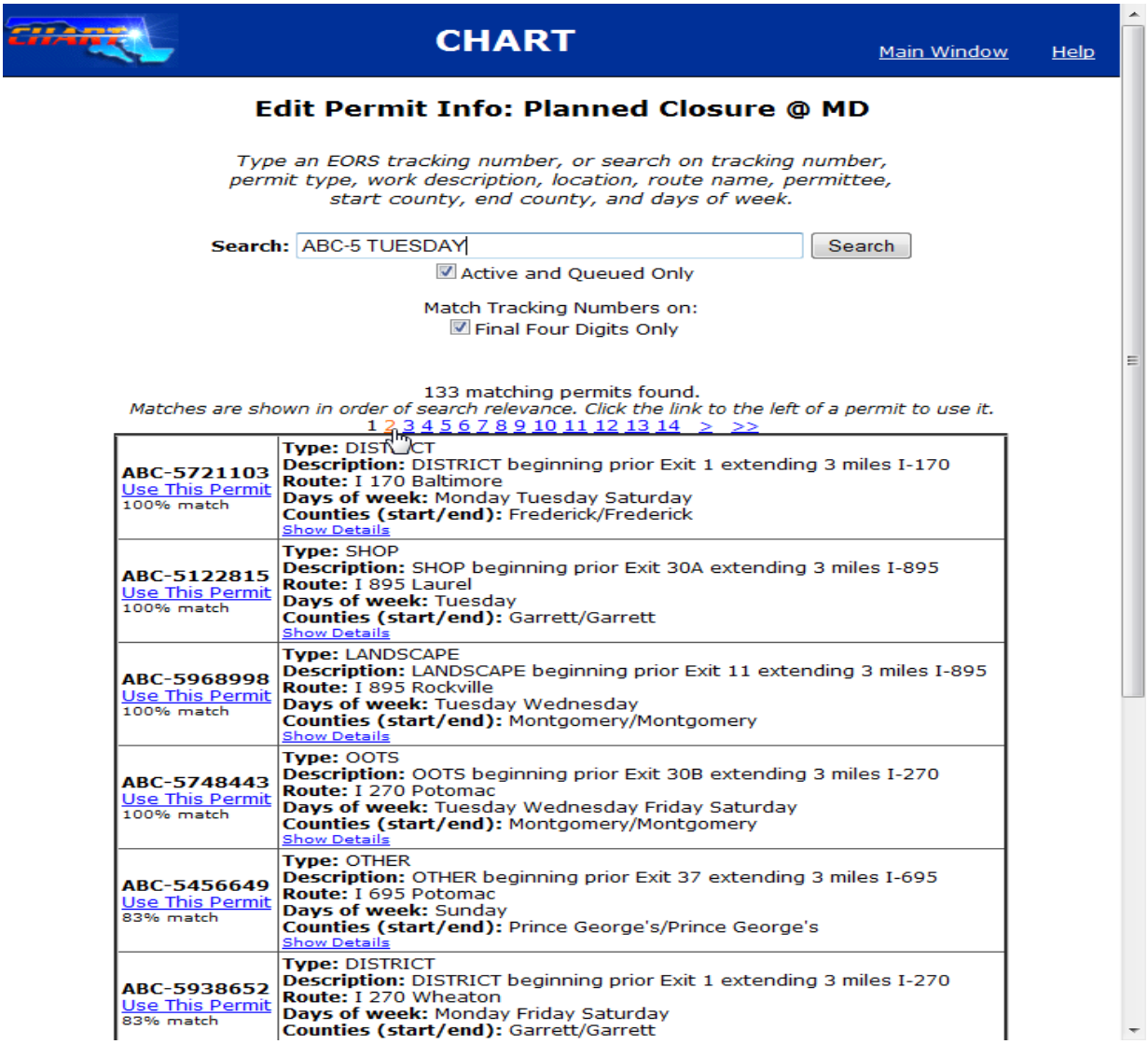


Figure 5-6 Permit Search Results

The user may click any page number to jump to that page, or may click the “>” or “<” to move forward or backward one page. The user may also use the “>>” or “<<” links to jump to the last or first page of results directly. In figure 4-7 the second page of results for the search is shown.

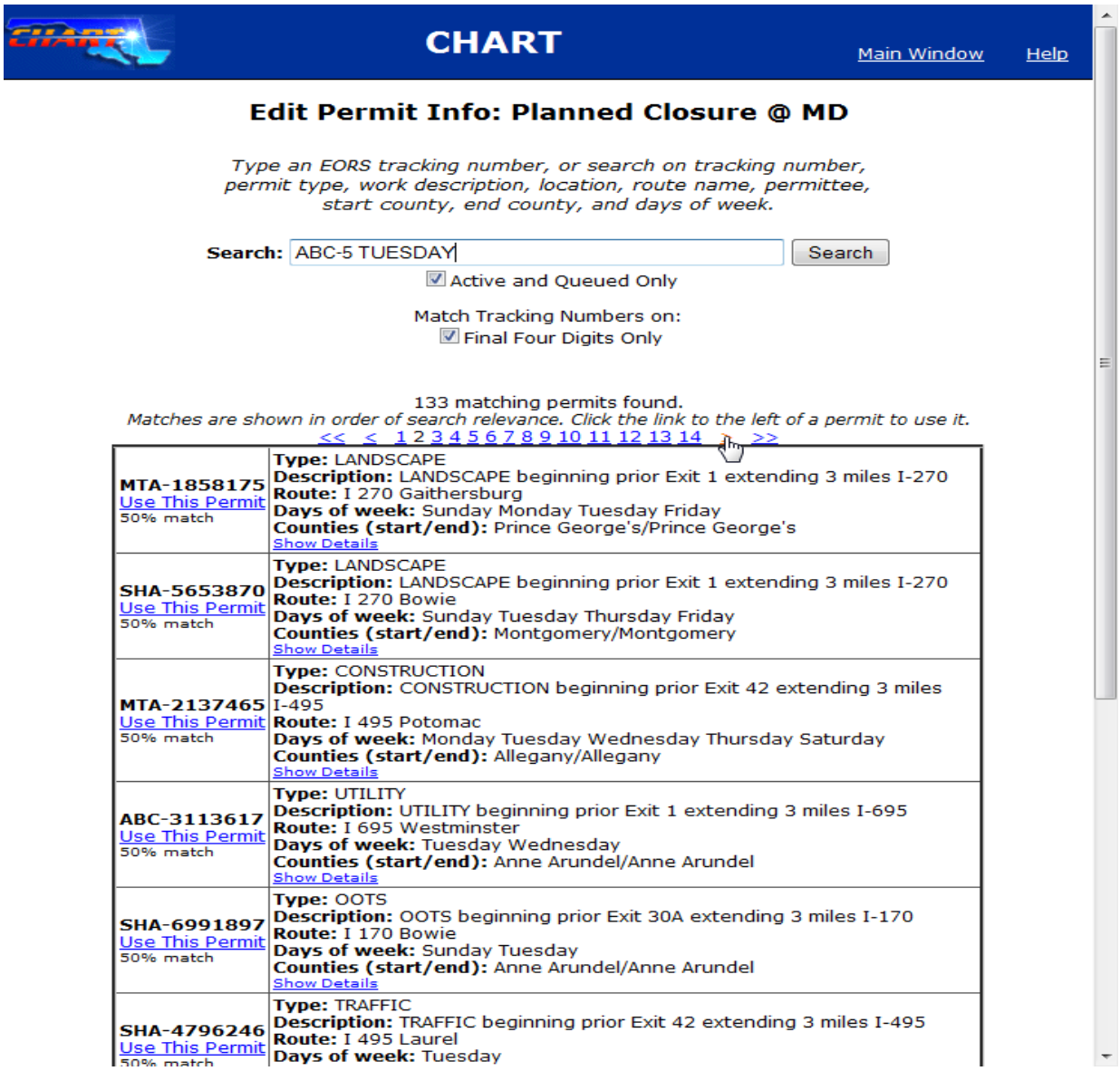


Figure 5-7 More Permit Search Results

For each permit in the search results, the system shows a summary of the permit data. A "Show Details" link is provided that allows the user to view the full permit details. Once this link has been clicked the details are shown below the permit summary row, and the link text changes to "Hide Details" as shown in figure 4-8. Clicking the "Hide Details" link will return the user to the prior view where only the summary information for that permit is visible.

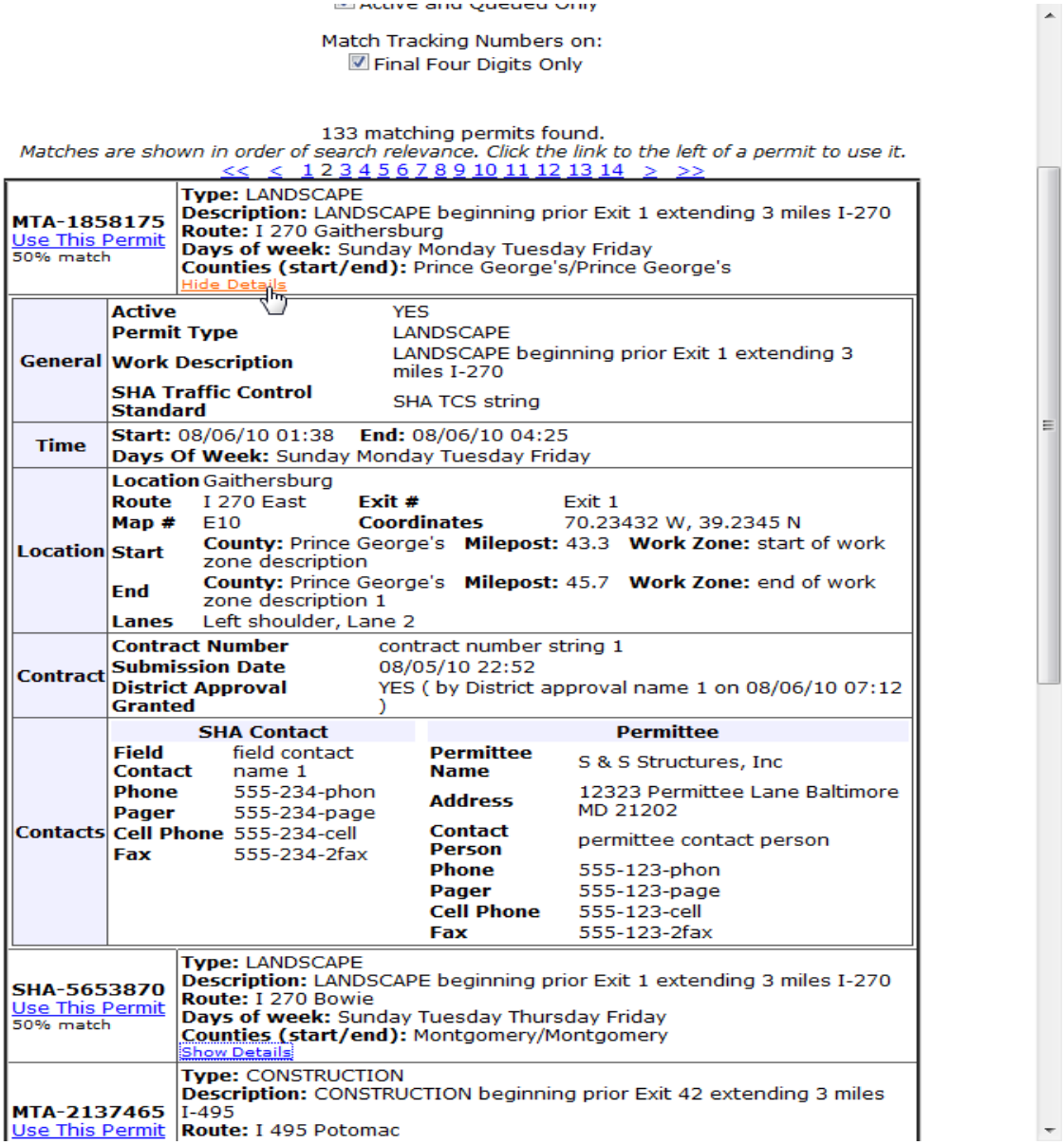


Figure 5-8 Show/Hide Permit Details

When the user decides that they have found the permit that they are searching for, they may click the "Use This Permit" link to the left of the summary (as shown in figure 4-9) to immediately associate the permit with the CHART planned closure. Doing so will return the user to the details page for the planned closure.

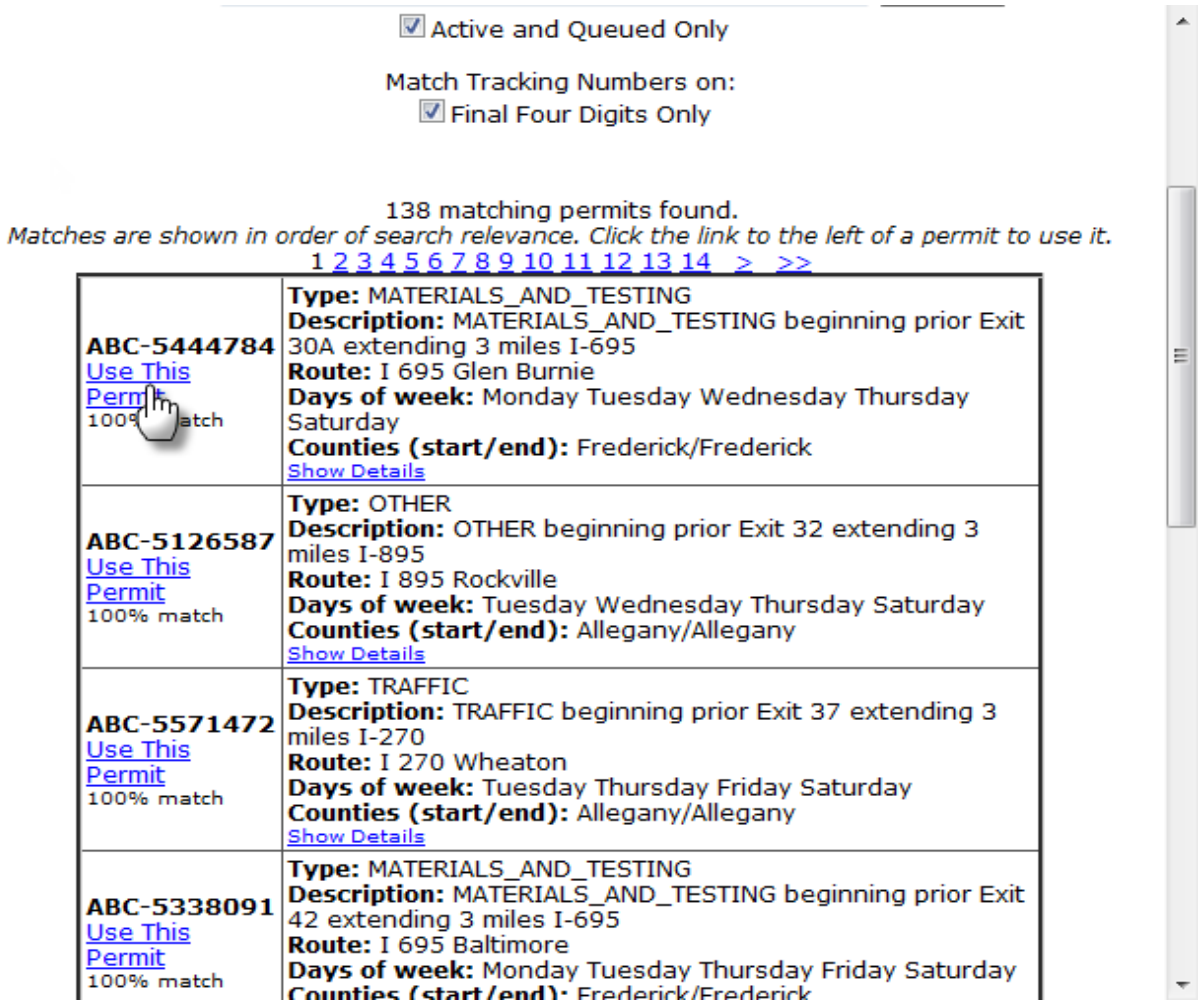


Figure 5-9 Use Search Result

Once the user has associated a permit with the CHART Planned Closure, either by using a suggested permit or from the search results pages, they are returned to the Planned Closure details page where they can see the full details of the permit that is associated with the event. If the user is not satisfied with the selected permit they may click the "Edit Permit Info" button to start the process over again, or they may click the "Clear Permit" button to remove the association and return the Planned Closure Event to a state where it has no permit associated. The Planned Closure details page is shown in figure 4-10.

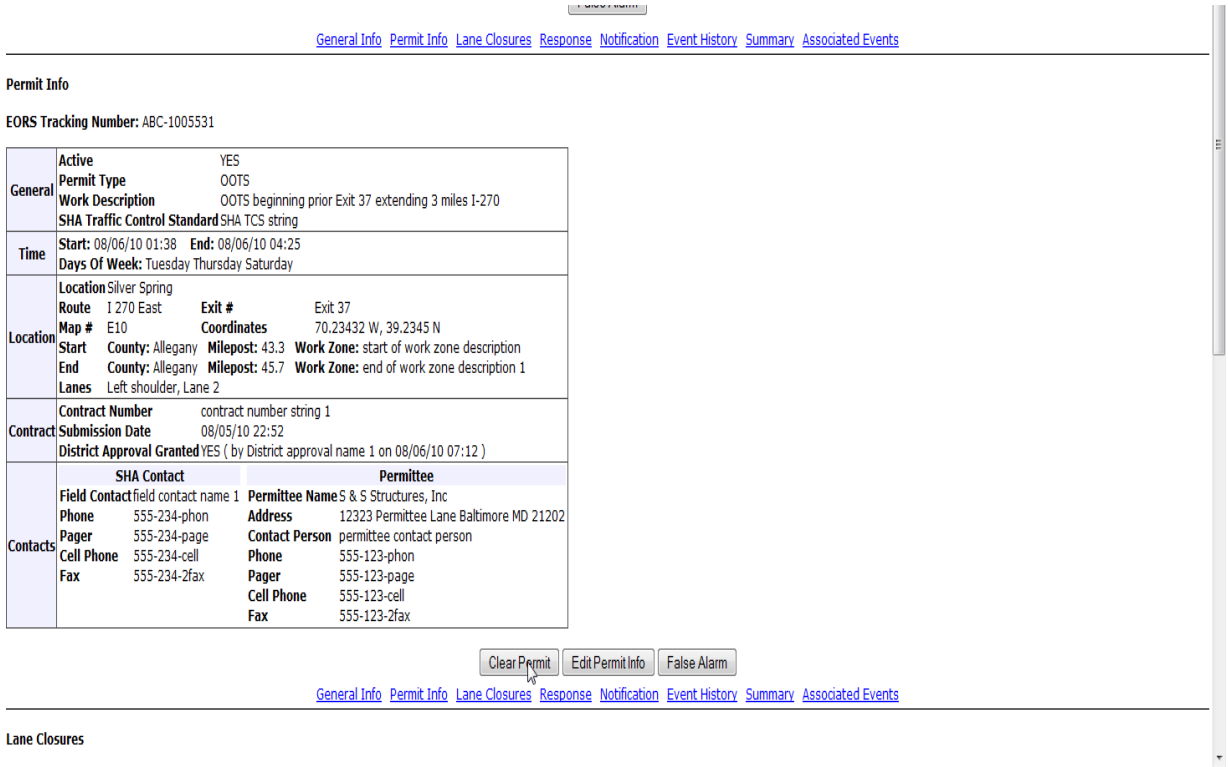


Figure 5-10 Planned Closure Event with Associated EORS Permit

If the user opts to clear the permit by clicking the “Clear Permit” button they will be asked to confirm that they actually want to remove the association before the permit is actually cleared as shown in figure 4-11. Clicking “Cancel” will result in the permit not being disassociated from the Planned Closure event. Clicking “OK” will remove the association and the Planned Closure Details page will show no permit associated with the event (Figure 4-1).

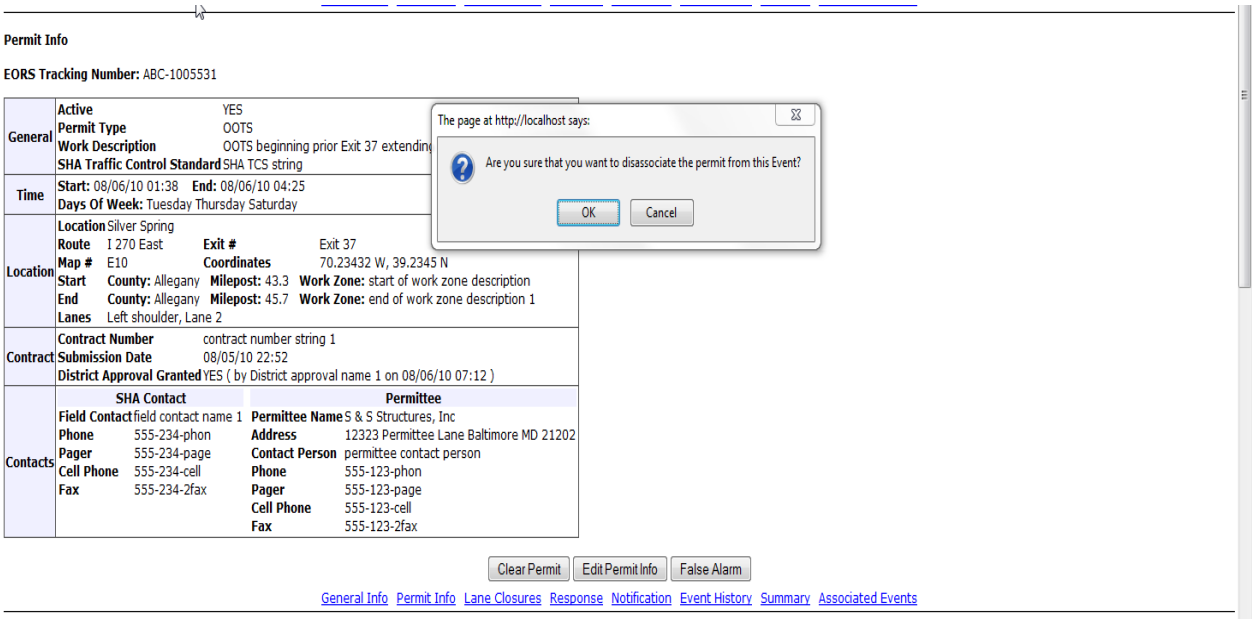


Figure 5-11 Clear Permit Confirmation

5.2 System Interfaces

The class diagrams in this section describe the CORBA interface classes and relationships that are being added or modified to support the EORS Integration feature.

5.2.1 Class Diagrams

5.2.1.1 EORS (Class Diagram)

This diagram shows interfaces and classes that define the EORS system data that is available to the CHART system.

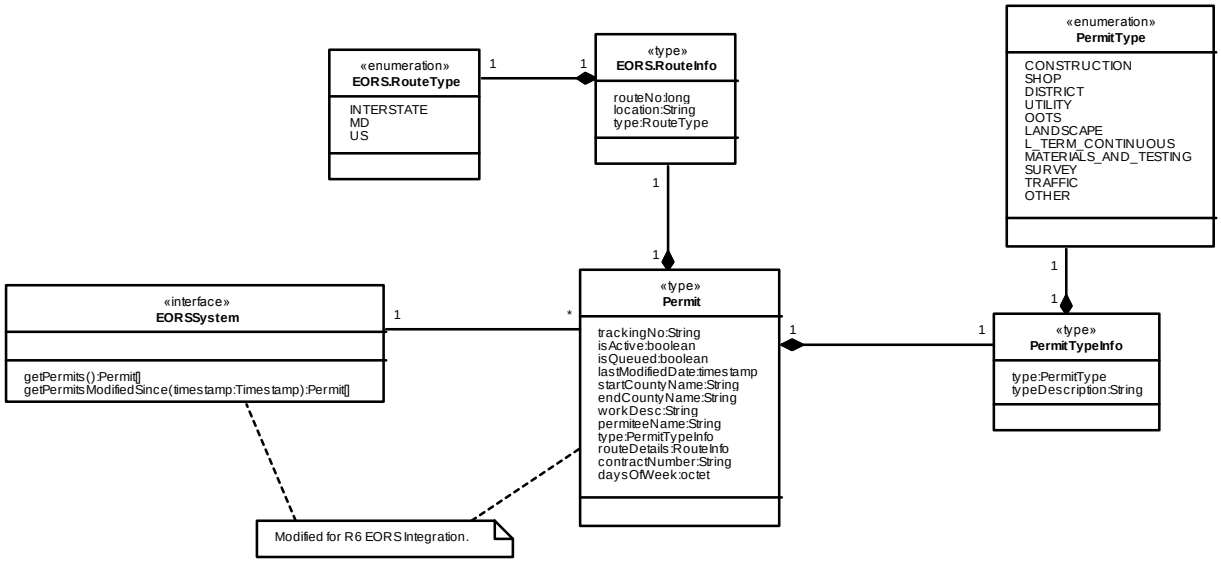


Figure 5-12 EORS (Class Diagram)

5.2.1.1.1 EORS.RouteInfo (Class)

This class contains information about the route affected by an EORS permit.

5.2.1.1.2 EORS.RouteType (Class)

This enumeration describes the types of routes supported by the EORS system.

5.2.1.1.3 EORSSystem (Class)

This interface shows the operations that are available for querying the list of permits currently available from the EORS system database.

5.2.1.1.4 Permit (Class)

This class represents an EORS permit. It shows a subset of the data elements available for each permit.

5.2.1.1.5 PermitType (Class)

This enumeration defines the permit types supported by the EORS system.

5.2.1.1.6 PermitTypeInfo (Class)

This class contains information related to the type of an EORS permit.

5.3 EORS Module

5.3.1 Class Diagrams

5.3.1.1 EORS Module Classes (Class Diagram)

This diagram shows the classes that compose the EORS Service module and the relationships between them.

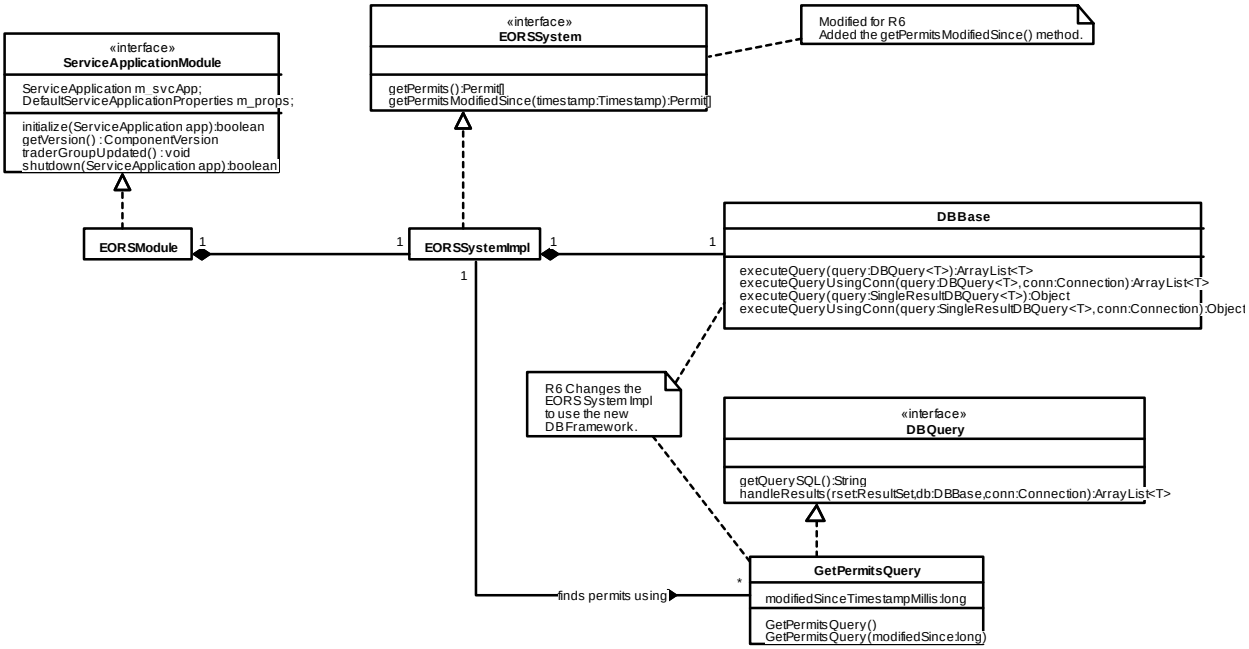


Figure 5-13 EORS Module Classes (Class Diagram)

5.3.1.1.1 DBBase (Class)

This class provides a base class that application DB classes can extend OR delegate to in order to take advantage of the DBQuery interface for performing database queries.

5.3.1.1.2 DBQuery (Class)

This interface should be implemented by any class that acts as a database query that can be executed by the DBBase and returns 0 to many records.

5.3.1.1.3 EORS Module (Class)

This class provides an implementation of the ServiceApplicationModule framework for the EORS module.

5.3.1.1.4 EORSSystem (Class)

This interface shows the operations that are available for querying the list of permits currently available from the EORS system database.

5.3.1.1.5 EORSSystemImpl (Class)

This class implements the EORSSystem CORBA interface. It is the CORBA object that the CHART system calls to get EORS permits from the EORS system database.

5.3.1.1.6 GetPermitsQuery (Class)

This class provides an implementation of the DBQuery interface that will return a collection of EORS.Permit objects. It can be used to find all permits, or all permits that have been modified since a specified timestamp.

5.3.1.1.7 Service Application Module (Class)

This interface is implemented by modules that serve CORBA objects. Implementing classes are notified when their host service is initialized and when it is shutdown. The implementing class can use these notifications along with the services provided by the invoking Service Application to perform actions such as object creation and publication.

5.3.2 Sequence Diagrams

5.3.2.1 EORSSystemImpl:getPermitsModifiedSince (Sequence Diagram)

This diagram shows the processing performed when a client requests the list of permits modified since a specified date/time. The system creates a new GetPermitsQuery using the client provided timestamp. The EORS database is then queried to get all permits that have been modified since the specified time. If an error occurs a CHART2Exception will be thrown, otherwise the set of Permits returned from the query will be returned to the client.

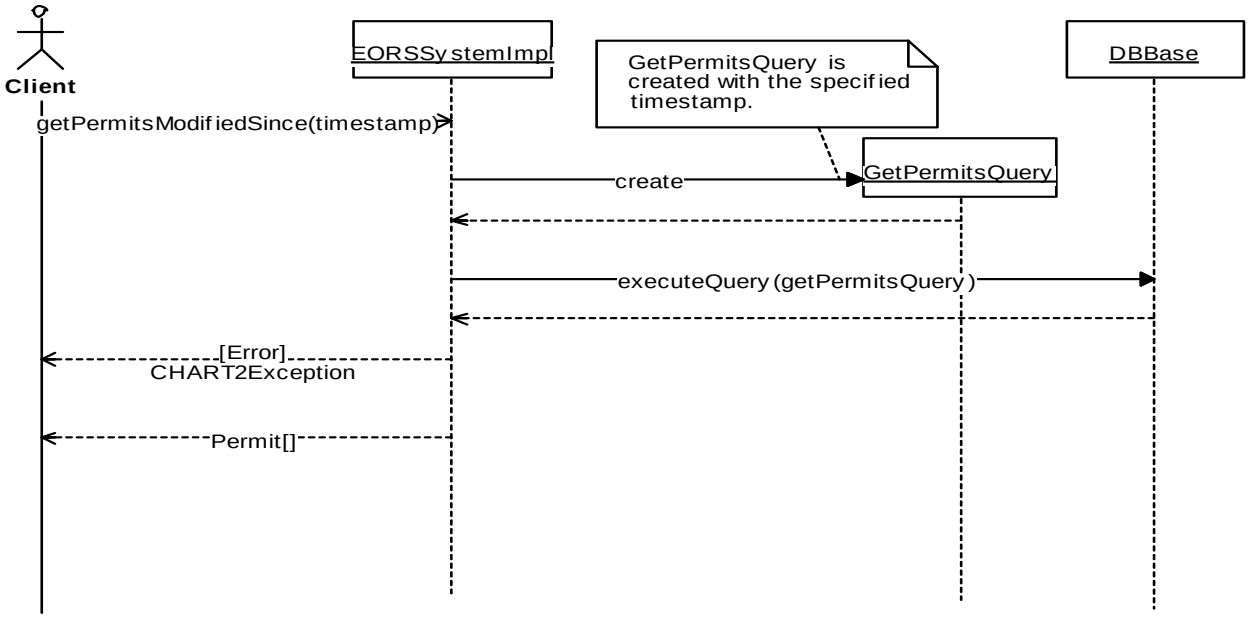


Figure 5-14 EORSSystemImpl:getPermitsModifiedSince (Sequence Diagram)

5.4 EORS GUI

5.4.1 Class Diagrams

5.4.1.1 EORSDataClasses (Class Diagram)

This diagram shows the data classes that are used for maintaining a cache of EORS permits and searching it.

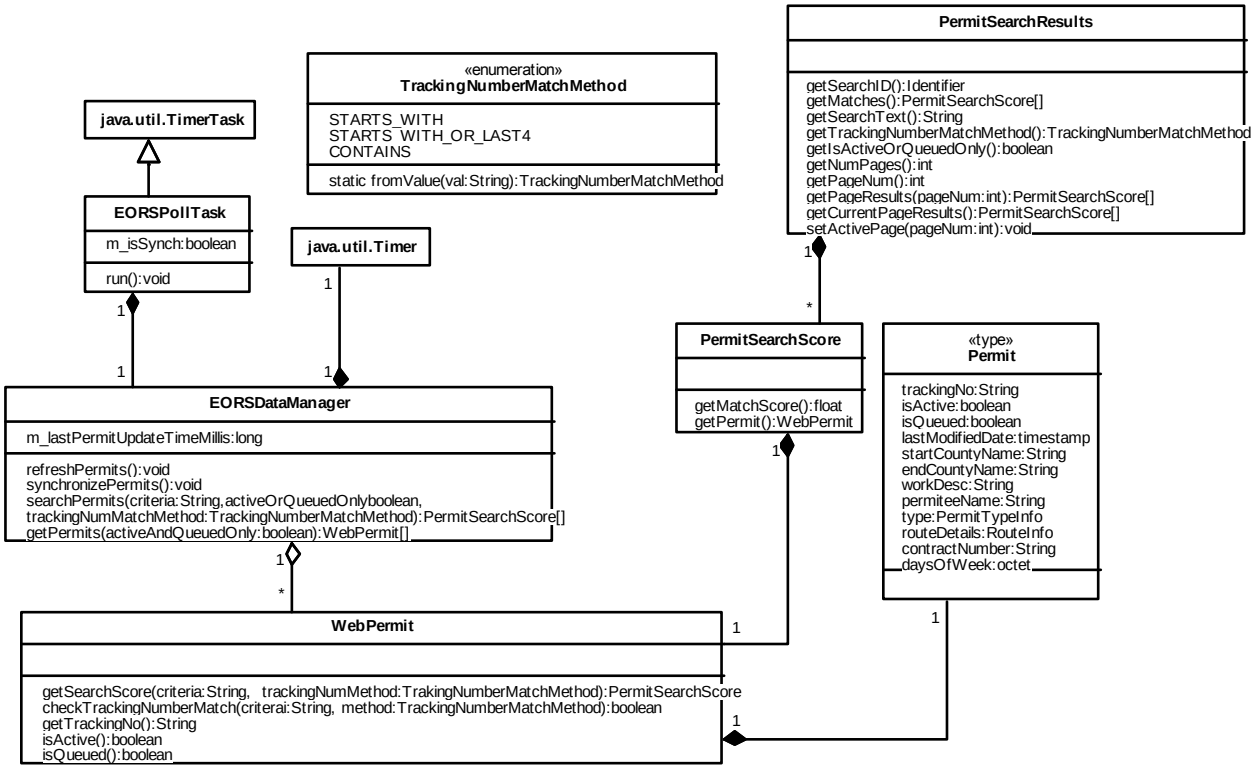


Figure 5-15 EORSDataClasses (Class Diagram)

5.4.1.1.1 EORSDataManager (Class)

This class is responsible for managing the cache of EORS permits.

5.4.1.1.2 EORSPollTask (Class)

This class is used to asynchronously refresh or synchronize the cache of EORS permits using a Timer.

5.4.1.1.3 java.util.Timer (Class)

This class provides asynchronous execution of tasks that are scheduled for one-time or recurring execution.

5.4.1.1.4 java.util.TimerTask (Class)

This class is an abstract base class which can be scheduled with a timer to be executed one or more times.

5.4.1.1.5 Permit (Class)

This class represents an EORS permit. It shows a subset of the data elements available for each permit.

5.4.1.1.6 PermitSearchResults (Class)

This class holds the results of a permit search for the purposes of retaining the search results and allowing the user to view them.

5.4.1.1.7 PermitSearchScore (Class)

This class is capable of holding the search score for a particular permit during a search.

5.4.1.1.8 TrackingNumberMatchMethod (Class)

This class enumerates the possible ways that tracking numbers can be checked to see if they match character input.

5.4.1.1.9 WebPermit (Class)

This class provides a GUI wrapper around an EORS permit.

5.4.1.2 EORSServletClasses (Class Diagram)

This diagram shows the classes involved in allowing users to search EORS permits.

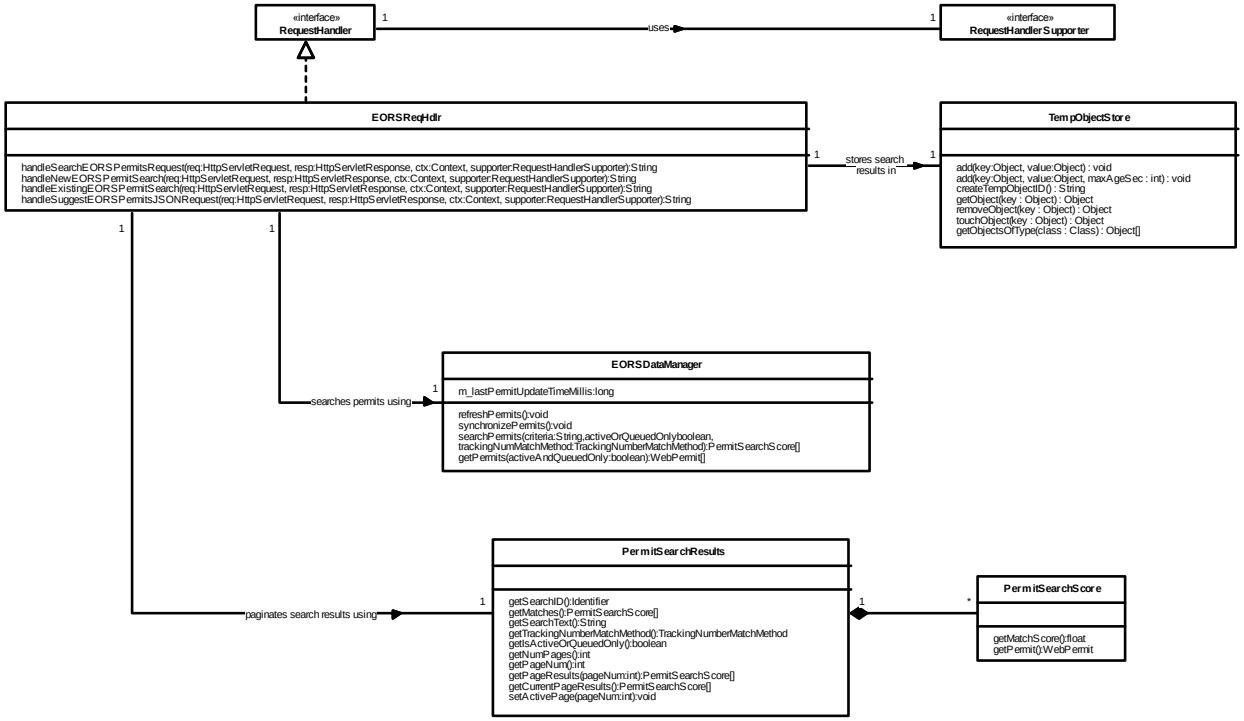


Figure 5-16 EORSServletClasses (Class Diagram)

5.4.1.2.1 EORSDataManager (Class)

This class is responsible for managing the cache of EORS permits.

5.4.1.2.2 EORSReqHdlr (Class)

This class handles the requests for suggesting and searching EORS permits.

5.4.1.2.3 PermitSearchResults (Class)

This class holds the results of a permit search for the purposes of retaining the search results and allowing the user to view them.

5.4.1.2.4 PermitSearchScore (Class)

This class is capable of holding the search score for a particular permit during a search.

5.4.1.2.5 RequestHandler (Class)

This interface specifies methods that are to be implemented by classes that are used to process requests.

5.4.1.2.6 RequestHandlerSupporter (Class)

This interface is implemented by any class that can provide access to objects or methods that are helpful to request handlers.

5.4.1.2.7 TempObjectStore (Class)

This class provides a self cleaning storage area for temporary objects.

5.4.2 Sequence Diagrams

5.4.2.1 EORSReqHdlr:handleSearchEORSPermitsRequest (Sequence Diagram)

This diagram shows the processing performed when a user performs EORS permit searching activities (either starting a new search, or paging through the results of a previously performed search). The EORSReqHdlr checks if the user can manage traffic events. If not, an error is shown. If so, the EORSReqHdlr checks for a searchID parameter. If this is null, it indicates that the user is starting a new search, so the handleNewEORSPermitSearch private method is called. If it is not null it indicates that the user is paging through the results of a prior search, so the handleExistingEORSPermitSearch private method is called. In either case, if an error is returned it is displayed for the user. If no error is returned the EORSSearchResults velocity template is returned for the user to veiw the results.

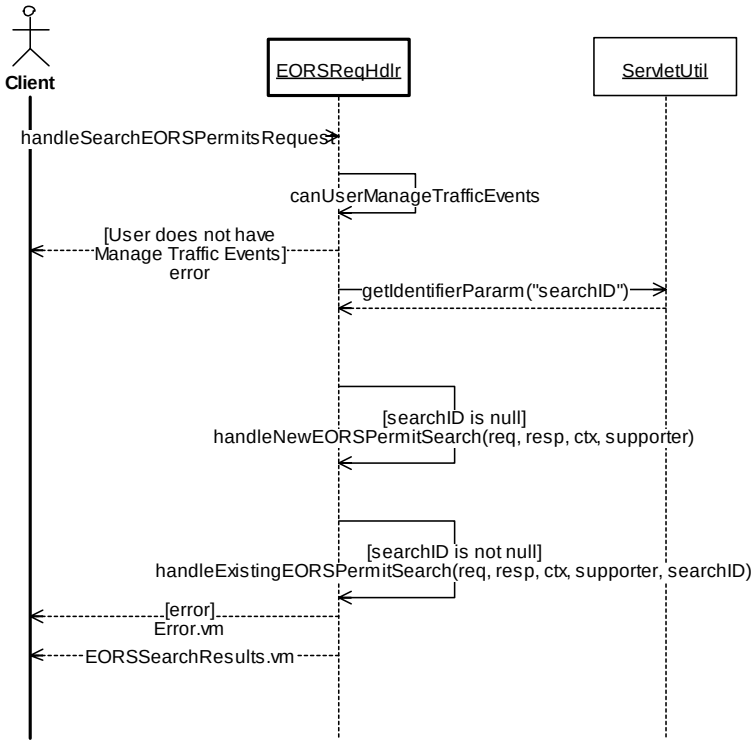


Figure 5-17 EORSReqHdlr:handleSearchEORSPermitsRequest (Sequence Diagram)

5.4.2.2 EORSReqHdlr:handleNewEORSPermitSearch (Sequence Diagram)

This diagram shows the processing performed when a user performs EORS searches in the CHART GUI. First a check is performed to see if the user has the Manage Traffic Events functional right. If the user does not have this right, an error message is returned for display. Otherwise the searchID parameter is queried from the current http request. If this parameter is missing, then this is a new EORS permit search and the EORSReqHdlr calls the handleNewEORSPermitSearch private method. If the parameter is found, the handleExistingEORSPermitSearch private method is called instead. Both methods will either throw an exception if there is an error that prevents the search from succeeding, or will return the EORSSearchResults.vm to display the results of the search.

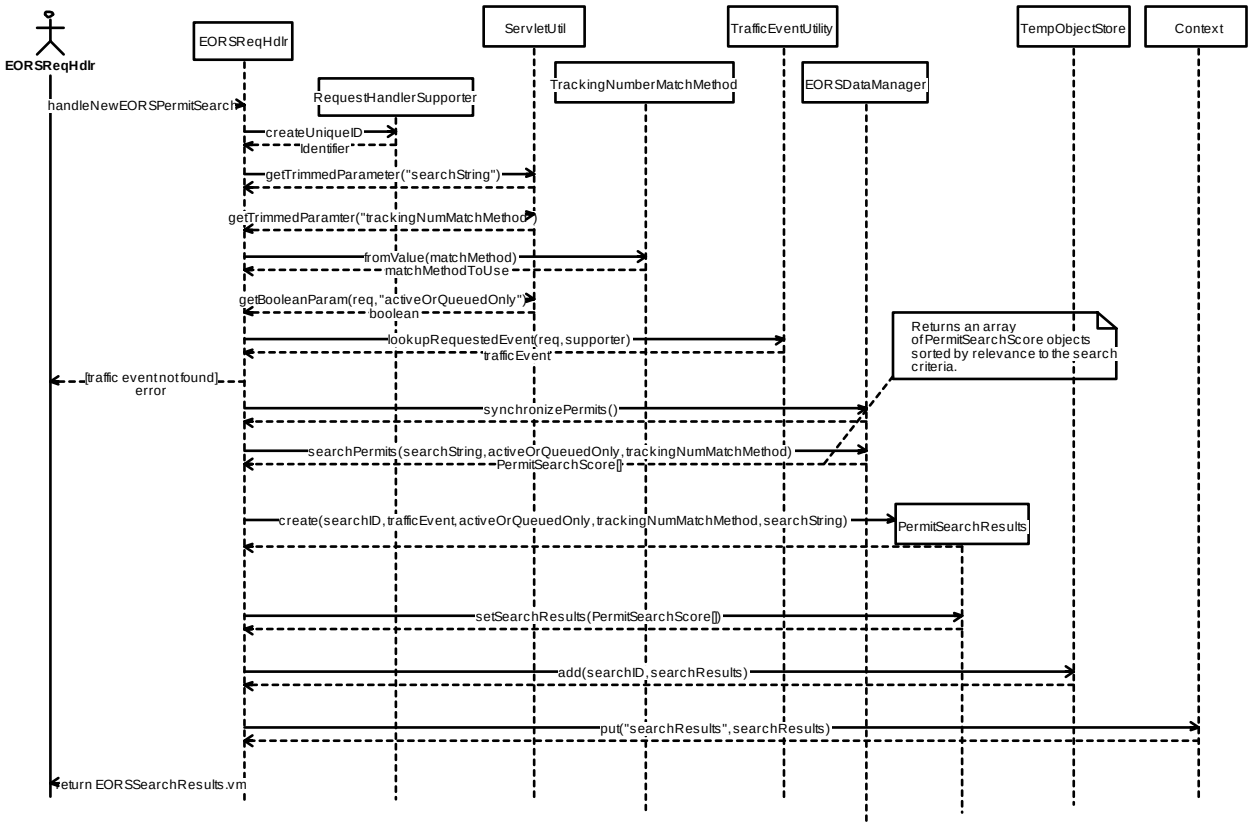


Figure 5-18 EORSReqHdlr:handleNewEORSPermitSearch (Sequence Diagram)

5.4.2.3 EORSReqHdlr:handleExistingEORSPermitSearch (Sequence Diagram)

This diagram shows the processing performed when a user chooses to change which page of results they are viewing from an EORS permit search. The request handler gets the TempObjectStore from the RequestHandlerSupporter. It then reads the searchID and pageNum parameters from the current request. Next the search results are queried (by searchID) from the TempObjectStore. If the TempObjecStore does not contain search results with the specified ID an error message is returned for display to the user. Otherwise, the PermitSearchResult is updated with the page number the user requested. The TempObjectStore is then called to touch the search results in order to keep them from timing out. Finally the PermitSearchResults are placed in the context of display by the returned EORSearchResults.vm velocity template.

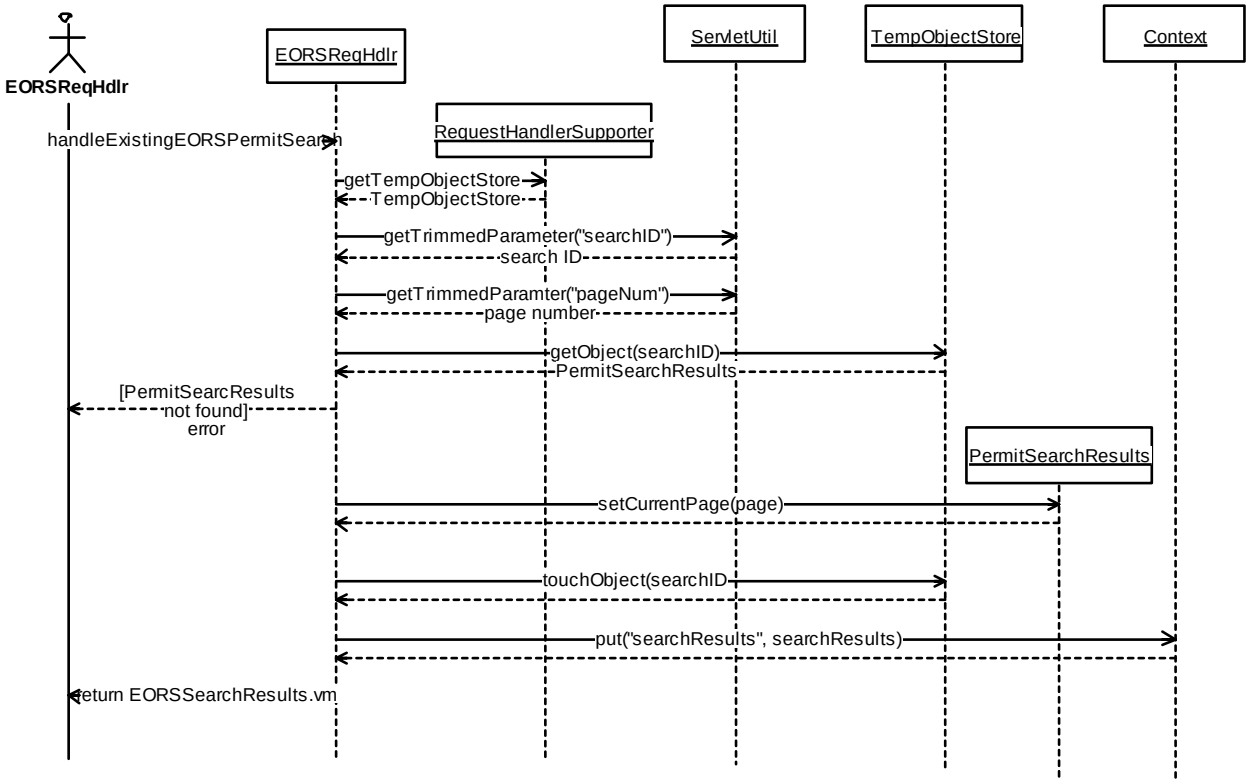


Figure 5-19 EORSReqHdlr:handleExistingEORSPermitSearch (Sequence Diagram)

5.4.2.4 EORSReqHdlr:handleSuggestEORSPermitsJSONRequest (Sequence Diagram)

This diagram shows the processing performed in order to provide suggested EORS permits when a user is typing in the search text input. This diagram shows how the servlet handles the request for suggestions and assembles a JSON response that includes the EORS permits to suggest. The EORSReqHdlr request handler gets the request from the MainServlet and gets the value of the "trackingNo", "maxResults" and "trackingNumMatchMethod" parameters. A TrackingNumberMatchMethod instance is then obtained for the specified parameter value. The GUI then reads the value of the "activeOrQueuedOnly" input parameter from the request to determine the scope of the search to perform. Next the request handler calls the EORSDataManager getPermits() method to get the list of permits to check. Before looping over the permits a JSONArray is created to hold any matching permits. Then each WebPermit is checked to see if its tracking number matches using the user requested method of matching (start or last four, last four only) If the WebPermit is a match a JSONObejct is created that represents the permit and added to the results array. Finally the response JSONObject is created, the results array is put into it, and the response JSONObject is sent.

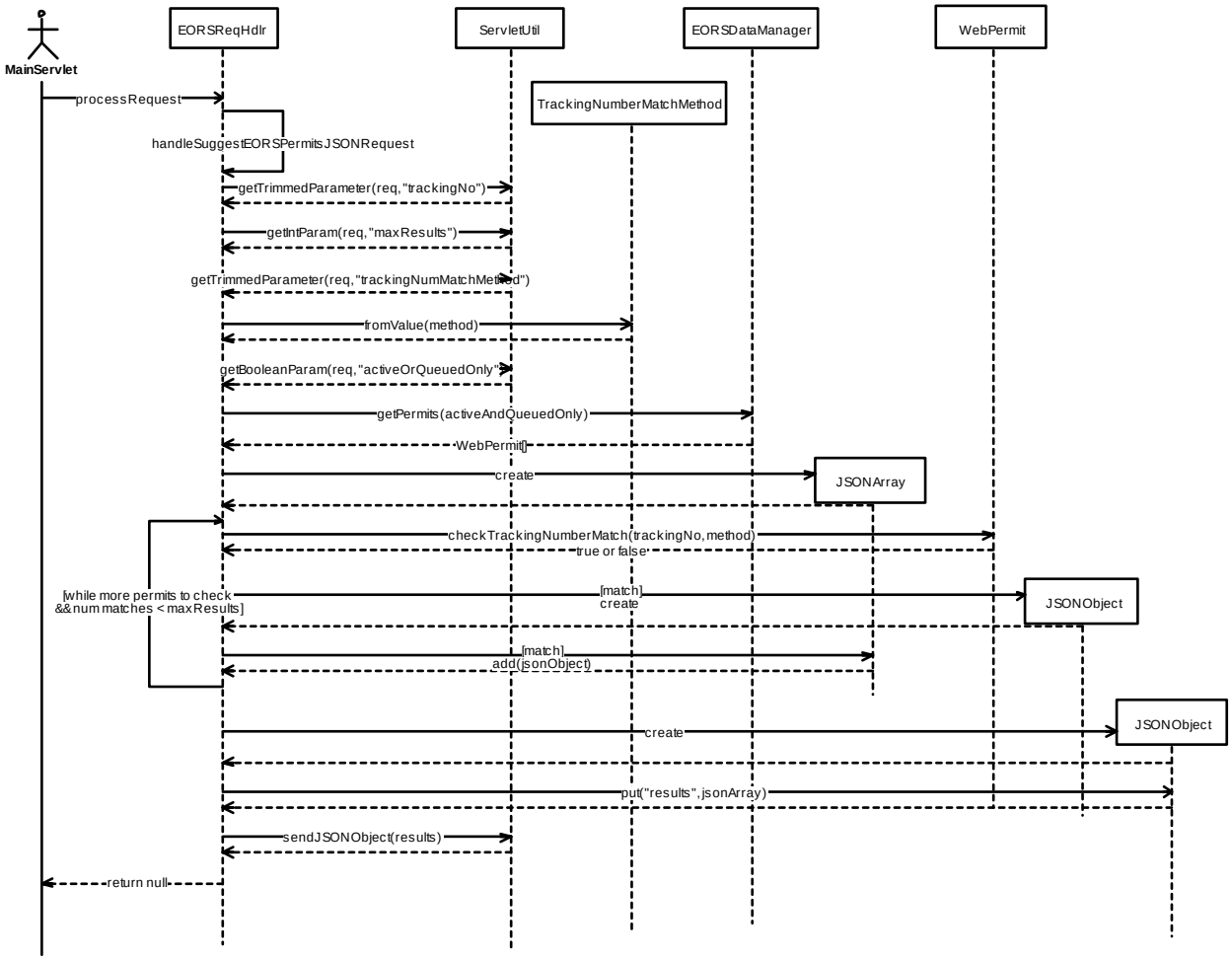


Figure 5-20 EORSReqHdlr:handleSuggestEORSPermitsJSONRequest (Sequence Diagram)

6 Use Cases – Lane Configuration

The use case diagrams depict new functionality for the CHART R6 lane configuration feature and also identify existing features that will be enhanced. The use case diagrams for this feature exist in the Tau design tool in the Release6 area. The sections below indicate the title of the use case diagrams that apply to this feature.

6.1 CHART

6.1.1 LaneConfiguration (Use Case Diagram)

This diagram shows use cases related to lane configurations and lane status.

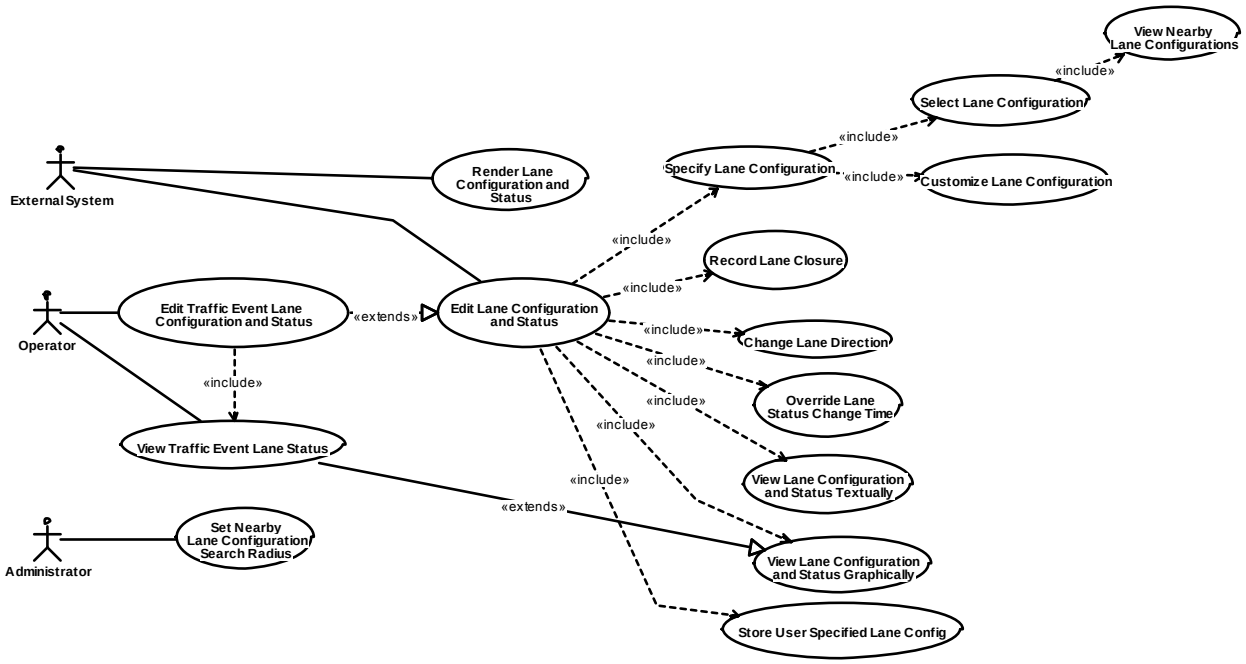


Figure 6-1 LaneConfiguration (Use Case Diagram)

6.1.1.1 Change Lane Direction (Use Case)

The lane configuration editor shall allow the user to change the traffic flow direction for a particular lane. This includes setting a lane to be multi-directional to indicate alternating use of a single lane. The editor shall allow the user to use hot keys to set the traffic flow direction. The ability to use "multi-directional" can be disabled as part of the initialization of the lane configuration editor.

6.1.1.2 Customize Lane Configuration (Use Case)

The system allows the user to customize a lane configuration after they have selected a lane configuration from the list of nearby and standard lane configurations. The user can add lanes to the configuration or remove them. The user can choose to remove all lanes so they can start "from scratch".

6.1.1.3 Edit Lane Configuration and Status (Use Case)

The system shall provide an http web service interface that allows lane configurations to be edited via a web page for use in client applications. This interface shall allow the client to initialize the lane configuration editor to customize certain aspects of the editor, and shall notify the client (via http request) when the editing session is completed so that the application can make use of the lane configuration and status as specified by the user. Customization supported during initialization of the lane configuration editor includes the following: Title for the editor; optional search criteria to be used to find nearby lane configurations for selection; optional initial lane configuration and status, to be used to pre-populate the editor when editing an existing configuration; name/value pairs to be passed back to the initiating application when the session has completed to allow application specific data to be carried through the editor; a list of optional features to be enabled (such as the ability to set a lane direction to bi-directional, and the feature that initializes lane states to unknown when a lane configuration is first selected.) Data provided to the client when the user finishes editing includes the lane configuration and status and any name/value pairs provided during initialization. As soon as the user changes any data within the lane editor, the lane editor will display a warning that their changes are not saved until they hit the submit button.

6.1.1.4 Edit Traffic Event Lane Configuration and Status (Use Case)

An operator with the manage traffic events user right may edit the lane status of a traffic event, including changing direction for a particular lane. This only applies to open Planned Roadway Closures, Incidents, and Special Events.

6.1.1.5 Override Lane Status Change Time (Use Case)

The user can override the system recorded time that indicates when a lane was opened or closed.

6.1.1.6 Record Lane Closure (Use Case)

The lane configuration editor shall allow the user to specify the status of each lane in the configuration as being open, closed, or unknown. A feature shall also exist to allow the user to set all lanes to open without having to set the status individually for each lane. Hot keys for setting lane status will be supported. The system will record the date/time a lane is opened or closed. When the lane configuration is initialized to include (not disable) the feature that sets the initial lane status in the main direction to unknown, then When a user makes the first change to the status of a lane in the main direction whose status was initially

defaulted to "unknown", the system will set the status of all other lanes in the main direction that have a status of "unknown" to "open".

6.1.1.7 Render Lane Configuration and Status (Use Case)

The system shall provide an interface to allow a lane configuration graphic to be rendered given a lane configuration and status.

6.1.1.8 Select Lane Configuration (Use Case)

The system shall allow the user to select a lane configuration that most closely resembles the roadway from a list that contains the following: - Lane configurations obtained from the mapping database that are nearby the traffic event - Lane configurations previously specified by users that are nearby the traffic event - Standard lane configurations

The list of lane configurations will allow the user to see the source of the lane configuration (mapping database, user specified, standard).

The list of lane configurations will be sorted in the following order: nearby user specified lane configurations, ordered by distance from the point specified at initialization, if any (nearest to furthest); nearby lane configurations from the mapping database, ordered by distance from the point specified at initialization, if any (nearest to furthest); standard lane configurations.

The list of lane configurations presented to the user will not include any duplicate configurations.

When a lane configuration is selected (including the case where the first one in the list is selected by default), all lanes in the same direction as the primary direction specified when the editor was initialized will be set to a status of "unknown". All lanes in the opposing direction will be set to a status of "open". This feature is optional and must be enabled as part of the initialization of the lane editor. (The CHART GUI will enable this feature when it initializes the lane editor).

6.1.1.9 Set Nearby Lane Configuration Search Radius (Use Case)

An administrator can set the radius used by the system when it searches for lane configurations that are nearby a traffic event.

6.1.1.10 Specify Lane Configuration (Use Case)

The user shall be able to specify the configuration of a roadway at a certain point, including the types of lanes included and the position of lanes relative to each other. The lane configuration editor will be initialized with the lane configuration and status provided during initialization of the editor if provided. Otherwise it will automatically pre-select the first lane configuration in the list of available lane configurations (which is sorted as specified in Select Lane Configuration).

6.1.1.11 Store User Specified Lane Config (Use Case)

When the user submits the lane configuration editor and the editor initialization included a lat/long and route, the system shall store the lane configuration as the user entered lane configuration for that location and record a timestamp of when the configuration was created. If a lane configuration already exists for that location, the newer lane configuration will replace the older lane configuration. (Note that if the lane configurations are identical this has the net effect of updating the timestamp). The timestamp is useful to allow old lane configurations to be removed from the database periodically if needed.

6.1.1.12 View Lane Configuration and Status Textually (Use Case)

The system shall provide a read-only textual description of the specified lane configuration and status.

6.1.1.13 View Lane Configuration and Status Graphically (Use Case)

A graphic shall show the type of each lane and the status of each lane (open, closed, or unknown). The direction of travel for each lane is also shown, including the current traffic flow direction. The graphic uses patterns in addition to colors to indicate lane status to make the status accessible to colorblind users.

6.1.1.14 View Nearby Lane Configurations (Use Case)

The system will include lane configurations that are nearby based on search criteria provided when the lane configuration editor is initialized, if any. Any nearby configurations found that meet the search criteria are presented for selection as part of the list of lane configurations available to the user. The nearby lane configurations will include lane configurations from the mapping database and also lane configurations that have been customized by users. The system will use the specified lat/long and a specified radius to find nearby lane configurations. When specified as part of the search criteria, the main route will be used to filter the nearby lane configurations for the specified route.

6.1.1.15 View Traffic Event Lane Status (Use Case)

The system shall show a graphic image depicting the roadway at the scene of a traffic event.

6.1.2 ManageTrafficEvents (Use Case Diagram)

This diagram models the actions that an operator may take that relate to traffic events.

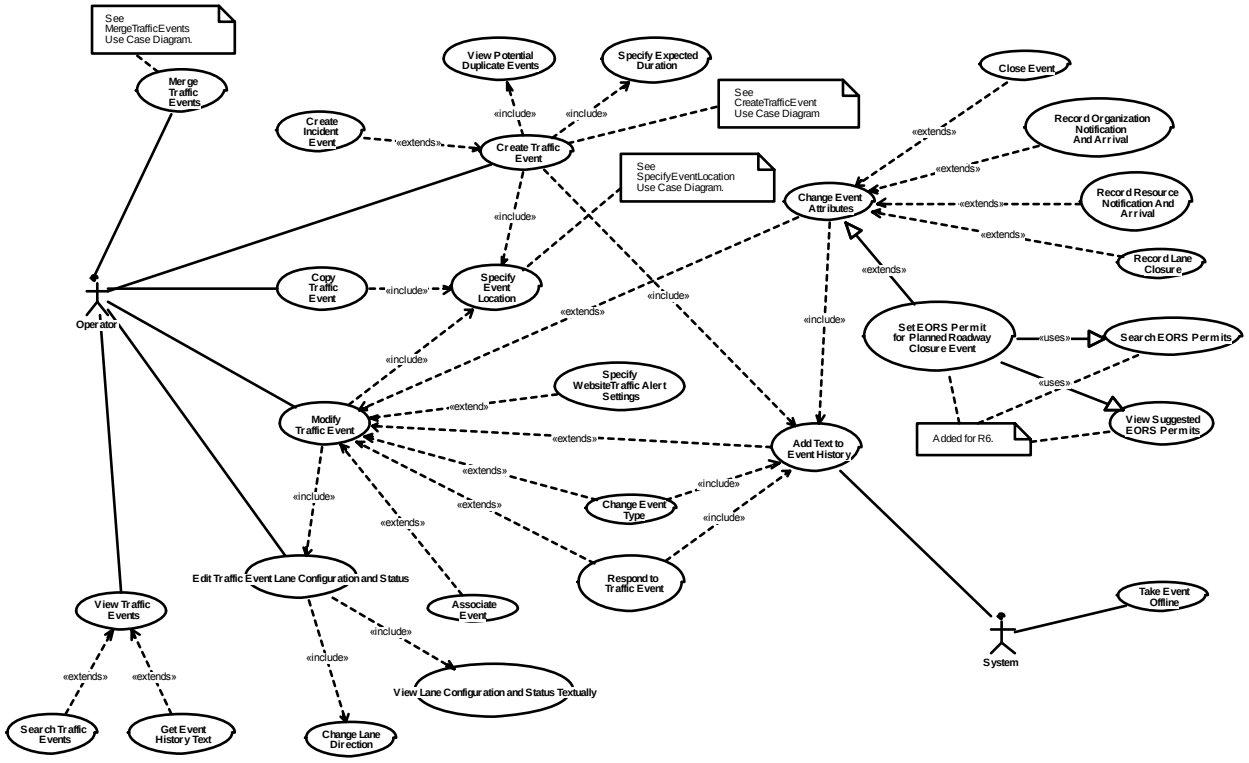


Figure 6-2 ManageTrafficEvents (Use Case Diagram)

6.1.2.1 Add Text to Event History (Use Case)

An operator with the proper functional rights may add text to a traffic event's event history.

6.1.2.2 Associate Event (Use Case)

An operator with the proper functional rights may associate related traffic events.

6.1.2.3 Change Event Attributes (Use Case)

An operator with the proper functional rights may edit traffic event information after the event has been created. This includes closing the event, adding text to the event history, recording lane closures, and recording organization and resource arrivals.

6.1.2.4 Change Event Type (Use Case)

An operator with the proper functional rights may change the traffic event type.

6.1.2.5 Change Lane Direction (Use Case)

The lane configuration editor shall allow the user to change the traffic flow direction for a particular lane. This includes setting a lane to be multi-directional to indicate alternating use of a single lane. The editor shall allow the user to use hot keys to set the traffic flow direction. The ability to use "multi-directional" can be disabled as part of the initialization of the lane configuration editor.

6.1.2.6 Close Event (Use Case)

An operator with the proper functional rights may close an event.

6.1.2.7 Copy Traffic Event (Use Case)

The user with the correct functional rights will be able to create a copy of an existing traffic event.

6.1.2.8 Create Incident Event (Use Case)

An operator with the proper functional rights may create a new incident event.

6.1.2.9 Create Traffic Event (Use Case)

The user with the correct functional rights may add a new traffic event. When creating a traffic event, the system will show the user a list of existing traffic events that may be duplicates of the new event being created based on the user's selections for the new event's location. This existing feature will be changed in R3B2 to ensure external and pending events do not appear as possible duplicate events.

6.1.2.10 Edit Traffic Event Lane Configuration and Status (Use Case)

An operator with the manage traffic events user right may edit the lane status of a traffic event, including changing direction for a particular lane. This only applies to open Planned Roadway Closures, Incidents, and Special Events.

6.1.2.11 Get Event History Text (Use Case)

An operator with the correct functional rights may view the text entries that have been added to an event.

6.1.2.12 Merge Traffic Events (Use Case)

This use case represents the merge traffic event operation. A user with manage traffic event right merges the data of two traffic events. See MergeTraffic Events use case diagram.

6.1.2.13 Modify Traffic Event (Use Case)

An operator with the proper functional rights may edit traffic event information after the

event has been created. This includes responding to the event, editing lane status, editing location, associating with another event, and specifying other event attributes.

6.1.2.14 Record Lane Closure (Use Case)

The lane configuration editor shall allow the user to specify the status of each lane in the configuration as being open, closed, or unknown. A feature shall also exist to allow the user to set all lanes to open without having to set the status individually for each lane. Hot keys for setting lane status will be supported. The system will record the date/time a lane is opened or closed. When the lane configuration is initialized to include (not disable) the feature that sets the initial lane status in the main direction to unknown, then When a user makes the first change to the status of a lane in the main direction whose status was initially defaulted to "unknown", the system will set the status of all other lanes in the main direction that have a status of "unknown" to "open".

6.1.2.15 Record Organization Notification And Arrival (Use Case)

An operator with the proper functional rights may record the participation of various organizations in the event resolution.

6.1.2.16 Record Resource Notification And Arrival (Use Case)

An operator with the proper functional rights may record the participation of various resources in the event resolution.

6.1.2.17 Respond to Traffic Event (Use Case)

The system allows an operator to control devices in response to an event through the use of a response plan. The user may add devices to the plan, select the desired state of the devices, then activate the plan. Any of the devices used by the event response plan may be deactivated while the event is open by removing the item for that device from the plan. When the event is closed, if the response plan is active, it will be deactivated automatically.

6.1.2.18 Search EORS Permits (Use Case)

The system will allow the user to perform a text search on the following fields of an EORS permit: permit tracking number, start county name, end county name, permit type, route location, route type, route number, work order description, permittee name, contract number and days of week. The system will score each matching permit based on the percentage of the user entered search terms were found in these fields and will present the results in order of relevance. The user will be shown a summary of each matching permit and will be able to show/hide additional details about each permit.

6.1.2.19 Search Traffic Events (Use Case)

An operator with the proper functional rights may search the CHART system for traffic events.

6.1.2.20 Set EORS Permit for Planned Roadway Closure Event (Use Case)

A user may set the EORS permit associated with a planned roadway closure event. Doing so will set the EORS permit tracking number into the planned roadway closure event details. The system will assist the user in finding the correct EORS permit by suggesting potential matching permits as the user types a permit tracking number. In the event that the user does not see the permit they are looking for in the list of suggested permits, the system will provide a more advanced searching capability that will search on other fields of the permit (in addition to the tracking number). The user will be able to specify if the list of permits considered for suggestion or searching should be limited to only the active and queued permits, or if all permits currently available in the EORS system should be considered.

The user will be able to indicate if permit tracking numbers should be considered to match their search terms if the permit number starts with the user entered text (only) or contains the user entered text anywhere within the permit number. Additionally the user may indicate that the permit may start with or end with (last 4 digits) the user entered text.

6.1.2.21 Specify Event Location (Use Case)

The event location choices will be populated using data from the CHART Mapping application database.

By default, MD will be selected as the state.

If the selected state is MD, the user will be required to select a predefined MD county/region. If a route is specified, the user will first select a route type from a pick list ("I", "US", or "MD") and the route type will be used to populate the list of predefined routes. To specify a route, the user will be required to select one of the predefined routes if the state is MD. If the state is not MD, the user will be able to enter a county name / region name and route number as freeform text.

If a route number is specified, the user will be able to select intersecting roads by route number or route name, or specify the state or county milepost. Additionally the user will be able to specify whether the traffic event is at, prior, or past the intersecting feature ("at" will be selected by default).

If the state is MD, the list of intersecting route numbers and names will be populated for the user as suggestions; however, the user can still specify freeform text for an intersecting route number, route name, county milepost, and state milepost even if the state is MD.

6.1.2.22 Specify Expected Duration (Use Case)

An operator with the proper functional rights may specify the expected duration of an event.

6.1.2.23 Specify WebsiteTraffic Alert Settings (Use Case)

An operator with the proper functional rights may specify whether or not the traffic event warrants a "Traffic Alert" on the public CHART web site, and may optionally provide specific alert text to be associated with the Alert.

6.1.2.24 Take Event Offline (Use Case)

The system periodically checks for closed events and takes them offline provided that a configured interval of time has elapsed since the event was closed.

6.1.2.25 View Lane Configuration and Status Textually (Use Case)

The system shall provide a read-only textual description of the specified lane configuration and status.

6.1.2.26 View Potential Duplicate Events (Use Case)

An operator with the correct functional rights will be presented with a list of potential duplicate traffic events based on the event location. The operator will have the option to then merge these events.

6.1.2.27 View Suggested EORS Permits (Use Case)

As the user types in the search field the system will present a set of suggested permits based on the permit tracking number. The suggested permits will each display the tracking number and some summary information about the permit as well as a link/button that allows the user to associate the permit to the planned closure directly from the suggestion.

6.1.2.28 View Traffic Events (Use Case)

An operator with the correct functional rights may view a traffic event.

6.2 Mapping

6.2.1 GISLaneConfigWebService (Use Case Diagram)

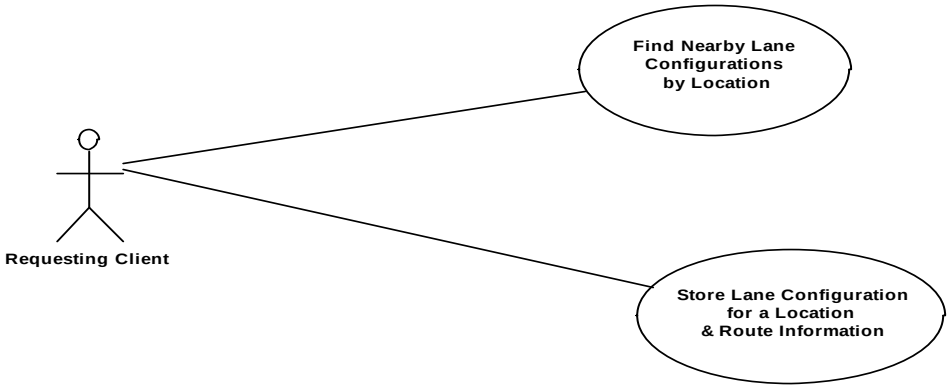


Figure 6-3 GISLaneConfigWebService (Use Case Diagram)

6.2.1.1 Find Nearby Lane Configurations by Location (Use Case)

The service will allow a requesting client to query lane configurations nearby a specified location. The client will specify a Location, Radius, and Route, if available. The system will locate lane configurations within the given radius of the location, and will filter the results by the specified route, if any.

6.2.1.2 Store Lane Configuration for a Location & Route Information (Use Case)

The service will allow the requesting client to store a lane configuration for a specified location and route. Each request will be authenticated based upon public/private key, request body, client ID, and a signature. If a lane configuration already exists for the specific location and route, the newer lane configuration will replace the older lane configuration. A timestamp will be saved for each posted lane configuration.

7 Detailed Design – Lane Configuration

7.1 Human-Machine Interface

7.1.1 Lane Configuration

7.1.1.1 Traffic Event Details Page

The Roadway Conditions section of the traffic event details page is changed to no longer allow status editing of the lane configuration directly on this page. Before a lane configuration is initially selected for the event, this section of the page looks the same as it did in R5:

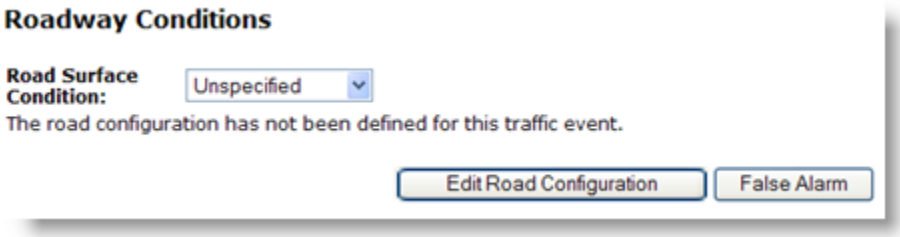


Figure 7-1 Roadway Conditions Section in Traffic Event

After a lane configuration has been specified, the lane image is shown (and per-lane tooltips are available) but the user can no longer click on lanes to set their status and direction:

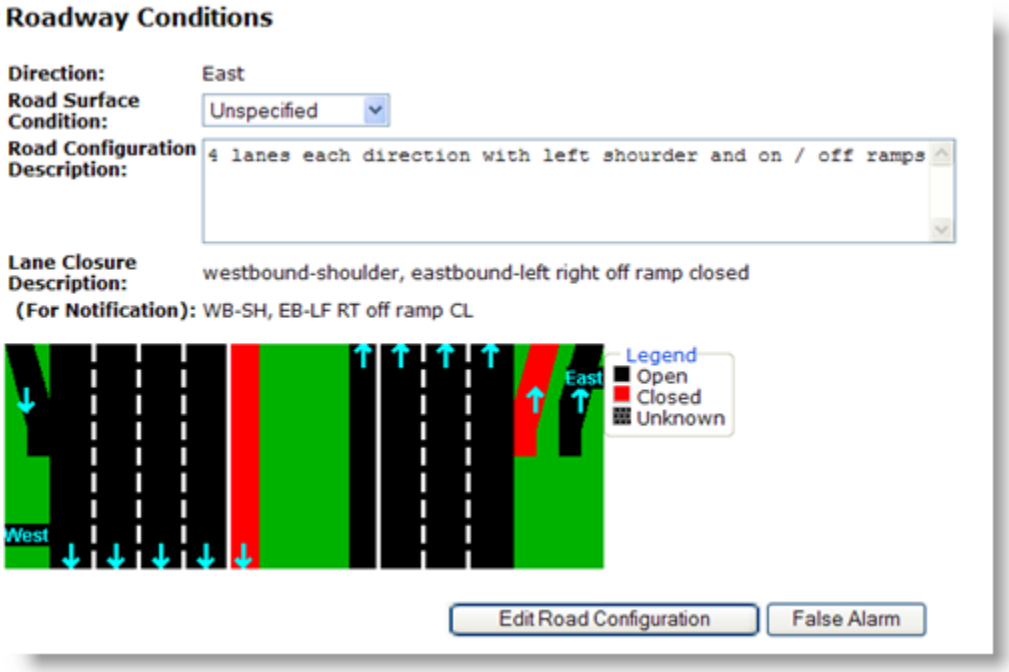


Figure 7-2 Roadway Conditions Popup

Instead, all edits to the lane configuration and status occur from the lane editor. The lane editor is accessed by clicking the ***Edit Road Configuration*** button.

7.1.1.2 Lane Editor Popup

When the user clicks the ***Edit Road Configuration*** button on the traffic event details page, the lane editor window will pop up:

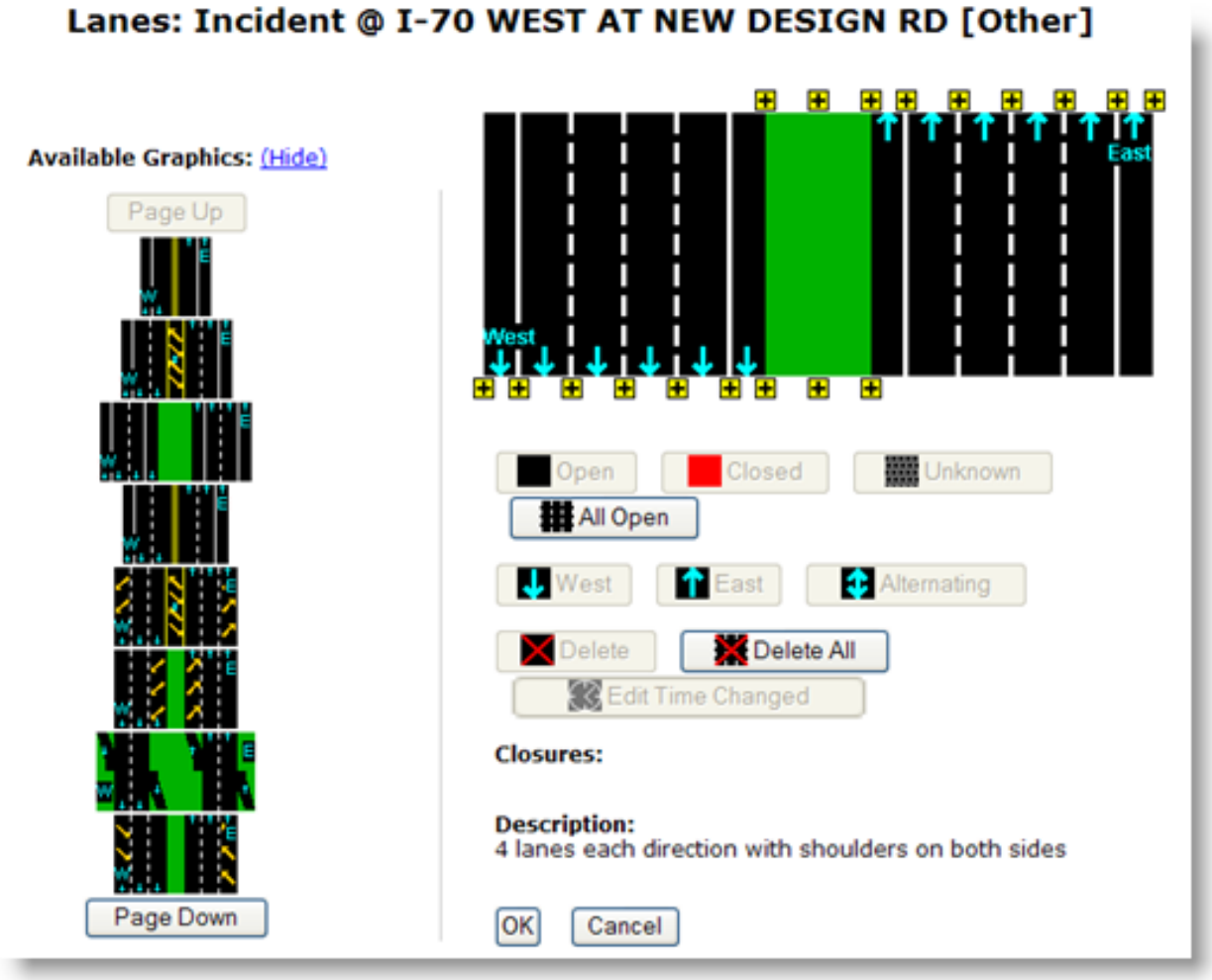


Figure 7-3 Lane Editor Popup

If a roadway configuration was not previously selected for the traffic event, the form will appear with a list of all available configurations shown on the left (see above). The available lane configurations include first any "nearby" lane configurations that are found. Nearby lane configurations are found based on the location of the traffic event; a query is made into a GIS database to find lane configurations for the roadway specified in the event location that are close to the location of the event (if any). These nearby lane configurations can be either specified by users, or part of the State's roadway database. The user specified configurations will be listed first, ordered by distance from the event, followed by configurations from the roadway database, also ordered by distance from the event. Standard (default) lane configurations, which are

always the same and are not based on proximity to the traffic event, will be included after any nearby configurations. The first configuration listed will be selected by default and its graphic will appear on the right side of the form.

If a roadway configuration was previously specified for the traffic event and the editor has been launched to change that configuration, the existing configuration will be pre-selected and its current status shown. The list of available graphics for selection will initially be hidden:

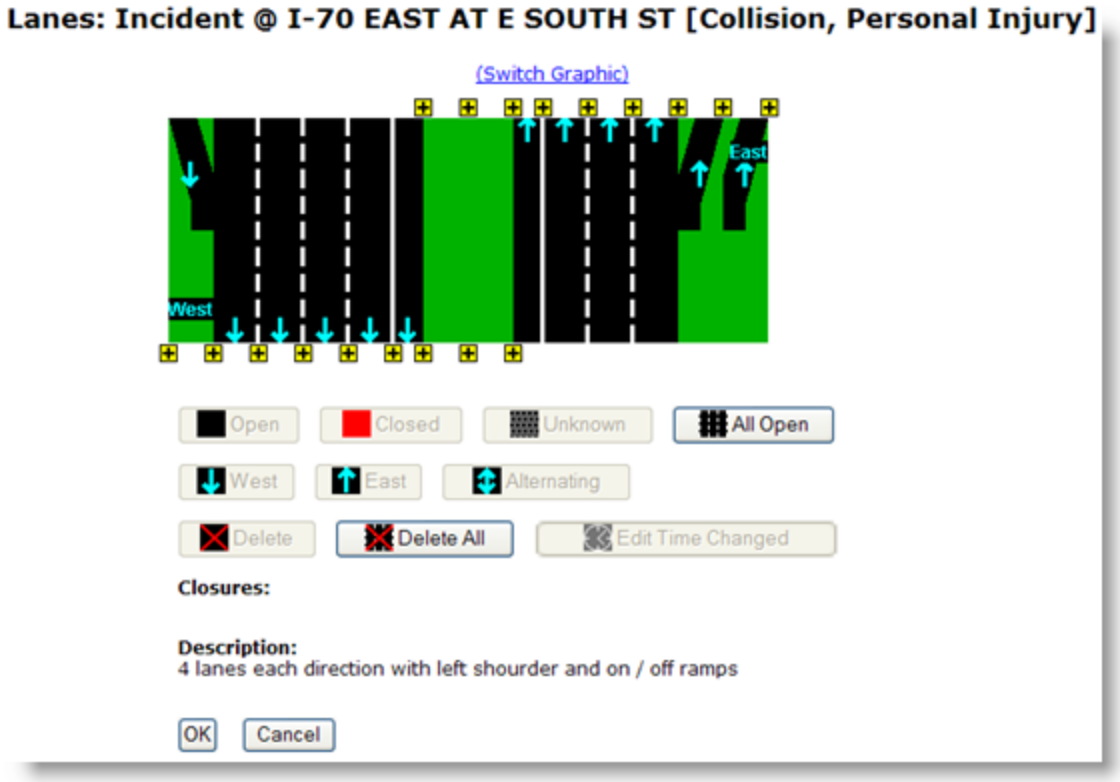


Figure 7-4 Lane Editor Popup

The ***Switch Graphic*** link can be used to cause the list of available graphics to appear.

7.1.1.2.1 Selecting a Graphic

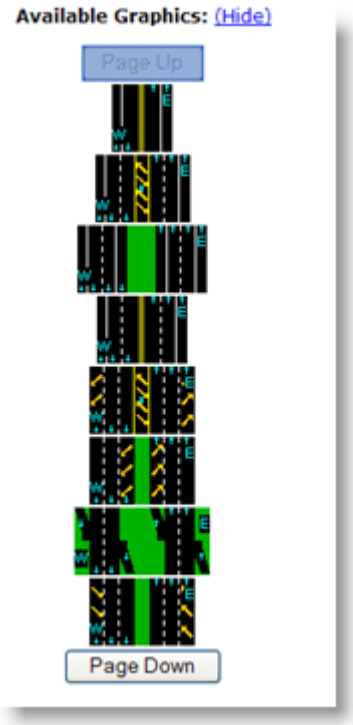


Figure 7-5 Selecting a Lane Configuration

The user can choose a different graphic to work with using the Available Graphics list. To choose a new graphic, the user can click on any of the graphics in the list. The Page Up and Page Down buttons can be used to view the prior or next page of graphics. After a graphic is selected, the Available Graphics list will be hidden, and the graphic will appear in the main area of the page and be available for edits and specifying lane status and direction.

7.1.1.2.2 Adding Lanes

The plus signs above and below the lane graphic allow a lane to be added at the location of the plus. The plus signs below the lane graphic are used to add lanes in the direction shown on the bottom of the graphic, and the plus signs above the graphic are used to add lanes in the direction shown on the top of the graphic.

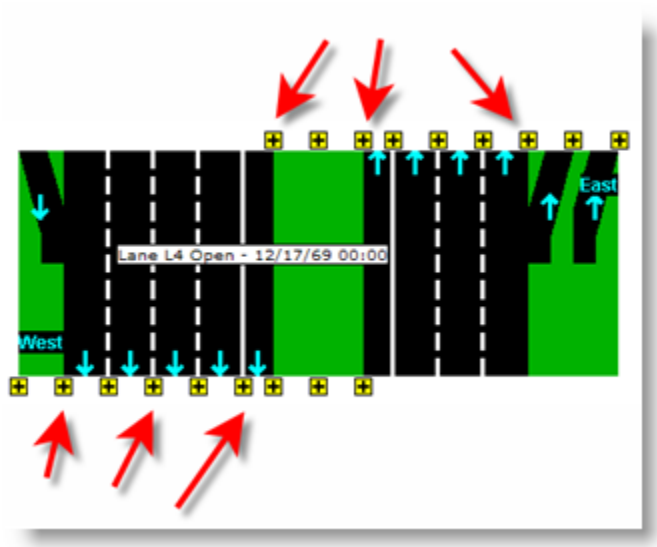


Figure 7-6 Lane Insertion Points

After clicking a plus sign, a list of available lane types appears, allowing the user to select the type of lane to be added:

Figure 7-7 Lane Types for New Lane

The lane type is selected by clicking on its link.

7.1.1.2.3 Selecting Lanes

One or more lanes can be selected (by clicking on them) to perform an operation on the lanes. Selected lanes are indicated with a transparent color change to highlight the lane. (In the graphic below, three of the westbound lanes are selected). A selected lane can be unselected by clicking on it again.

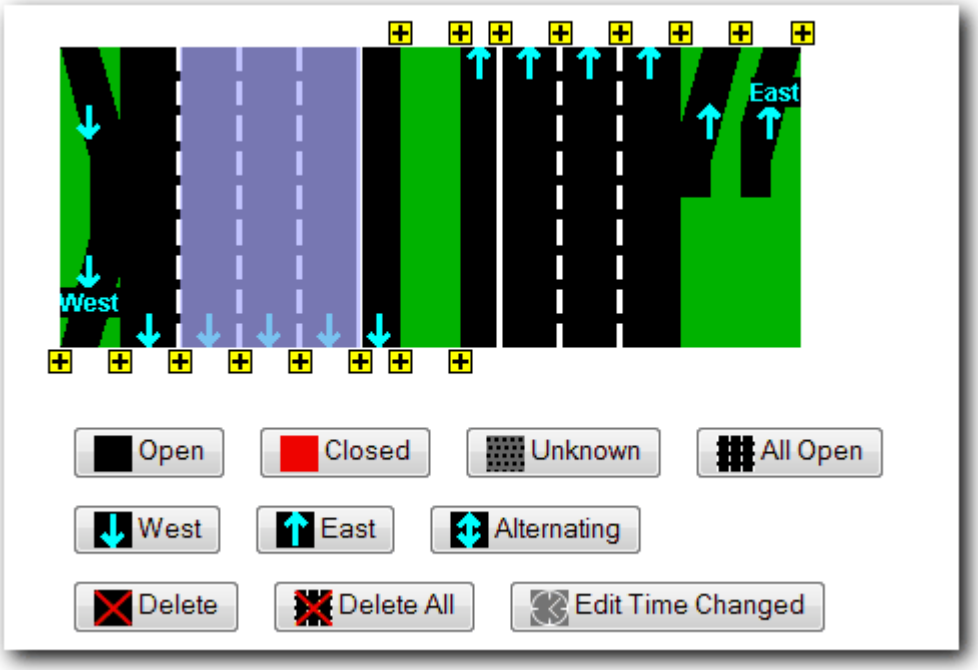


Figure 7-8 Selecting Multiple Lanes

The action buttons below the graphic automatically become enabled or disabled depending on whether the action is appropriate for the number of lanes selected:

- The lane state buttons (Open, Closed, Unknown), traffic flow direction buttons (West, East, Alternating), and the Delete button are only enabled when one or more lanes are selected.
- Edit Time Changed is only enabled if a single lane is selected
- All Open and Delete All are always enabled regardless of how many lanes are selected

7.1.1.2.4 Removing Lanes

To remove one or more lanes, the lanes are first selected as described above. Clicking on the Delete button will delete the selected lanes. (Note that changes are not saved until submitting the editor form.)

Clicking on Delete All will remove all lanes from the graphic.

If all lanes are removed from the graphic, the graphic will appear as a single line with plus signs at the top and bottom for adding lanes. At this point the user can either add single lanes using the plus signs, or can switch to a predefined lane configuration.

Figure 7-9 Empty Lane Configuration

7.1.1.2.5 Setting Lane Status

To set the status of one or more lanes, the lanes are first selected as described above. The user can then click on the Open, Closed, or Unknown buttons to set the selected lanes to the specified state.

Clicking on All Open will mark all lanes as Open. This shortcut may be helpful when a traffic event is ready to be closed.

7.1.1.2.5.1 Default Lane Status

An optional feature supported by the Lane Editor, which is used by the CHART GUI, is the ability to set the default lane status. If the editor is brought up for the first time for a traffic event, or if the user switches to a new lane configuration graphic, the lanes in the primary direction (i.e., the direction of the traffic event) are set to Unknown status while the lanes in the non-primary direction are set to Open. This feature allows an operator to define the road configuration before the real lane status is known, which helps avoid confusion that might be caused by marking the lanes as either Open or Closed before the real status is known.

For example, the picture below shows an example of lanes being set to their default status for this lane configuration, given that the primary (traffic event) direction is West:

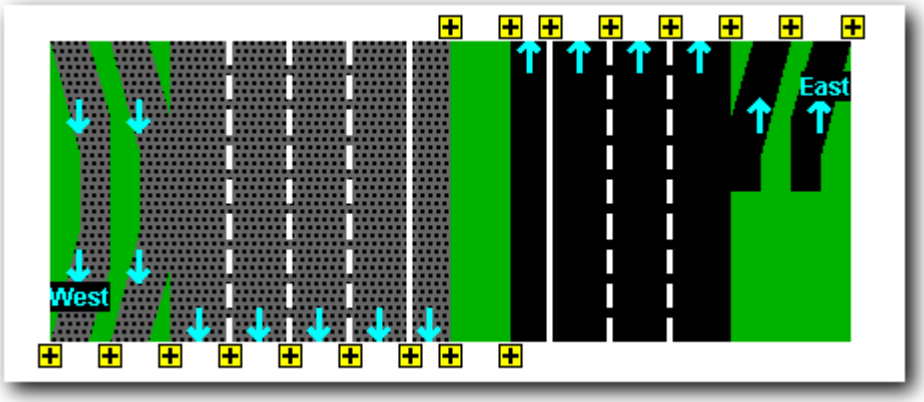


Figure 7-10 CHART Default Status for Lane Editor

As soon as the user marks any lane in the primary direction as either Open or Closed it is assumed that the user knows the status of the lanes, so at that point the default status is over and all of the Unknown lanes in the primary direction are set to Open.

7.1.1.2.6 Setting Lane Direction

To set the traffic flow direction of one or more lanes, the lanes are first selected as described above. The user can then click on the direction buttons (e.g., East, West, North, South, Inner Loop, or Outer Loop, or Alternating) to set the selected lanes' traffic flow. (Alternating traffic flow indicates the use of a single lane for both directions in an alternating fashion, using traffic-controlling mechanisms such as a flagging operation or stop lights.) The ability to set a lane direction to Alternating is an optional feature of the lane editor used by the CHART GUI.

When the direction is changed, the blue direction arrows will change to indicate the new direction.

7.1.1.2.7 Editing Status Change Time

To set the time when a lane's status really changed (as opposed to when the user changed the status), a single lane is selected and the Edit Time Changed button is clicked. This brings up a small popup:

Figure 7-11 Setting Lane State Changed Time

Clicking on the calendar icon brings up a more complex control that allows the user to choose the date and time:

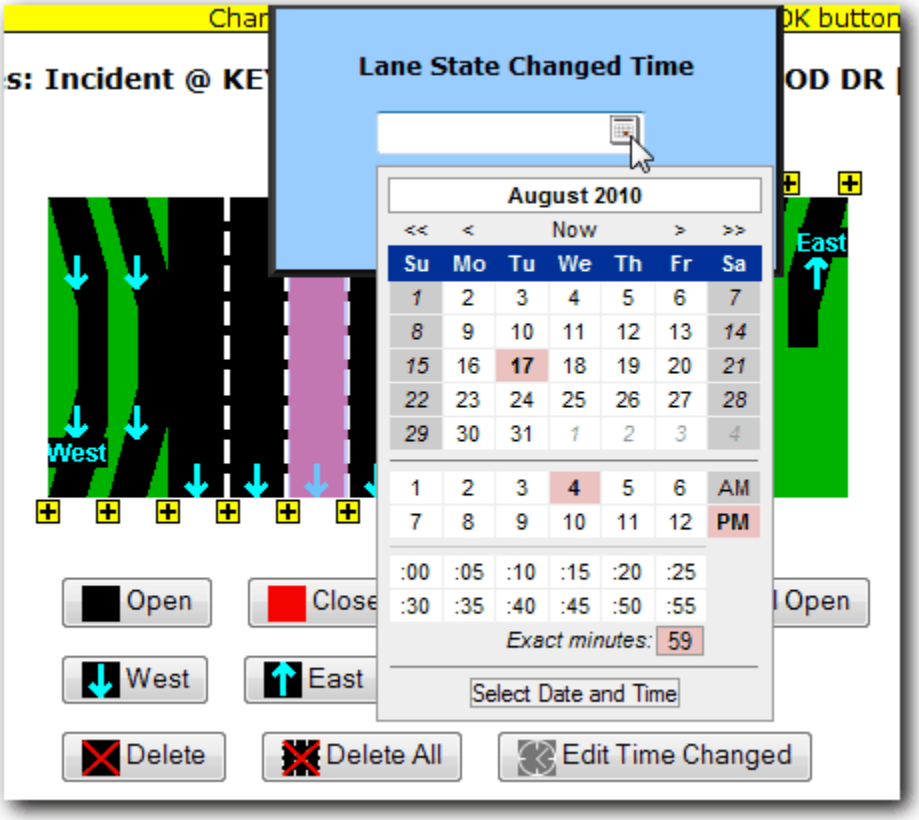


Figure 7-12 Setting Lane State Changed Time (Advanced)

When done editing the date/time on the calendar control, the user clicks Select Date and Time, which populates the date/time into the Lane State Changed Time pop up's text field. Then the user clicks Submit to apply the new time to the lane.

7.1.1.2.8 Textual Status

The Lane Editor will display a description of the lane closure status, and a description of the lane configuration. These descriptions will be automatically updated as changes are made.

An example is shown below:

Closures:
1/4 westbound-center left traffic lane closed

Description:
4 westbound and 3 eastbound traffic lanes, with shoulders, right on ramps, right off ramps, and median.

Figure 7-13 Textual Lane Status

7.1.1.2.9 Changes Not Saved Warning

When the user makes any changes in the lane editor, a warning message (shown below) appears to warn them that their changes are not saved until they submit the form.

Changes will not be saved until you click the OK button.

Figure 7-14 Changed Not Saved Warning

This message appears at the top and bottom of the lane editor as soon as the user makes a change to the lane configuration or status.

7.2 System Interfaces

The class diagrams in this section describe the CORBA interface classes and relationships that are being added or modified to support the Lane Configuration feature.

7.2.1 Class Diagrams

7.2.1.1 TrafficEventManagement (Class Diagram)

This class diagram contains all classes relating to Traffic Events

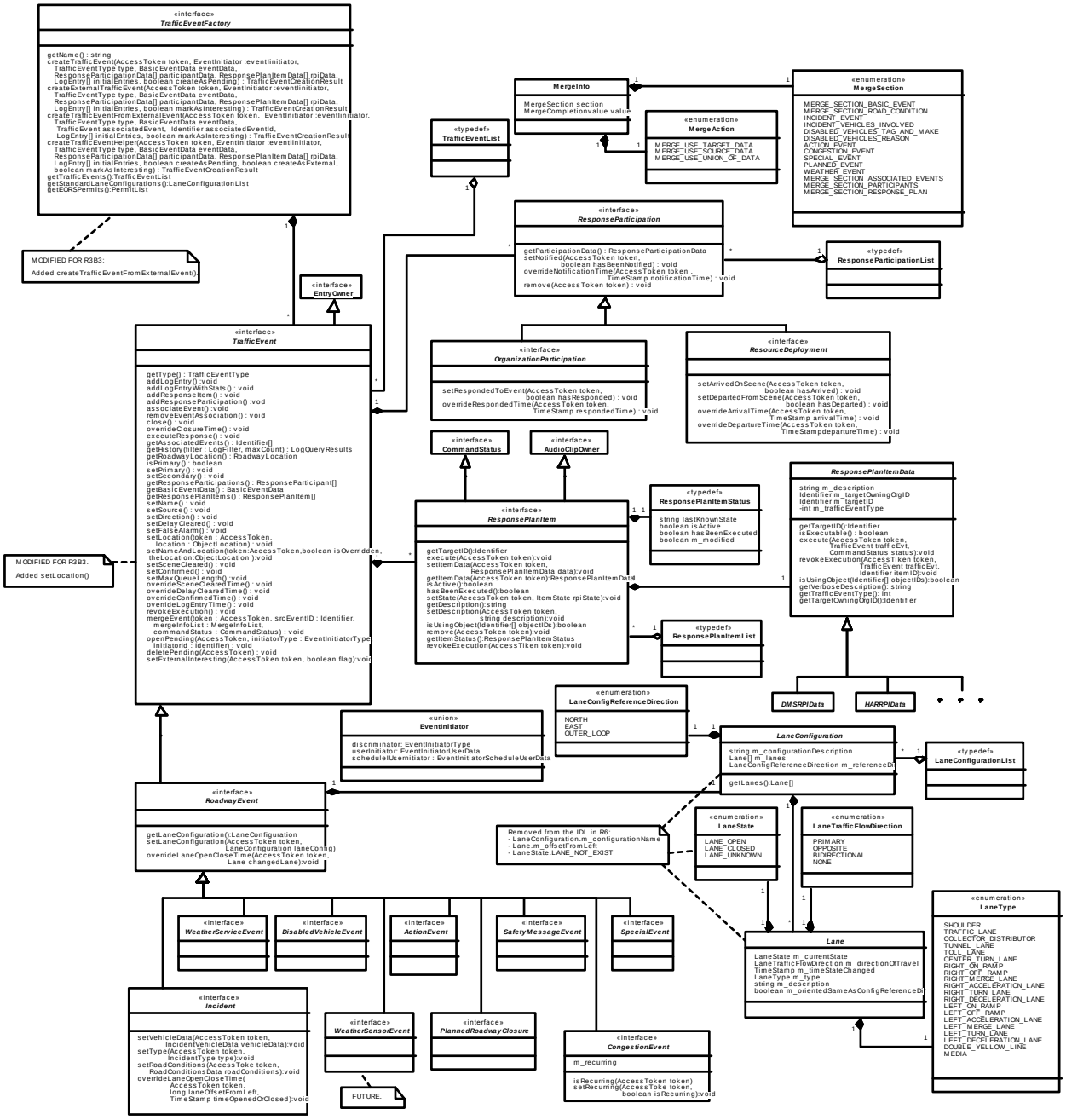


Figure 7-15 TrafficEventManagement (Class Diagram)

7.2.1.1.1 ActionEvent (Class)

This class models roadway events that require an operations center to take action but do not fit well into the other event categories. An example of this type of event would be debris in the roadway.

7.2.1.1.2 AudioClipOwner (Class)

This interface allows the AudioClipManager to check whether there are any parties interested in an audio clip. If no AudioClipOwners claim interest in a clip, the clip can be deleted.

7.2.1.1.3 CommandStatus (Class)

The CommandStatus CORBA interface is used to allow a calling process to be notified of the progress of a long-running asynchronous operation. This is normally used when field communications are involved to complete a method call. The most common use is to allow a GUI to show the user the progress of an operation. It can also be used and watched by a server process when it needs to call on another server process to complete an operation. The long running operation typically calls back to the CommandStatus object periodically as the command is being executed, to provide in-progress status information, and it always makes a final call to the CommandStatus when the operation has completed. The final call to the CommandStatus from the long running operation indicates the success or failure of the command.

7.2.1.1.4 CongestionEvent (Class)

This class models roadway congestion which may be tagged as recurring or non-recurring through the use of an attribute.

7.2.1.1.5 DisabledVehicleEvent (Class)

These class models disabled vehicles on the roadway.

7.2.1.1.6 DMSRPIData (Class)

The DMSRPIData class is an abstract class which describes a response plan item for a DMS. It contains the unique identifier of the DMS to contain the DMSMessage, and the DMSMessage itself.

7.2.1.1.7 EntryOwner (Class)

Interface which must be implemented by any class which is responsible for putting an ArbQueueEntry on a device's arbitration queue. This validate method of this interface can be called by the device to determine continued validity of the entry (either during recovery or as a final check of the validity of an entry before putting its message on the device).

7.2.1.1.8 EventInitiator (Class)

This union contains information about the entity or entities involved in the initiation of a traffic event. This can be the schedule, if a schedule was involved in initiating the event, and/or a user, if a user was involved in initiating the event. This union allows for possible expansion in future releases, where traffic events may be initiated by a schedule without user confirmation, or by CHART devices (traffic sensors, weather sensors, etc.) or external interfaces (RITIS, etc.) initially with, or possibly later without, user involvement.

7.2.1.1.9 HARRPIData (Class)

This class represents an item in a traffic event response plan that is capable of issuing a command to put a message on a HAR when executed. When the item is executed, it adds an ArbQueueEntry to the specified HAR, which stores the entry in its MessageQueue. When the item's execution is revoked, or the item is removed from the response plan (manually or implicitly through closing the traffic event) the item asks the HAR to remove the entry. The HARRPIData object also allows specification of a subset (0 to all) of the HARNotifier devices (SHAZAM or DMS devices acting as SHAZAMs) to be activated if and while the message is being broadcast on the HAR.

7.2.1.1.10 Incident (Class)

This class models objects representing roadway incidents. An incident typically involves one or more vehicles and roadway lane closures.

7.2.1.1.11 Lane (Class)

This class represents a single traffic lane at the scene of a Roadway Event.

7.2.1.1.12 LaneConfigReferenceDirection (Class)

This enumeration restricts the possible reference directions for a lane configuration, which is necessary because the lane offsets are defined relative to the "left" side, which is an ambiguous term. For example, if the direction is North then "left" to the West, but if the direction is South (also valid on a North-South roadway) then "left" could be considered (if not for this enumeration) to East. Thus if the direction of the lane config were to change from North to South, the lanes would "flip" unintentionally. This enumeration holds the reference direction for a North-South roadway to always be to the West (regardless of whether the direction of the event is North or South), and holds similarly for East-West roadways and beltways (Inner-Outer loops).

7.2.1.1.13 LaneConfiguration (Class)

This class contains data that represents the configuration of the lanes.

7.2.1.1.14 LaneConfigurationList (Class)

A collection of LaneConfiguration objects.

7.2.1.1.15 Lane State (Class)

This enumeration lists the possible states that a traffic lane may be in.

7.2.1.1.16 LaneTrafficFlowDirection (Class)

Defines the possible directions of traffic flow, relative to the lane orientation.

7.2.1.1.17 Lane Type (Class)

This enumeration lists the types of lanes.

7.2.1.1.18 Merge Action (Class)

This enumeration specifies how to merge a section of data during a traffic event merge operation.

7.2.1.1.19 Merge Info (Class)

This value type is passed between Chartlite to Chart to provide instructions for performing the merge

7.2.1.1.20 Merge Section (Class)

This idl enum defines values for each merge section

7.2.1.1.21 Organization Participation (Class)

This class is used to manage the data captured when an operator notifies another organization of a traffic event.

7.2.1.1.22 PlannedRoadwayClosure (Class)

This class models planned roadway closures such as road construction. This interface will be expanded in future releases to include interfacing with the EORS system.

7.2.1.1.23 Resource Deployment (Class)

This class is used to store the data captured when an operator deploys resources to the scene of a traffic event.

7.2.1.1.24 Response Participation (Class)

This interface represents the involvement of one particular resource or organization in response to a particular traffic event.

7.2.1.1.25 ResponseParticipationList (Class)

A collection of Response Participation objects.

7.2.1.1.26 ResponsePlanItem (Class)

Objects of this type can be executed as part of a traffic event response plan. A ResponsePlanItem can be executed by an operator, at which time it becomes the responsibility of the System to activate the item on the Response Device as soon as it is appropriate.

7.2.1.1.27 ResponsePlanItemData (Class)

This class is a delegate used to perform the execute and remove tasks for the response plan item. Derived classes of this base class have specific implementations for the type of device the response plan item is used to control.

7.2.1.1.28 ResponsePlanItemList (Class)

A collection of ResponsePlanItem objects.

7.2.1.1.29 ResponsePlanItemStatus (Class)

This structure contains data that describes the current state of a response plan item.

7.2.1.1.30 Roadway Event (Class)

This class models any type of incident that can occur on a roadway. This point in the hierarchy provides a break off point for traffic event types that pertain to other modals.

7.2.1.1.31 SafetyMessageEvent (Class)

This type of event is created by an operator when he/she would like to send a safety message to a device.

7.2.1.1.32 Special Event (Class)

This class models special events that affect roadway conditions such as a concert or professional sporting event.

7.2.1.1.33 TrafficEvent (Class)

Objects of this type represent traffic events that require action from system operators.

7.2.1.1.34 TrafficEventFactory (Class)

This interface is supported by objects that are capable of creating traffic event objects in the system.

7.2.1.1.35 TrafficEventList (Class)

A collection of TrafficEvent objects.

7.2.1.1.36 WeatherSensorEvent (Class)

This class models roadway weather events such as snow or fog that are reported by the system's weather monitoring devices. Operators will need to manually enter the information in these events for this release. In future releases, these events will be automatically generated by the system.

7.2.1.1.37 WeatherServiceEvent (Class)

This class models roadway weather events such as snow or fog that are manually entered by an operator in response to receiving an alert from the national weather service.

7.3 Lane Config Utility Package

7.3.1 Class Diagrams

7.3.1.1 LaneConfigModelClasses (Class Diagram)

This diagram contains classes used for modeling a lane configuration. The classes mirror those defined in the TrafficEventManagement IDL, but do not use the IDL generated classes.

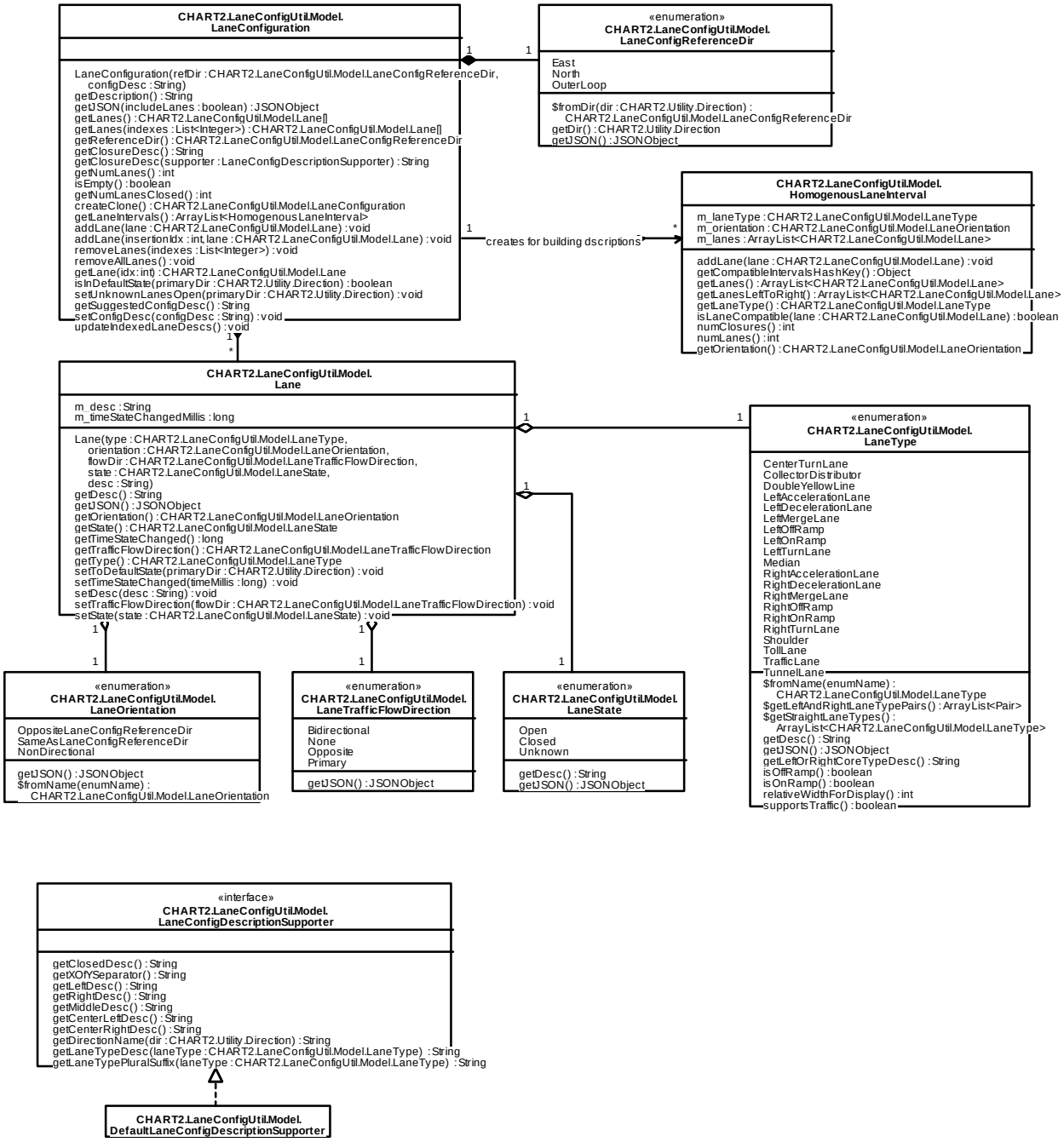


Figure 7-16 LaneConfigModelClasses (Class Diagram)

7.3.1.1.1 CHART2.LaneConfigUtil.Model. DefaultLaneConfigDescriptionSupporter (Class)

This class will provide a default implementation of the LaneConfigDescriptionSupporter interface, and will produce non-abbreviated lane configuration and lane closure descriptions.

7.3.1.1.2 CHART2.LaneConfigUtil.Model. HomogenousLaneInterval (Class)

This class represents a contiguous block of lanes within the lane configuration having the same orientation and lane type. It is useful for performing analysis on the lanes, such as when building the lane closure description.

7.3.1.1.3 CHART2.LaneConfigUtil.Model. Lane (Class)

This class mirrors the Lane structure in the IDL. It is used to model a lane without relying on the IDL.

7.3.1.1.4 CHART2.LaneConfigUtil.Model. LaneConfigDescriptionSupporter (Class)

This interface allows for some customization of the lane configuration description and lane closure description algorithms. This type of customization is currently used for notification messages, which require abbreviated descriptions.

7.3.1.1.5 CHART2.LaneConfigUtil.Model. LaneConfigReferenceDir (Class)

This enumeration mirrors the LaneConfigReferenceDirection structure in the IDL. It is used to enumerate the lane config reference dir without relying on the IDL.

7.3.1.1.6 CHART2.LaneConfigUtil.Model. LaneConfiguration (Class)

This class mirrors the LaneConfiguration structure in the IDL. It is used to model a lane configuration without relying on the IDL.

7.3.1.1.7 CHART2.LaneConfigUtil.Model. LaneOrientation (Class)

This enumeration mirrors the LaneOrientation structure in the IDL. It is used to model lane orientation without relying on the IDL.

7.3.1.1.8 CHART2.LaneConfigUtil.Model. LaneState (Class)

This enumeration mirrors the LaneState structure in the IDL. It is used to model lane state without relying on the IDL.

7.3.1.1.9 CHART2.LaneConfigUtil.Model. LaneTrafficFlowDirection (Class)

This enumeration mirrors the LaneTrafficFlowDirection structure in the IDL. It is used to model lane traffic flow direction without relying on the IDL.

7.3.1.1.10 CHART2.LaneConfigUtil.Model. LaneType (Class)

This enumeration mirrors the LaneType structure in the IDL. It is used to model lane type without relying on the IDL.

7.3.1.2 LaneDisplayClasses (Class Diagram)

This diagram shows classes involved in rendering a lane configuration graphic and

providing access to metadata for the image.

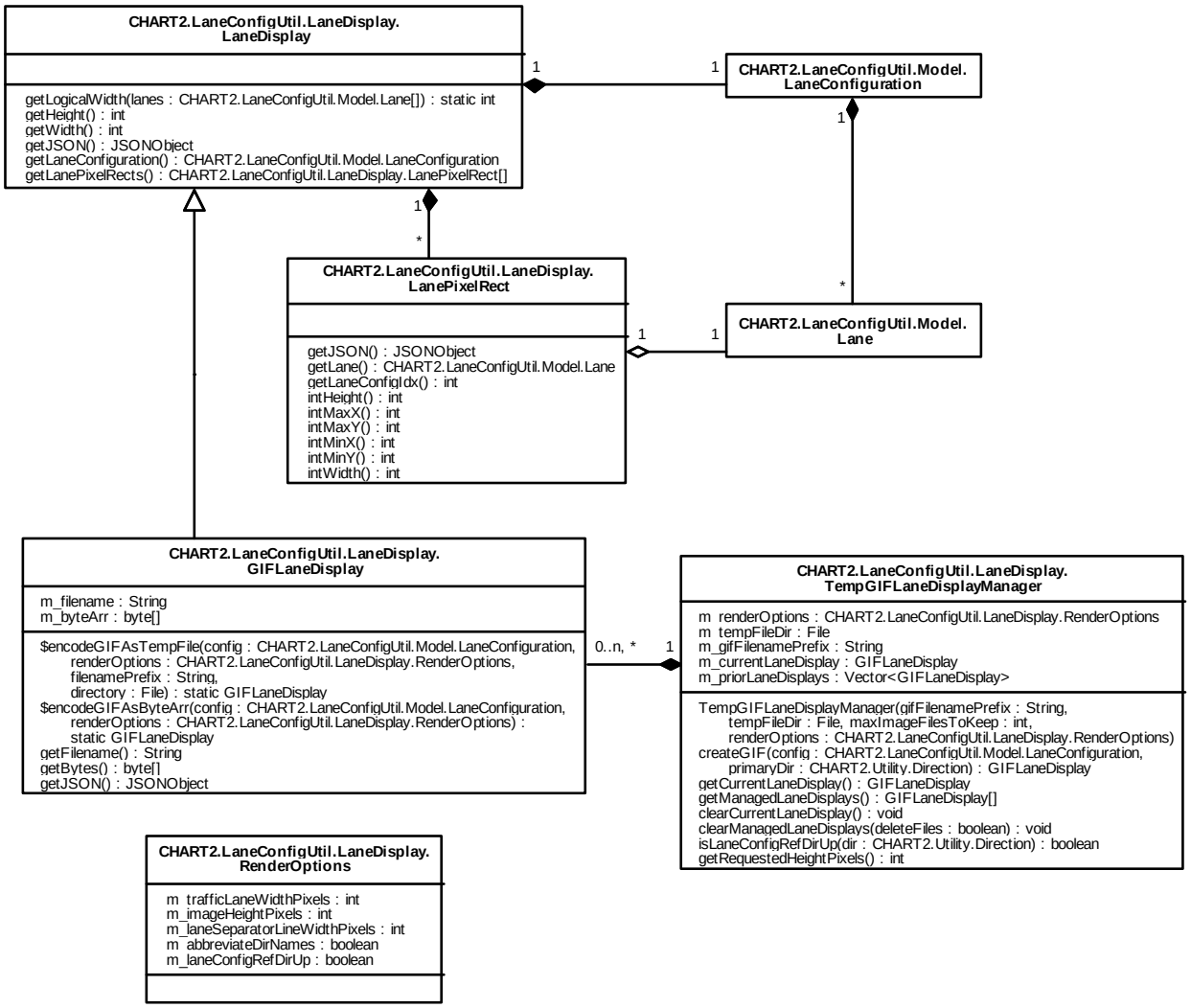


Figure 7-17 LaneDisplayClasses (Class Diagram)

7.3.1.2.1 CHART2.LaneConfigUtil.LaneDisplay. GIFLaneDisplay (Class)

This class represents a Lane Display image represented by GIF-encoded data. The data is typically stored in a file (in which case the filename will not be empty); however, the data can instead be stored in memory in a byte array (in which case the filename will be empty).

7.3.1.2.2 CHART2.LaneConfigUtil.LaneDisplay. Lane Display (Class)

This class is used to render a given LaneConfiguration object, and to provide a data model of the rendered graphic that can be used to identify lane attributes within the graphic. If the configuration changes (i.e., lanes are added or removed) a new Lane Display object will be needed to model it.

7.3.1.2.3 CHART2.LaneConfigUtil.LaneDisplay. LanePixelRect (Class)

This class represents data about a lane that is rendered in a lane display graphic. It contains the bounding rectangle of the lane, and has a reference to the Lane data itself.

7.3.1.2.4 CHART2.LaneConfigUtil.LaneDisplay. Render Options (Class)

This class contains options for rendering lane configuration images.

7.3.1.2.5 CHART2.LaneConfigUtil.LaneDisplay. TempGIFLaneDisplayManager (Class)

This class is used to manage a set of temporary GIF file(s) representing a single lane configuration display or a single part of the roadway (which can support changing lane configurations). This stores information for previously rendered images also, to ensure that images are still available if referenced by a filename that is slightly out of date (for example, a web page that has not been updated recently may display an older version of the image). Usually, however, it is only the most current image that is of interest. The temporary files will be set to be deleted when the process exits, or the application may want to delete them sooner. The getManagedLaneDisplays() method allows the application's cleanup code to avoid prematurely deleting the images, as they are still in use. The lane graphics will be drawn horizontally, but the orientation of the entire graphic (i.e., which direction is pointing up vs. down) can be specified to some extent. If the keepPimaryDirUp parameter is true, the primary direction will always be shown as up (or, if it's a combination direction, the lane config reference direction will be "up"). If keepPrimaryDirection is false, the lane config reference direction (N, E, or Outer Loop) will always point up.

7.3.1.2.6 CHART2.LaneConfigUtil.Model. Lane (Class)

This class mirrors the Lane structure in the IDL. It is used to model a lane without relying on the IDL.

7.3.1.2.7 CHART2.LaneConfigUtil.Model. LaneConfiguration (Class)

This class mirrors the LaneConfiguration structure in the IDL. It is used to model a lane configuration without relying on the IDL.

7.3.1.3 LaneConfigIDLUtilClasses (Class Diagram)

This diagram contains classes for dealing with IDL-based lane configuration data.

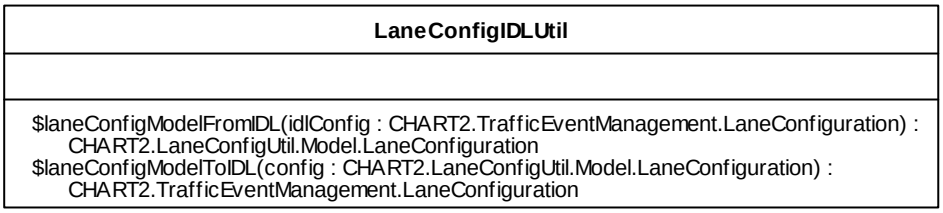


Figure 7-18 LaneConfigIDLUtilClasses (Class Diagram)

7.3.1.3.1 LaneConfigIDLUtil (Class)

This class provides lane configuration functionality related to the definitions in the IDL.

7.4 Utility Package

7.4.1 Class Diagrams

7.4.1.1 UtilityClasses3 (Class Diagram)

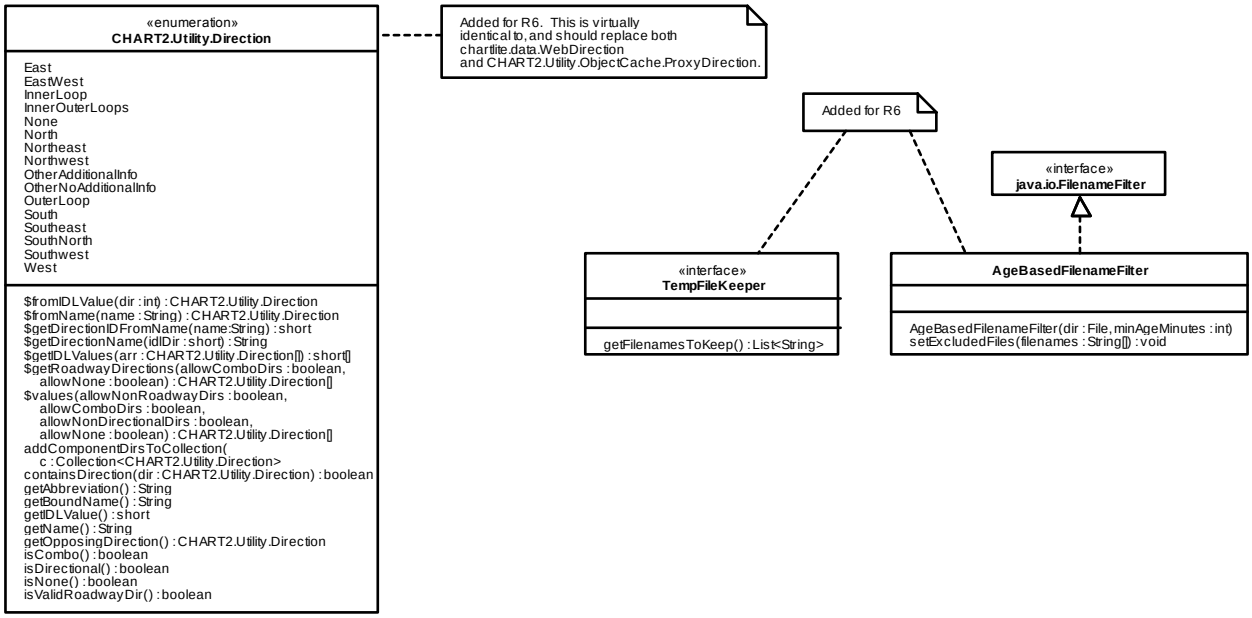


Figure 7-19 UtilityClasses3 (Class Diagram)

7.4.1.1.1 AgeBasedFilenameFilter (Class)

This class allows files that are older than the specified age to be accepted by the filter. This would typically be used to delete old files. Filenames to keep can be set into the filter, and these will not be accepted by the filter.

7.4.1.1.2 CHART2.Utility.Direction (Class)

This is an enumeration that mirrors the Common::Direction IDL enumeration, but provides additional functionality.

7.4.1.1.3 java.io.FilenameFilter (Class)

This interface is used to filter files by name.

7.4.1.1.4 TempFileKeeper (Class)

This interface allows a class to "own" temporary files for the purpose of preventing them from being deleted by periodic cleanup functionality.

7.5 chartlite.servlet.trafficevents

7.5.1 Class Diagrams

7.5.1.1 chartlite.servlet.trafficevents_classes (Class Diagram)

This diagram shows the various classes that are used to handle requests related to traffic events.

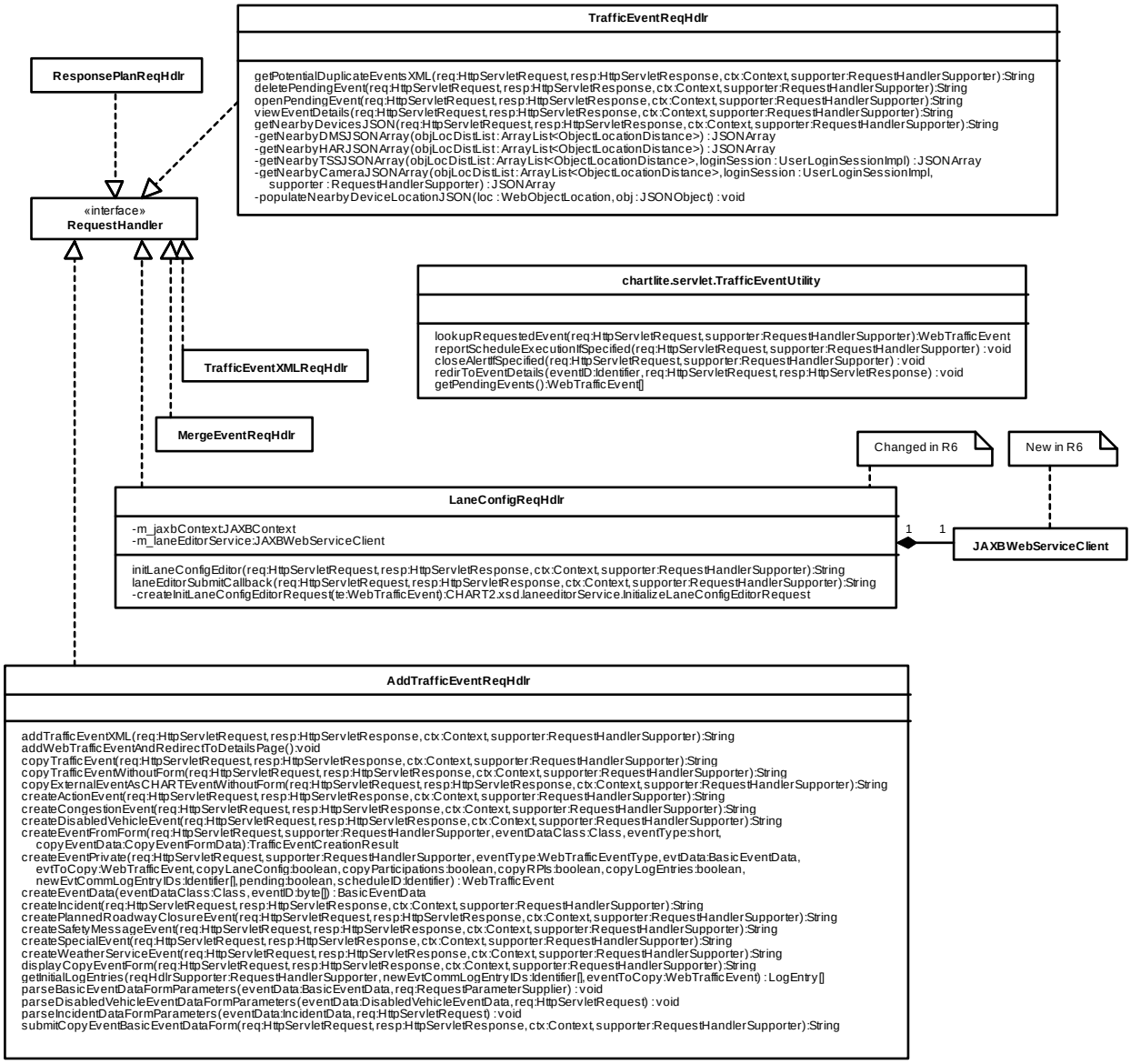


Figure 7-20 chartlite.servlet.trafficevents_classes (Class Diagram)

7.5.1.1.1 AddTrafficEventReqHdlr (Class)

This class is used to handle requests related to adding a traffic event to the system.

7.5.1.1.2 chartlite.servlet.TrafficEventUtility (Class)

This class contains methods that are useful for one or more traffic event related request handlers.

7.5.1.1.3 JAXBWebServiceClient (Class)

This class is a wrapper for an XMLHTTPService that makes it easy to send/receive data

to/from the service using objects that are generated via xsd and JAXB.

7.5.1.1.4 LaneConfigReqHdlr (Class)

This class handles any requests related to the traffic event lane configuration. The lane configuration editing has been moved to a web service as of R6, therefore the functionality of this class is mainly to initialize an editing session within the lane editor web service and to handle the results of the lane editing session when called back from the lane editor web service.

7.5.1.1.5 MergeEventReqHdlr (Class)

This class handles all requests related to merging traffic events.

7.5.1.1.6 RequestHandler (Class)

This interface specifies methods that are to be implemented by classes that are used to process requests.

7.5.1.1.7 ResponsePlanReqHdlr (Class)

This class handles requests related to traffic event response plans.

7.5.1.1.8 TrafficEventReqHdlr (Class)

This class handles requests related to traffic events that are not handled by one of the other specific traffic event request handlers.

7.5.1.1.9 TrafficEventXMLReqHdlr (Class)

This class handles requests related to traffic events that return XML for the Flex2 application.

7.5.2 Sequence Diagrams

7.5.2.1 LaneConfigReqHdlr:createInitLaneConfigEditorRequest2 (Sequence Diagram)

This diagram shows the processing that takes place to create a lane editing request to edit the roadway configuration and status for a traffic event. Note that in CHART, a traffic event must have a roadway location to have a direction, so any traffic event that does not have its roadway location set (main route and direction) cannot have its roadway configuration and status set. The general pattern followed in this method is to retrieve data from the traffic event, convert it to a JAXB generated object, and store it into the request object. The flag in the request that determines if the reference direction is to be displayed in the "up" direction is obtained from the system profile. The callback URL is formed using the URI of the request being processed with an "action" parameter that will direct the request to the proper handler when it arrives. The event ID is the only callback parameter included - it will be used to apply the lane configuration and status to the proper traffic

event when the callback is received. The title is set using the name of the traffic event. The request object is returned to the caller and is ready to be submitted to the lane editor web service.

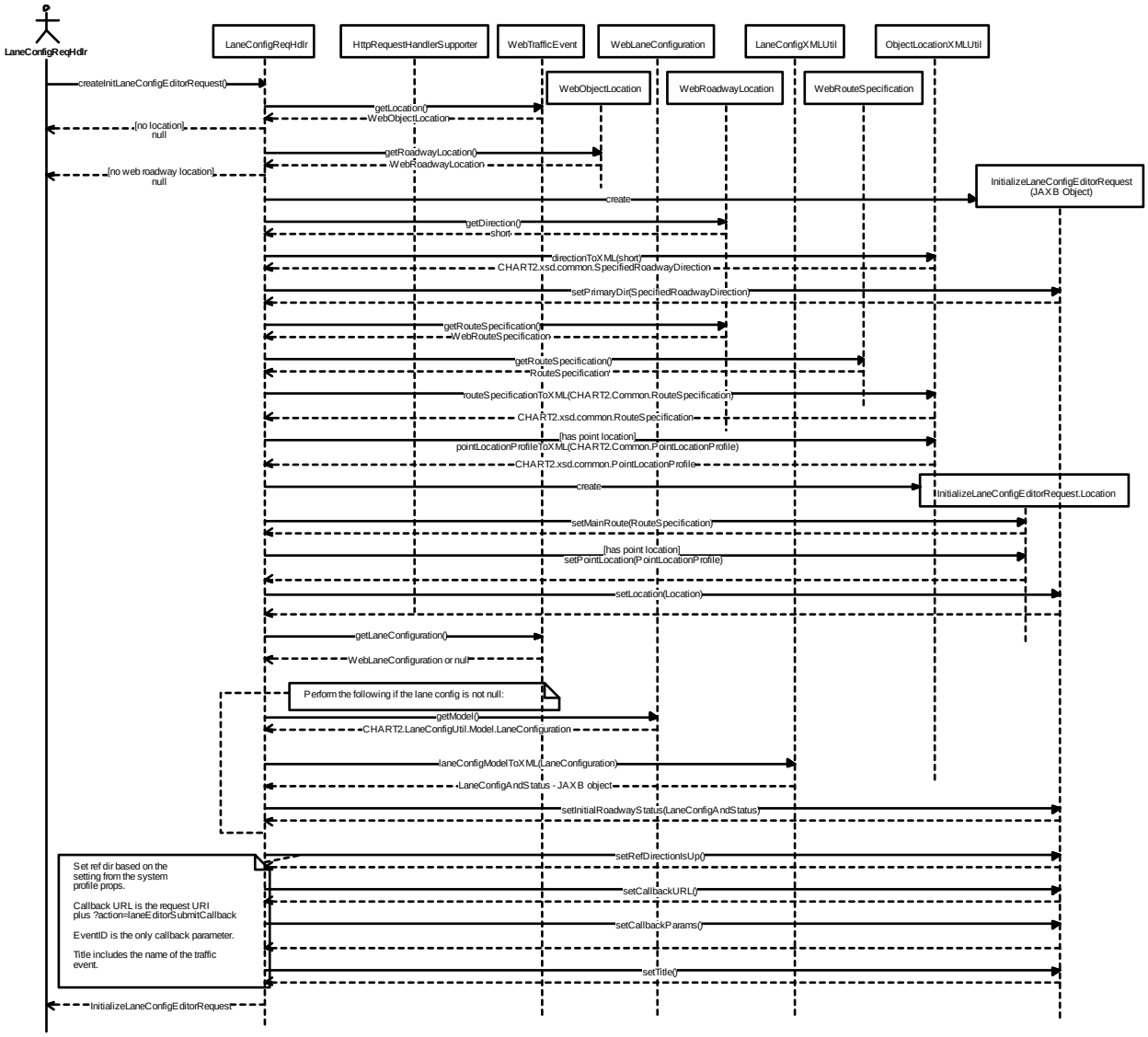


Figure 7-21 LaneConfigReqHdlr:createInitLaneConfigEditorRequest2 (Sequence Diagram)

7.5.2.2 LaneConfigReqHdlr:initLaneConfigEditor (Sequence Diagram)

This diagram shows the processing that takes place when the user clicks the Edit Roadway Configuration button on the traffic event details page in the CHART GUI. The web page uses AJAX to submit this request asynchronously, and the servlet returns the result as JSON. The user's rights are checked and only logged in users with the ManageTrafficEvents right are allowed to perform this request. The traffic event whose

roadway configuration is to be edited is retrieved from the GUI cache and passed to the createInitLaneConfigEditorRequest() method which returns a request object populated with data from the traffic event. The request is posted to the lane editor web service via the JAXBWebServiceClient and an object is returned that contains the response to the request. If an error occurs sending the request or receiving the response, an exception is thrown and an error message is returned to the browser. The same is true if the response indicates an error reported by the lane editor web service, however in that case the actual error messages received from the web service will be returned to the browser. If the request to the web service was successful, the lane editor instance ID is retrieved from the response and returned to the browser in a JSON object.

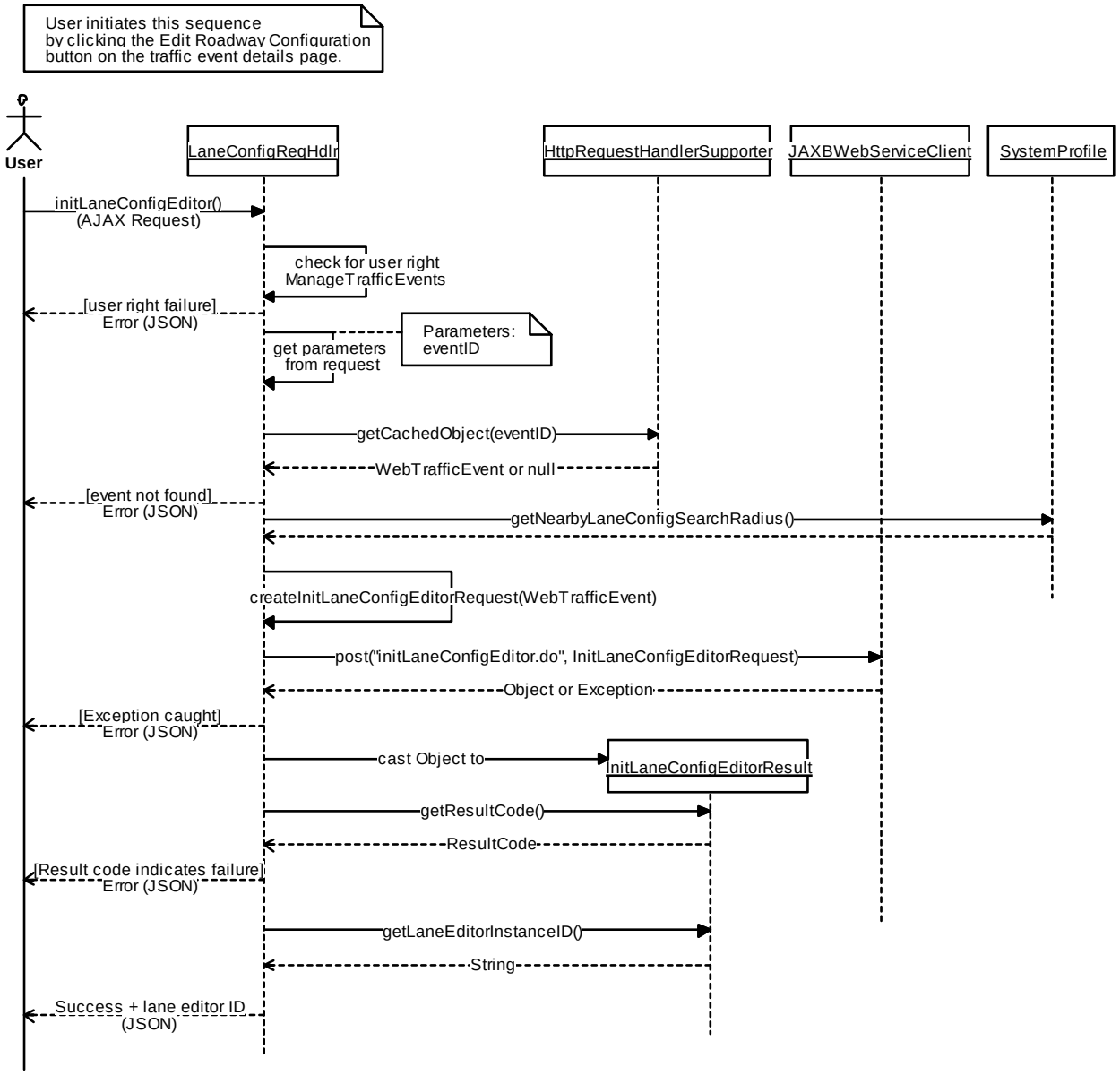


Figure 7-22 LaneConfigReqHdlr:initLaneConfigEditor (Sequence Diagram)

7.5.2.3 LaneConfigReqHdlr:laneEditorSubmitCallback (Sequence Diagram)

This diagram shows the processing that occurs when the lane editor web service calls back into the CHART GUI servlet when the user has submitted the lane editor. The CHART GUI uses the clientID and signature parameters to verify that the request is submitted by an authorized client and the XML data is unaltered. It then uses the access token for the client to make sure the client is authorized to manage traffic events in the CHART system. Any errors that are detected result in XML being returned with a detailed error message. After the client is authenticated, the XML is marshaled into java objects using the JAXB tool, and the traffic event ID is retrieved from the callback parameters included in the request. The traffic event ID is used to lookup the traffic event in the GUI cache. If not found an XML error response is returned. The JAXB object that contains the lane configuration and status is passed to the LaneConfigXMLUtil to convert it into a generic LaneConfiguration, and the generic LaneConfiguration is passed to the LaneConfigIDLUtil to convert the lane configuration into IDL defined objects. The IDL version of the lane config is then passed to the CHART server to set the traffic event's lane configuration. The lane configuration is then passed to the WebTrafficEvent object so it can update the configuration in the cache. An XML Success message is returned to the lane editor web service.

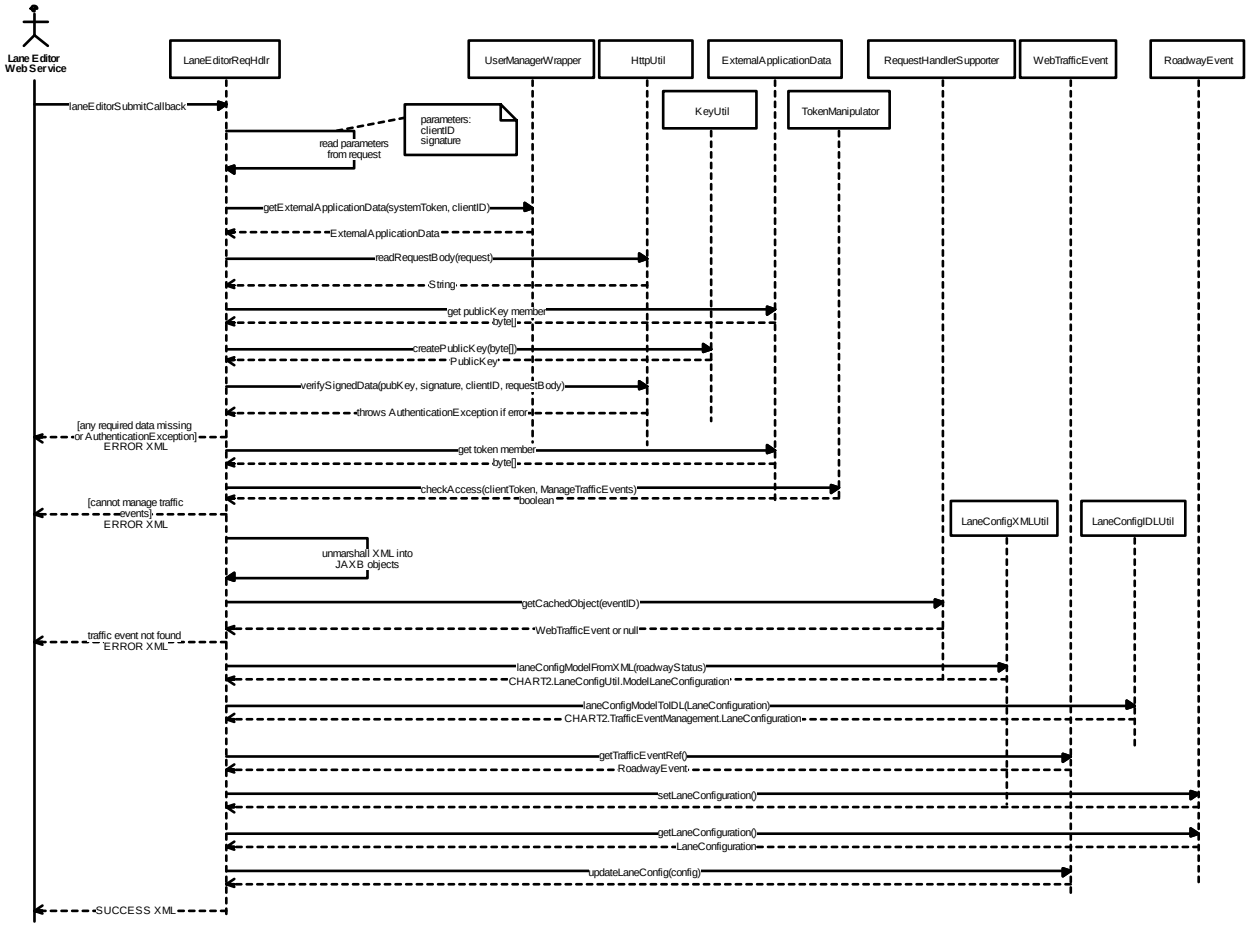


Figure 7-23 LaneConfigReqHdlr:laneEditorSubmitCallback (Sequence Diagram)

7.6 webservices.laneeditormodule

7.6.1 Class Diagrams

7.6.1.1 LaneEditorClassDiagram (Class Diagram)

This diagram shows classes for the Lane Editor web service, a web service that provides the ability to view and edit lane configurations and status.

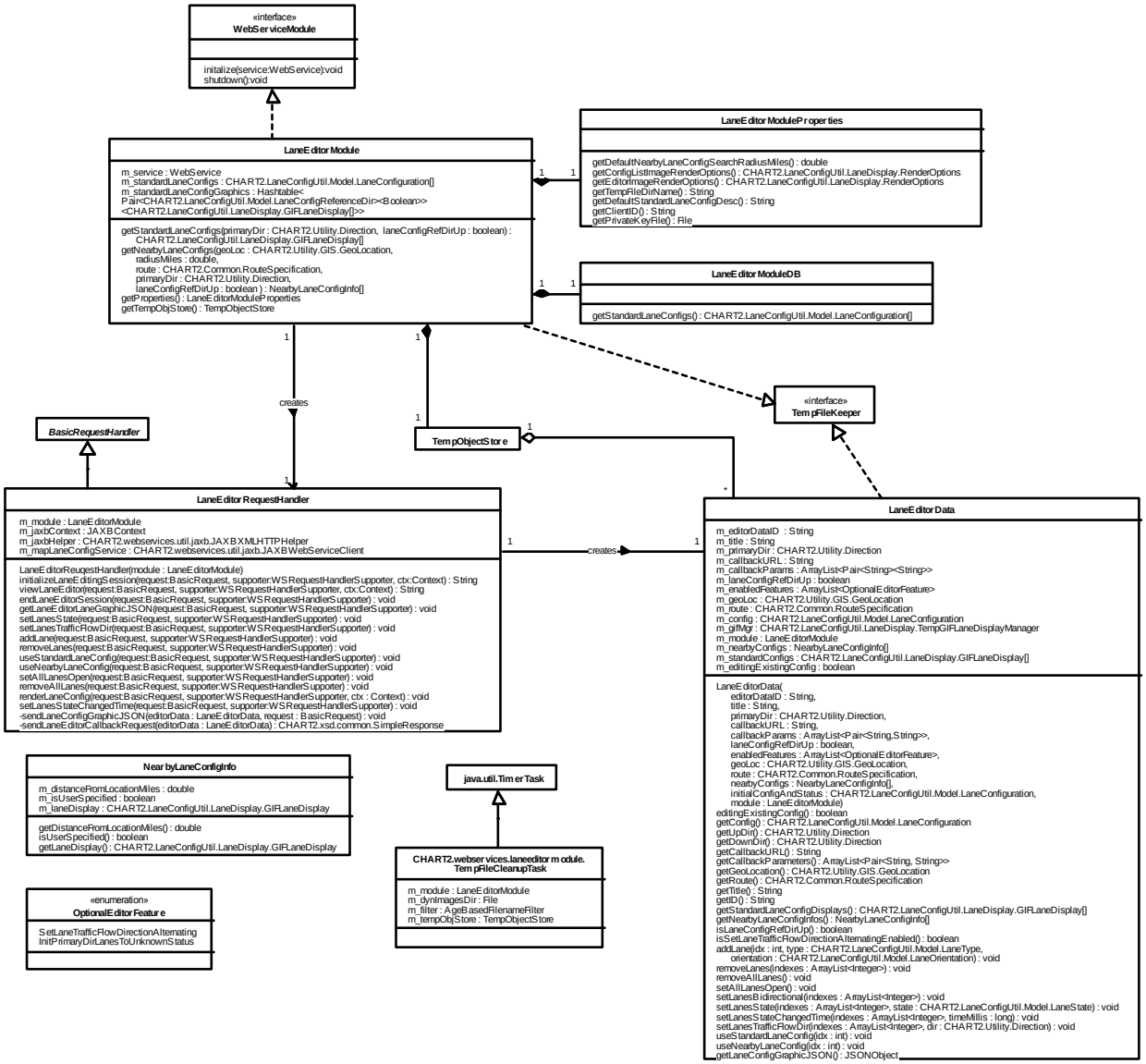


Figure 7-24 LaneEditorClassDiagram (Class Diagram)

7.6.1.1.1 BasicRequestHandler (Class)

This abstract base class provides an implementation of the WSRequestHandler.processRequest() method that provides optional XML validation against specified XSD files and optional digital signature verification as well. It is intended to be used by request handlers that plan to take XML in and return XML to the calling client.

7.6.1.1.2 CHART2.webservices.laneeditormodule. TempFileCleanupTask (Class)

This class is used to periodically clean up temporary lane configuration image files.

7.6.1.1.3 java.util.TimerTask (Class)

This class is an abstract base class which can be scheduled with a timer to be executed one or more times.

7.6.1.1.4 LaneEditorData (Class)

This class represents all data needed to model a lane configuration / status editing session. It contains methods for accessing and modifying the lane configuration and status data in memory.

7.6.1.1.5 LaneEditorModule (Class)

This class represents the lane editor module, which supports the editing of lane configuration and status.

7.6.1.1.6 LaneEditorModuleDB (Class)

This class contains functionality for accessing data needed by the lane editor module from the database.

7.6.1.1.7 LaneEditorModuleProperties (Class)

This class allows access to configurable parameters defined in the service's properties file.

7.6.1.1.8 LaneEditorRequestHandler (Class)

This class handles requests related to editing lane configuration and status.

7.6.1.1.9 NearbyLaneConfigInfo (Class)

This class contains information about lane configurations that are near to a requested location.

7.6.1.1.10 OptionalEditorFeature (Class)

This enumeration lists optional editor features.

7.6.1.1.11 TempFileKeeper (Class)

This interface allows a class to "own" temporary files for the purpose of preventing them from being deleted by periodic cleanup functionality.

7.6.1.1.12 TempObjectStore (Class)

This class provides a self cleaning storage area for temporary objects.

7.6.1.1.13 WebServiceModule (Class)

This interface defines the methods that each module must implement in order to run within the web service framework.

7.6.2 Sequence Diagrams

7.6.2.1 LaneEditorClient:LaneEditorInitializationFromClient (Sequence Diagram)

This diagram shows how the CHART GUI initiates the lane editor using the lane editor web service. While in the CHART GUI, the user clicks a button indicating they wish to edit the lane configuration for a specific traffic event. That request is sent to the CHART GUI servlet via AJAX, meaning the message is sent asynchronously to the serlvet. The CHART GUI servlet checks the user's rights to make sure they have permission to edit the traffic event. The XML for the request is prepared using JAXB objects. An object hierarchy is built up and values such as the callback URL and callback parameters are stored in those objects. A digital signature using the CHART GUI private key is created for the XML data. An http request is made to the lane editor web service passing the CHART GUI client id and digital signature as http parameters and the XML data as the request body. The lane editor web service returns the ID for the new lane editing session, and the CHART GUI returns this to the browser as a response to the AJAX request. When the CHART GUI web page receives this AJAX response, it creates a popup window and uses the lane editor ID in an http request to the lane editor web service, and it responds with html for the lane config editor. The lane config editor html (and JavaScript) is coded to interact with the lane editor web service, so once the lane editor is displayed, the client application has no further interaction until the user submits the form. See LaneEditorSubmitToClient for details.

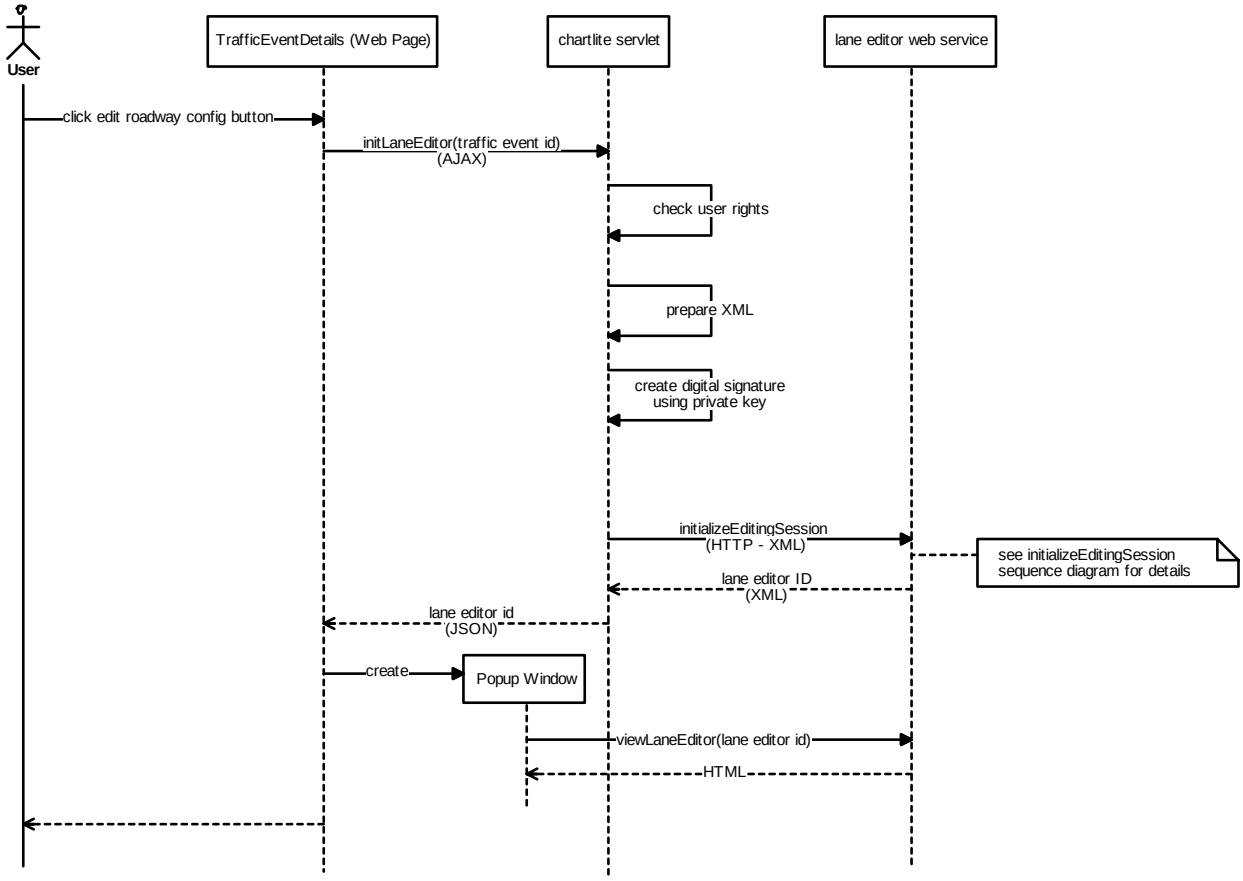


Figure 7-25 LaneEditorClient:LaneEditorInitializationFromClient (Sequence Diagram)

7.6.2.2 LaneEditorClient:LaneEditorSubmitToClient (Sequence Diagram)

This diagram shows the processing that occurs when the user has submitted the lane editor form. The lane editor form (using JavaScript) sends an AJAX request to the lane editor web service to indicate the user is finished editing the configuration. The lane editor web service calls the client application using the callback URL provided when the client initialized the editing session. The lane editor web service passes its clientID and a digital signature of the XML it includes in the request body. The XML includes any callback name/value pairs that were provided during initialization as parameters of the request, the final lane configuration/status that appeared in the editor, and other data as defined in XSD. The client application can then process the submit and returns a callback result as XML (defined in XSD). When the lane editor web service receives this response, it provides an AJAX response to the lane editor form which can display an error message if needed or close the popup window. Before closing, it will make a JavaScript call to its parent if the parent has a function defined named "laneEditorClosed". This allows the parent window to update information related to the lane configuration that was submitted.

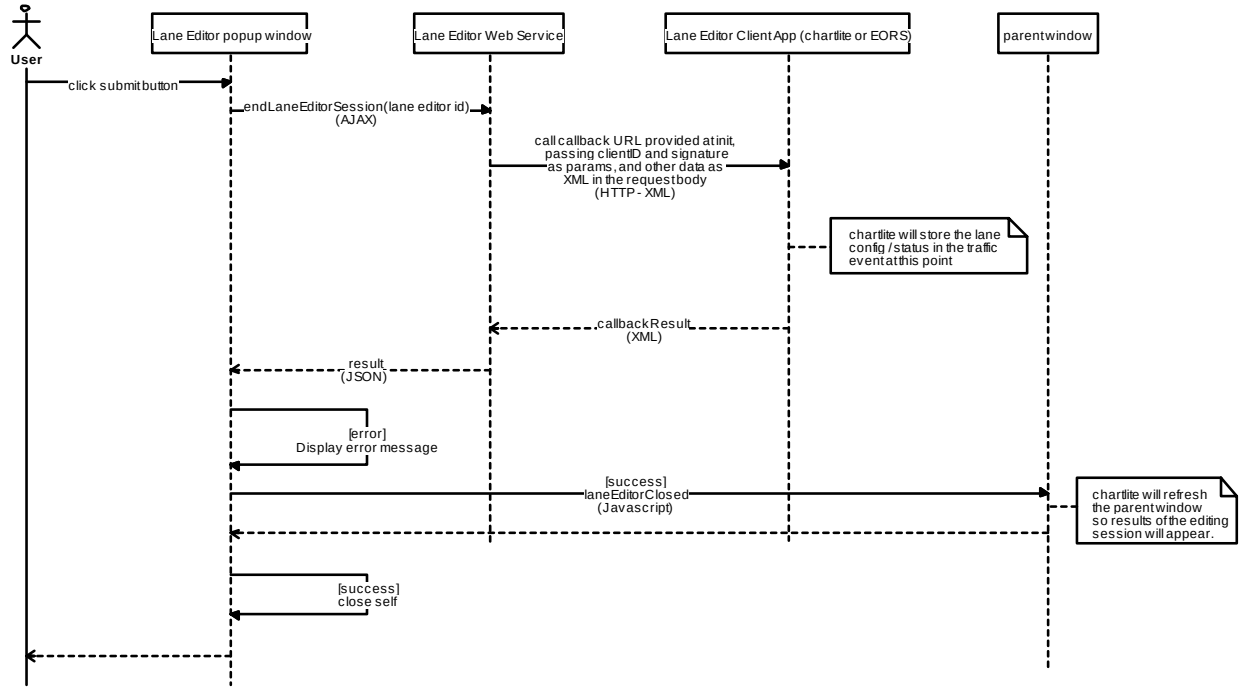


Figure 7-26 LaneEditorClient:LaneEditorSubmitToClient (Sequence Diagram)

7.6.2.3 LaneEditorModule:getNearbyLaneConfigs (Sequence Diagram)

This diagram shows the processing that is performed when the getNearbyLaneConfigs method is called to retrieve lane configurations that are close to a specified location and optionally on a specified route. A new GetNearbyLaneConfigsRequest is created and populated with information from the method parameters, and sent to the GIS Lane Config Service via a JAXBWebServiceClient. The Object that is returned is cast to a GetNearbyLaneConfigsResult and the lane configurations in the results are retrieved. Each lane config is processed by creating a lane display object and then storing the lane display and lane config in a NearbyLaneConfigInfo object. The list of NearbyLaneConfigInfo Objects is returned to the caller.

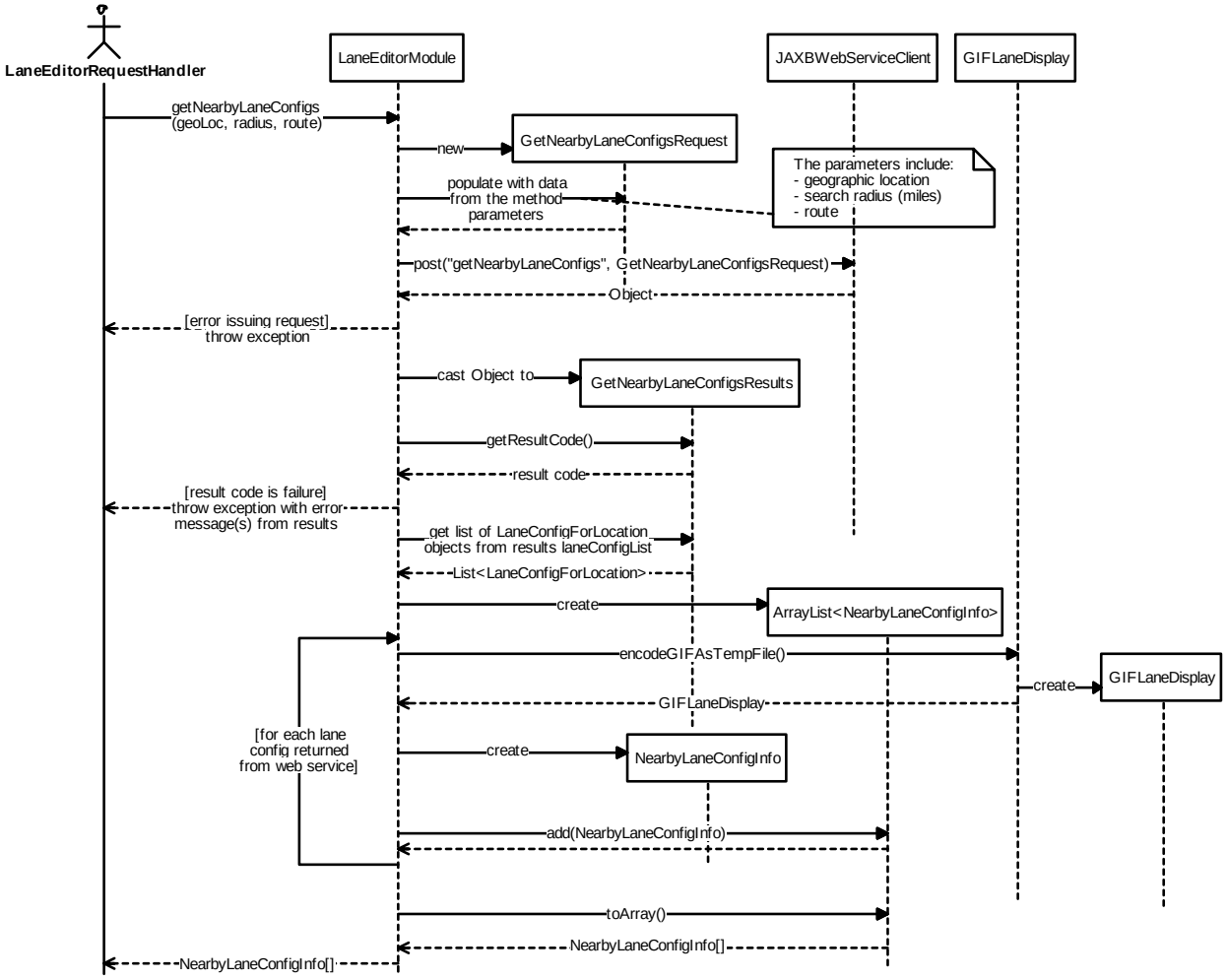


Figure 7-27 LaneEditorModule:getNearbyLaneConfigs (Sequence Diagram)

7.6.2.4 LaneEditorRequestHandler:addLane (Sequence Diagram)

This diagram shows the processing when a lane is added. The editor data ID specified in the request is used to look up the LaneEditorData object from the module's TempObjectStore. The lane index, type, and orientation request parameters are used to create a Lane object. Then, if the feature to initialize the lane configuration to the default state (i.e., lanes in the primary direction have Unknown state and the lanes in the opposite direction are Open) is not disabled and the configuration is still in the default state, the new lane is set to the default state also. The lane is added to the list of lanes in the lane configuration model, and a new lane configuration description is generated to account for the new lane. The lane descriptions are also updated to use new index values, such as L1, L2, etc.. A new GIF file is created to represent the configuration with the newly added lane. Finally the GIF image and associated metadata is sent back in the response in JSON format.

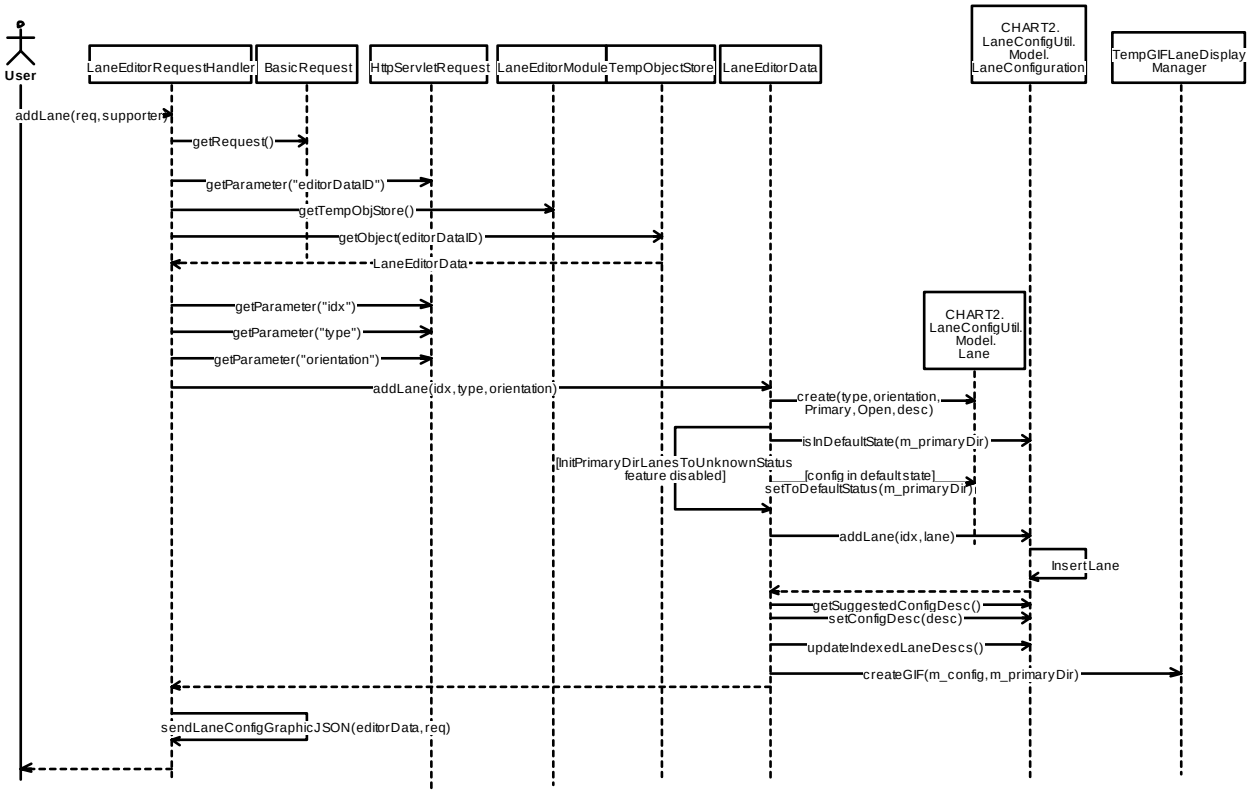


Figure 7-28 LaneEditorRequestHandler:addLane (Sequence Diagram)

7.6.2.5 LaneEditorRequestHandler:endLaneEditorSession (Sequence Diagram)

This diagram shows the processing used when the user submits the Lane Editor form. The LaneEditorData object is retrieved from the TempObjectStore, and sendLaneEditorCallbackRequest() is called to send the lane configuration and status back to the client application. (See that diagram for details). A JSON response object is then sent back to the browser so that the popup window can be closed (or error message can be displayed). The lane configuration is then sent to the Map Lane Configuration Service via a call to sendStoreLaneConfigRequest() (see that sequence diagram for details). The LaneEditorData object is removed from the TempObjectStore, as it is no longer needed.

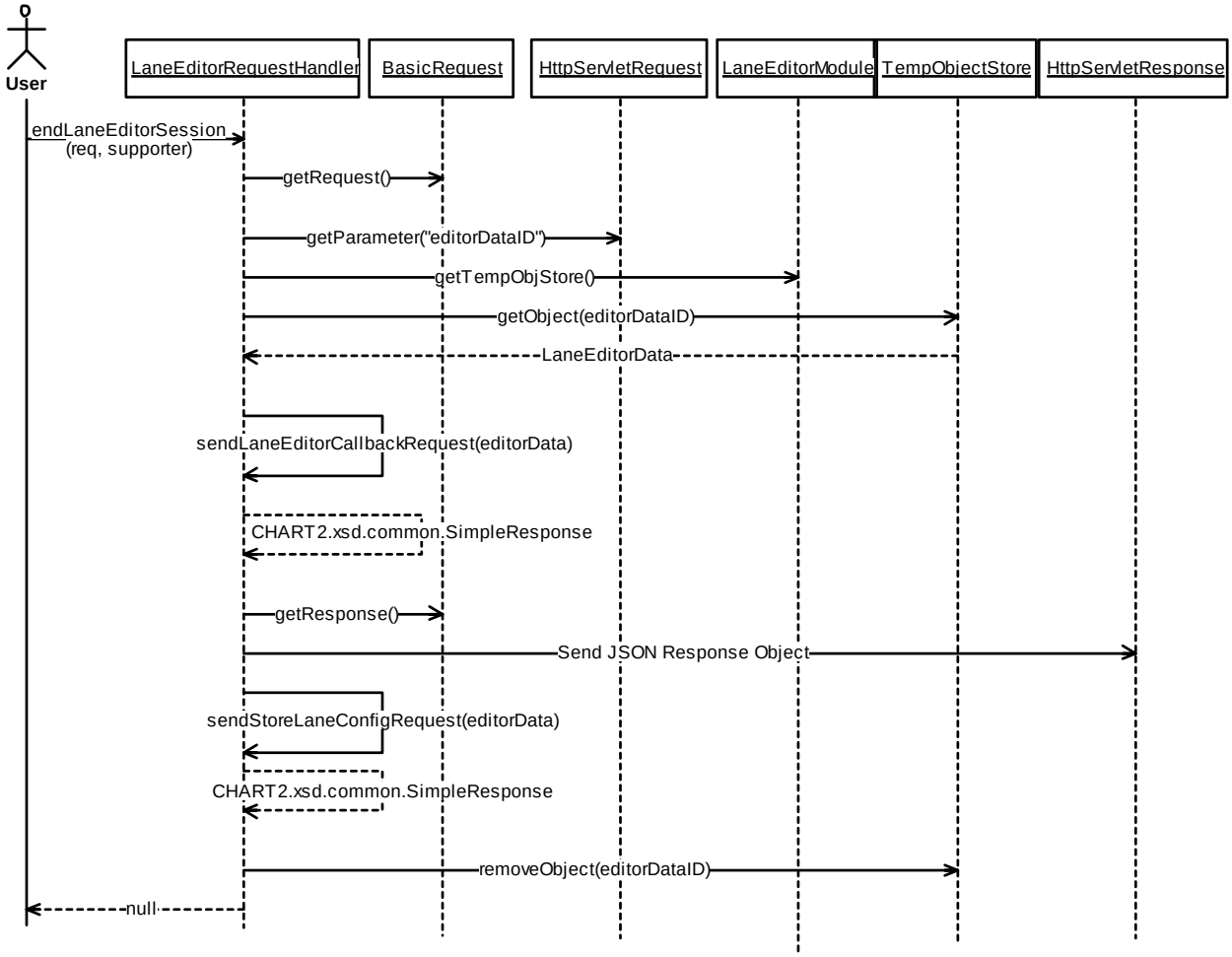


Figure 7-29 LaneEditorRequestHandler:endLaneEditorSession (Sequence Diagram)

7.6.2.6 LaneEditorRequestHandler:initializeEditingSession (Sequence Diagram)

This diagram shows the processing that takes place when the Lane Editor web service receives a request from a client to initialize an editing session. The authentication mechanism of the base BasicRequestHandler base class is utilized to ensure that only authorized clients (namely CHART GUI and EORS) are allowed to perform this request. Likewise, XML validation is done by the base class. If authentication and validation pass, the processRequest method of the LaneEditorRequestHandler will be called. The XML is retrieved from the http post request body and the JAXB tool is used to unmarshall the XML into Java objects that mirror the definitions in the XSD. Calls are then made to retrieve the request parameters from the JAXB generated objects, and they are stored in a new LaneEditingSession object. The LaneEditingSession object is stored in the temporary object store and is used to store the initial lane configuration as well as changes that occur while the user edits the lane configuration and status. The ID of this temp object is passed back to the caller so it can be used to initially display the editor form within the context of the calling application.

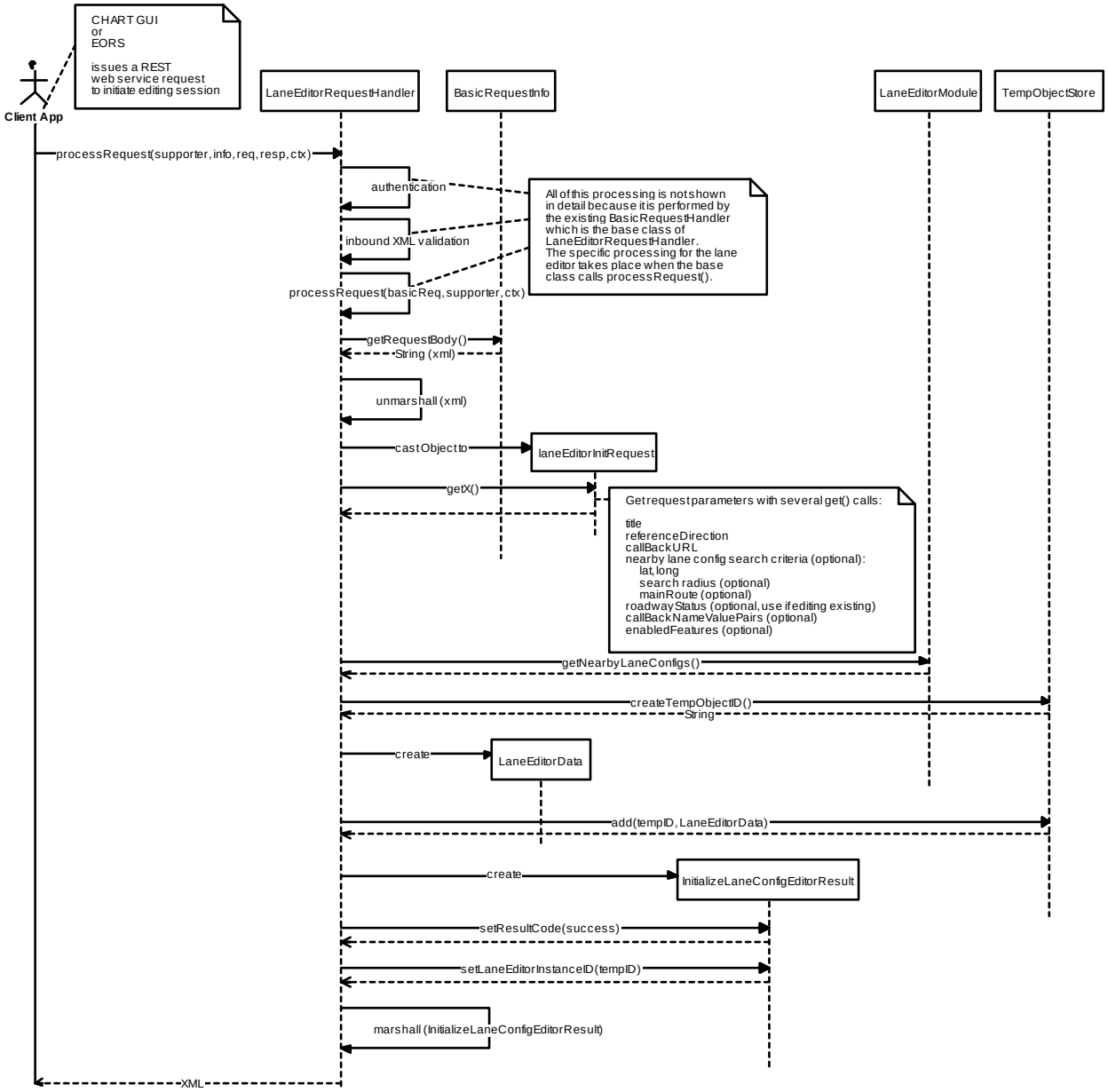


Figure 7-30 LaneEditorRequestHandler:initializeEditingSession (Sequence Diagram)

7.6.2.7 LaneEditorRequestHandler:marshall (Sequence Diagram)

This diagram shows the processing that is done to convert a JAXB generated object into XML.

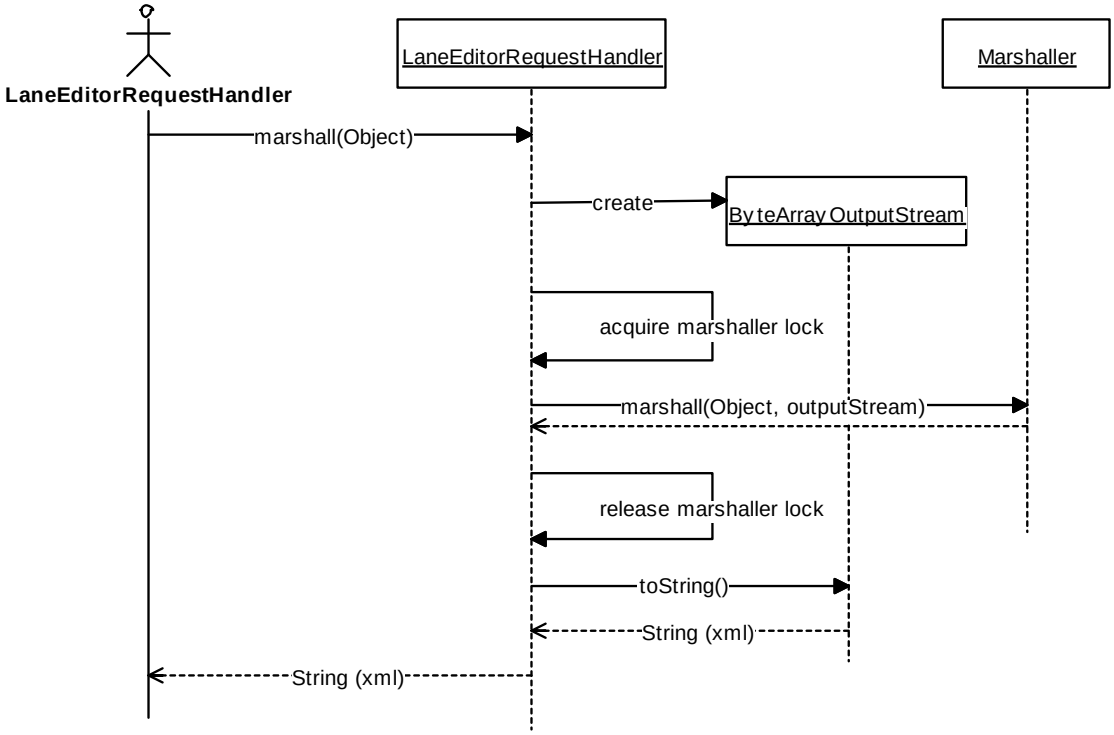


Figure 7-31 LaneEditorRequestHandler:marshall (Sequence Diagram)

7.6.2.8 LaneEditorRequestHandler:removeAllLanes (Sequence Diagram)

This diagram shows the processing when all lanes are removed. The editor data ID specified in the request is used to look up the LaneEditorData object from the module's TempObjectStore. All lanes are removed from the lane configuration model, and a new lane configuration description is generated. The current image is cleared via the GIF manager. Finally the JSON (without GIF image filename) describing the lane configuration is sent back in the response.

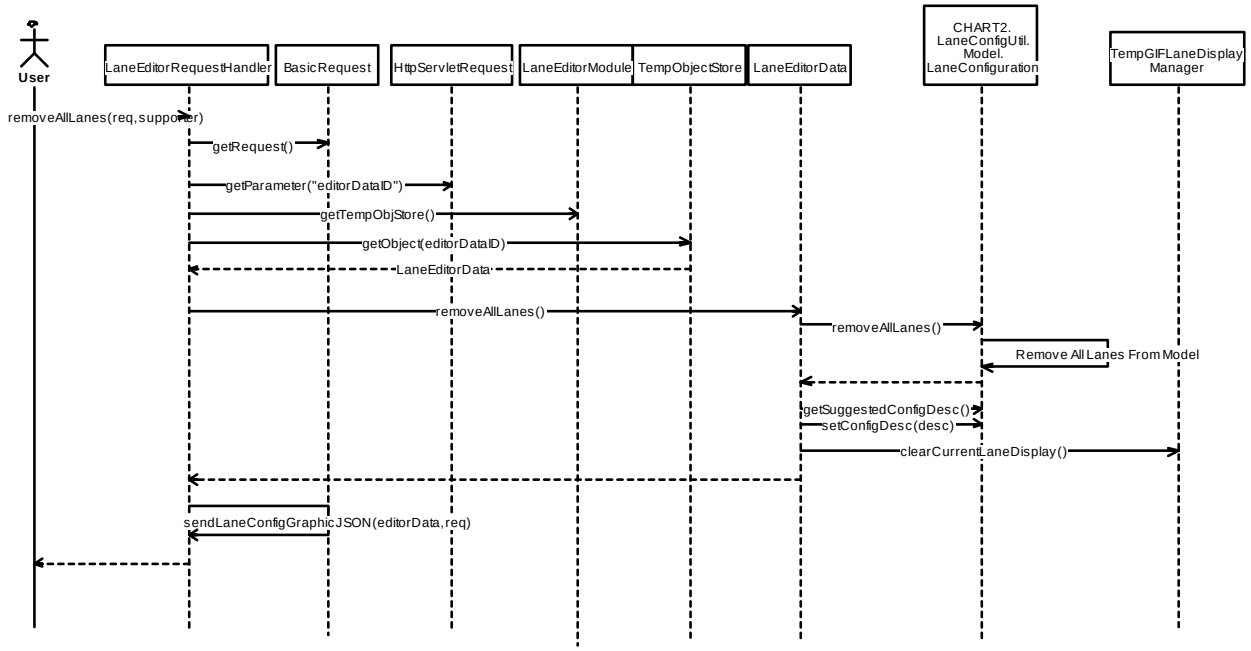


Figure 7-32 LaneEditorRequestHandler:removeAllLanes (Sequence Diagram)

7.6.2.9 LaneEditorRequestHandler:removeLanes (Sequence Diagram)

This diagram shows the processing when lanes are removed. The editor data ID specified in the request is used to look up the LaneEditorData object from the module's TempObjectStore. The lane index values to remove are retrieved from a request parameter, and these index values are passed to the lane configuration model. The lanes are removed from the model, and a new lane configuration description is generated to account for the removal of the lane(s). The lane descriptions are also updated to use new index values, such as L1, L2, etc.. If the configuration is now empty, the current image is cleared; otherwise, a new GIF image is created to represent the configuration with the lane(s) removed. Finally the GIF image and associated metadata is sent back in the response in JSON format.

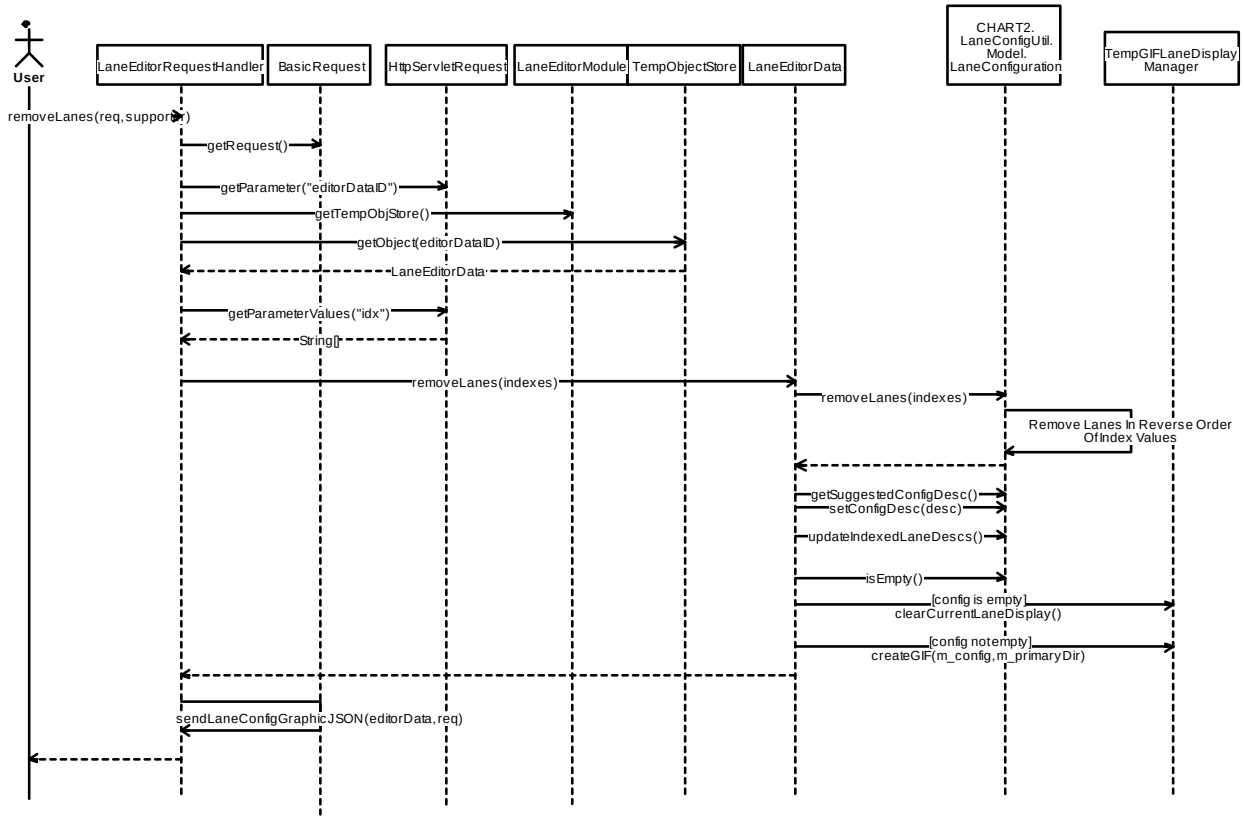


Figure 7-33 LaneEditorRequestHandler:removeLanes (Sequence Diagram)

7.6.2.9.1 LaneEditorRequestHandler:renderLaneConfig (Sequence Diagram)

This diagram shows the processing to render a lane configuration and status object into GIF-encoded data. First the request is unmarshalled and the lane config and status, and requested image settings JAXB objects are extracted. These are converted into LaneConfiguration model and RenderOptions objects. The GIFLaneDisplay class is called to encode the GIF as a byte array, which creates a GIFLaneDisplay object. This contains the byte array, and all associated metadata for the lane configuration and for each lane. A RenderLaneConfigResult JAXB object is prepared using the available data including the GIF bytes, which are also encoded as a Base64 string. The RenderLaneConfigResult object is marshalled to an XML string, which is put into the Velocity context, and the Velocity template is returned.

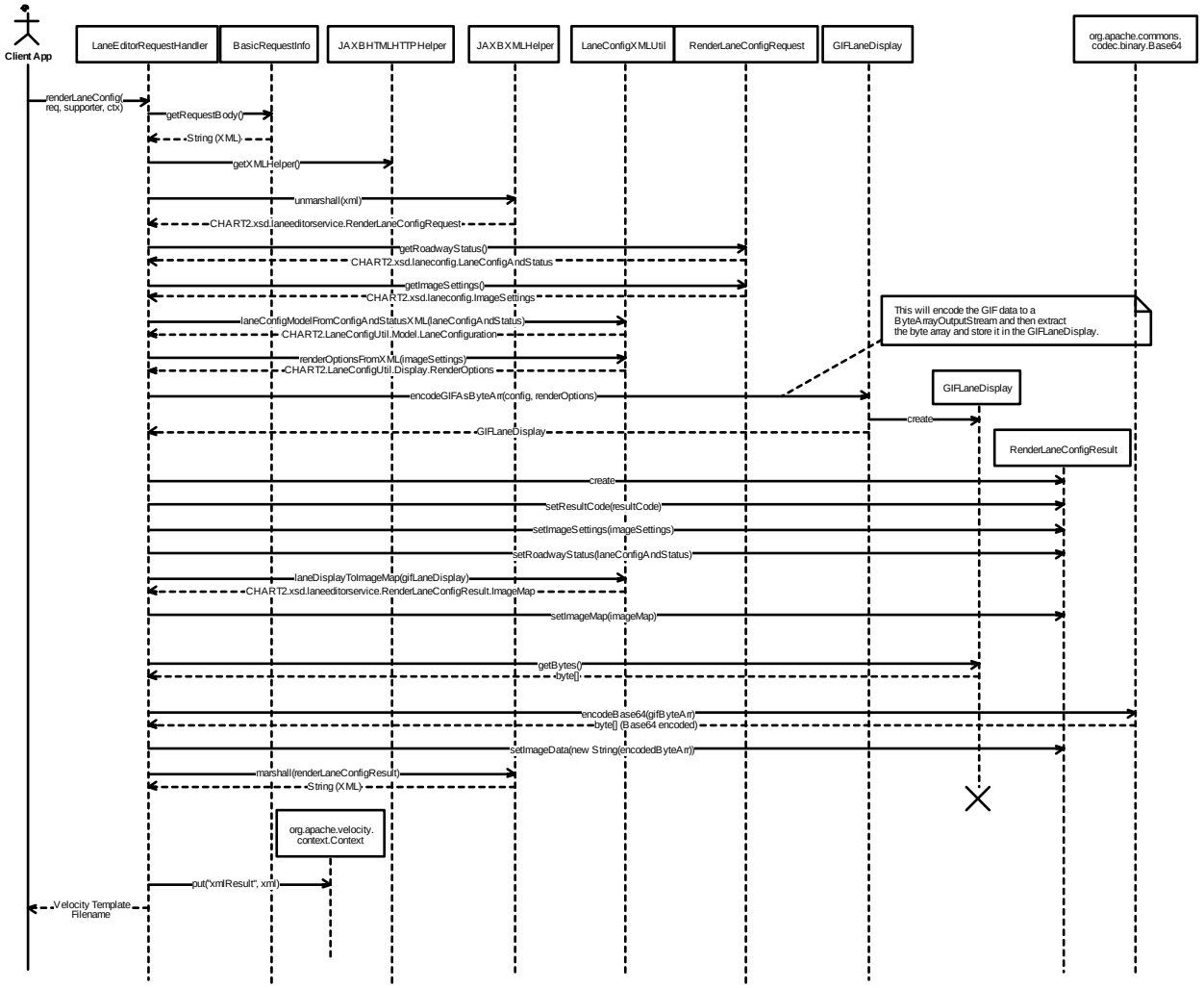


Figure 7-34 LaneEditorRequestHandler:renderLaneConfig (Sequence Diagram)

7.6.2.10 LaneEditorRequestHandler:sendLaneConfigGraphicJSON (Sequence Diagram)

This diagram shows how the lane configuration graphic JSON is built and sent to the browser. This JSON contains the GIF image filename, as well as the status for the lane configuration and per-lane information also. If the lane configuration has lanes, the TempGIFLaneDisplayManager is called to get the current GIFLaneDisplay object, which is called to build the JSON containing the GIF filename and all configuration and status for the configuration and per-lane info as well. The configuration description and lane closure description are also added, as well as some rendering information. The JSON object is then send back in the response, so that the browser can display the graphic and associated data.

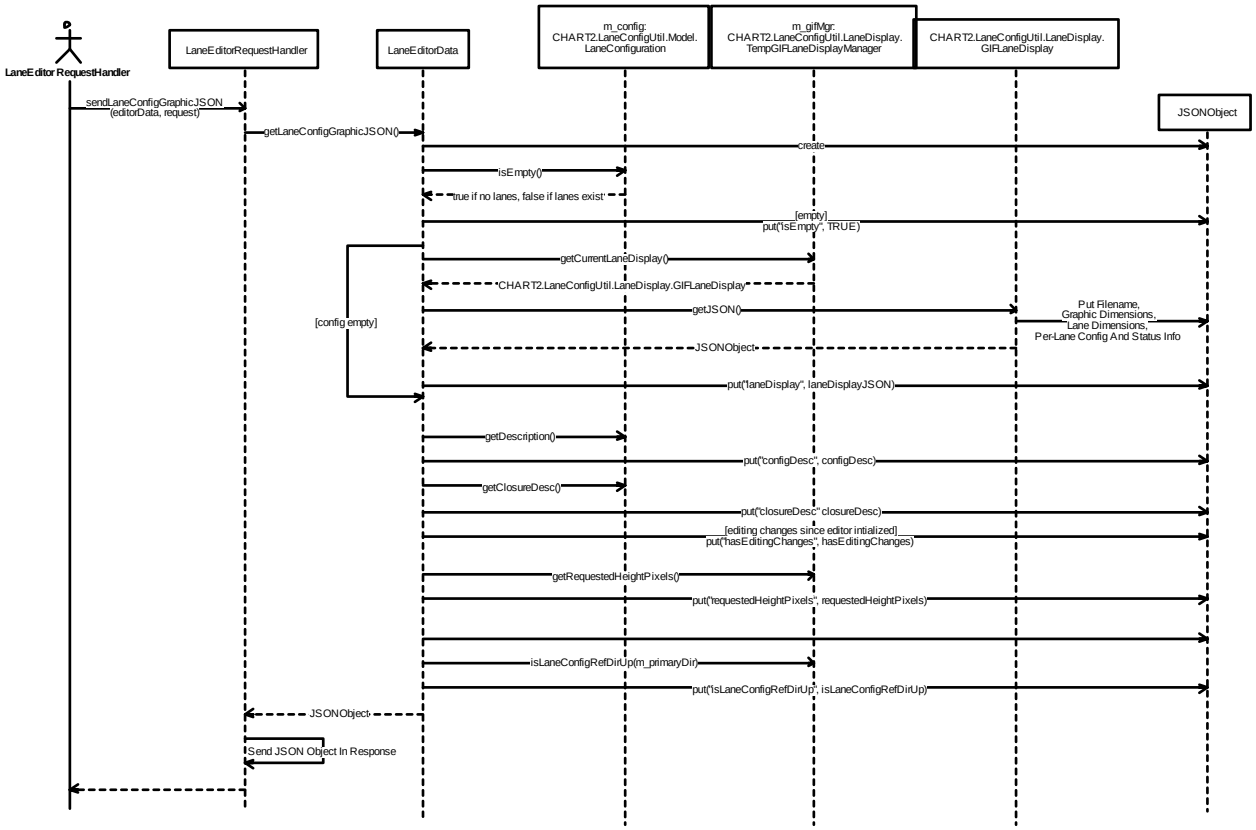


Figure 7-35 LaneEditorRequestHandler:sendLaneConfigGraphicJSON (Sequence Diagram)

7.6.2.11 LaneEditorRequestHandler:sendLaneEditorCallbackRequest (Sequence Diagram)

This diagram shows the processing used to send the lane editor callback request back to the client application that invoked the Lane Configuration Editor. An XSD / JAXB object is created representing the callback request data. The lane configuration and status, editor instance ID, and callback parameters are set into this object. A JAXBWebServiceClient helper object is created, which knows how to marshal / unmarshal the request and response data to (and from) XML, and which also inserts an authentication signature for the Lane Editor Service. The post() method is used to post the data, and the client application is expected to return a SimpleResponse XML document. This is unmarshalled and returned to the caller as a JAXB object.

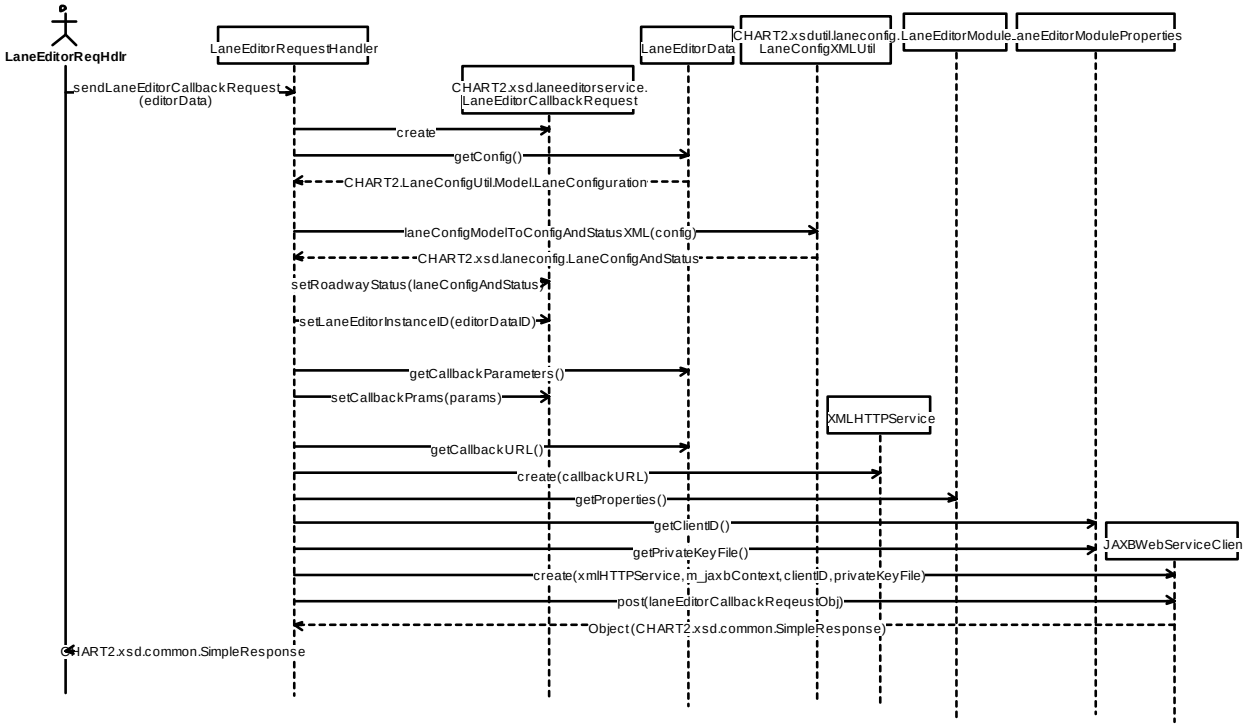


Figure 7-36 LaneEditorRequestHandler:sendLaneEditorCallbackRequest (Sequence Diagram)

7.6.2.12 LaneEditorRequestHandler:sendStoreLaneConfigRequest (Sequence Diagram)

This diagram shows the processing used to the request to store the lane configuration to the Map Lane Configuration Service. An XSD / JAXB object is created representing the request data. The lane configuration, point, and route specification are set into this object. The JAXBWebServiceClient helper object for the Map Lane Configuration Service is called to post the data. This object knows how to marshal / unmarshal the request and response data to (and from) XML, and which also inserts an authentication signature for the Lane Editor Service. The service is expected to return a SimpleResponse XML document, which is unmarshalled and returned to the caller as a JAXB object.

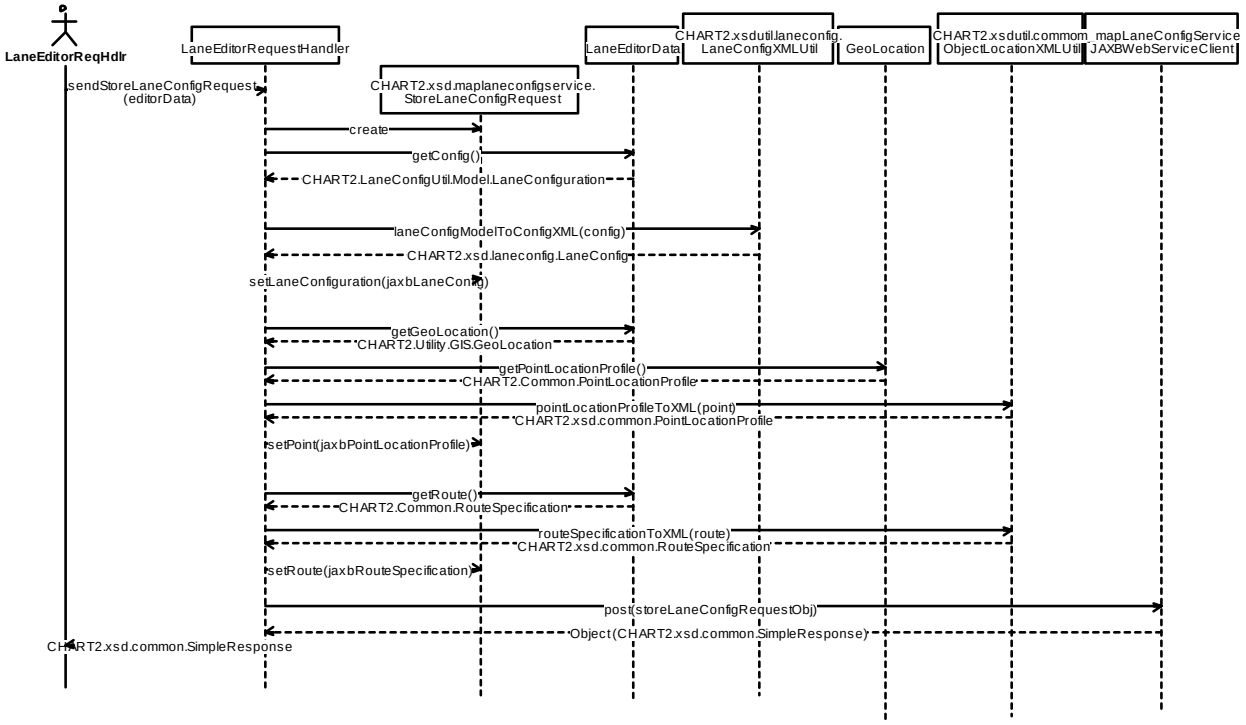


Figure 7-37 LaneEditorRequestHandler:sendStoreLaneConfigRequest (Sequence Diagram)

7.6.2.13 LaneEditorRequestHandler:setAllLanesOpen (Sequence Diagram)

This diagram shows the processing when all lanes are set to the Open state. The editor data ID specified in the request is used to look up the LaneEditorData object from the module's TempObjectStore, and the LaneEditorData is called to set all lanes to Open. It calls the lane configuration model to get the lanes, and sets the state and time for each lane that is not already open. A new GIF image is created to represent the configuration with the lane state changed. Finally the GIF image and associated metadata are sent back in the JSON response.

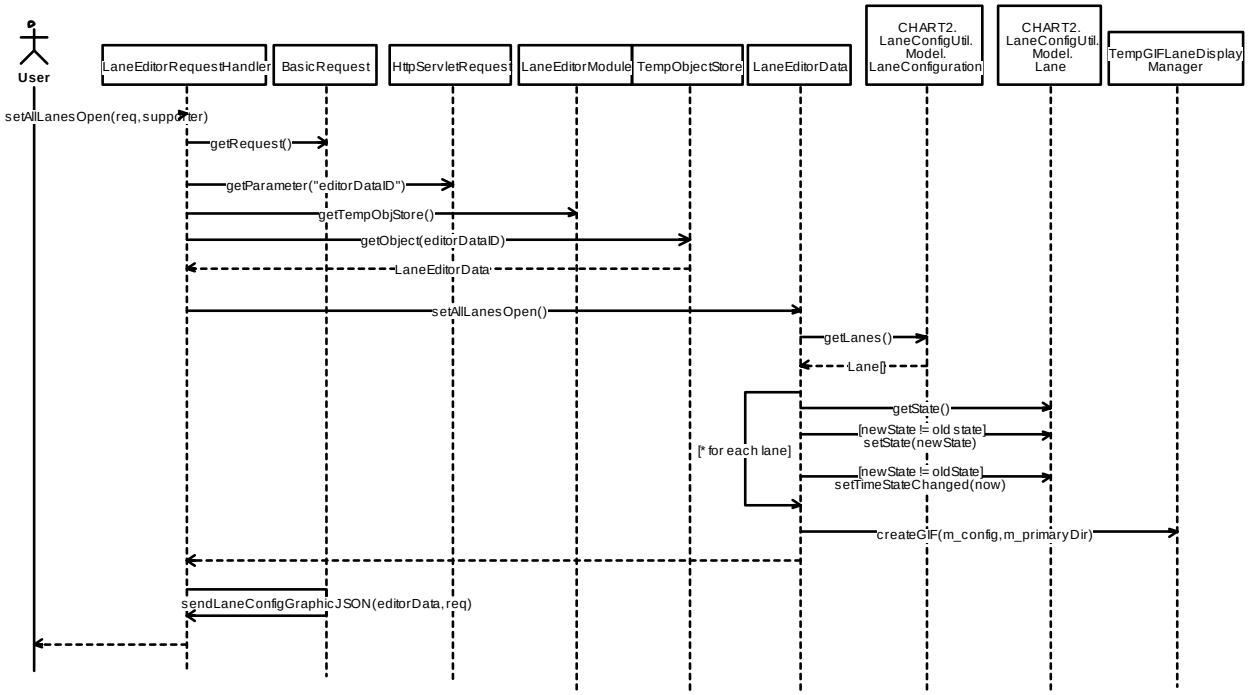


Figure 7-38 LaneEditorRequestHandler:setAllLanesOpen (Sequence Diagram)

7.6.2.14 LaneEditorRequestHandler:setLanesState (Sequence Diagram)

This diagram shows the processing when the lane state is set for one or more lanes. The editor data ID specified in the request is used to look up the LaneEditorData object from the module's TempObjectStore. The new lane state and the lane index values are retrieved from request parameters, and the LaneEditorData is called to set the lanes' state. It calls the lane configuration model first to determine whether the lanes are in the default state (i.e., Unknown in the primary direction and Open in the opposite direction) prior to the change. Then it sets the state and time for each lane that is not already in the new state. If the lane configuration was in the default state before, the lane configuration model is called once again to see if it is still in the default state. If it is no longer in the default state, all Unknown lanes in the primary direction are set to Open. A new GIF image is created to represent the configuration with the lane state changed. Finally the GIF image and associated metadata are sent back in the JSON response.

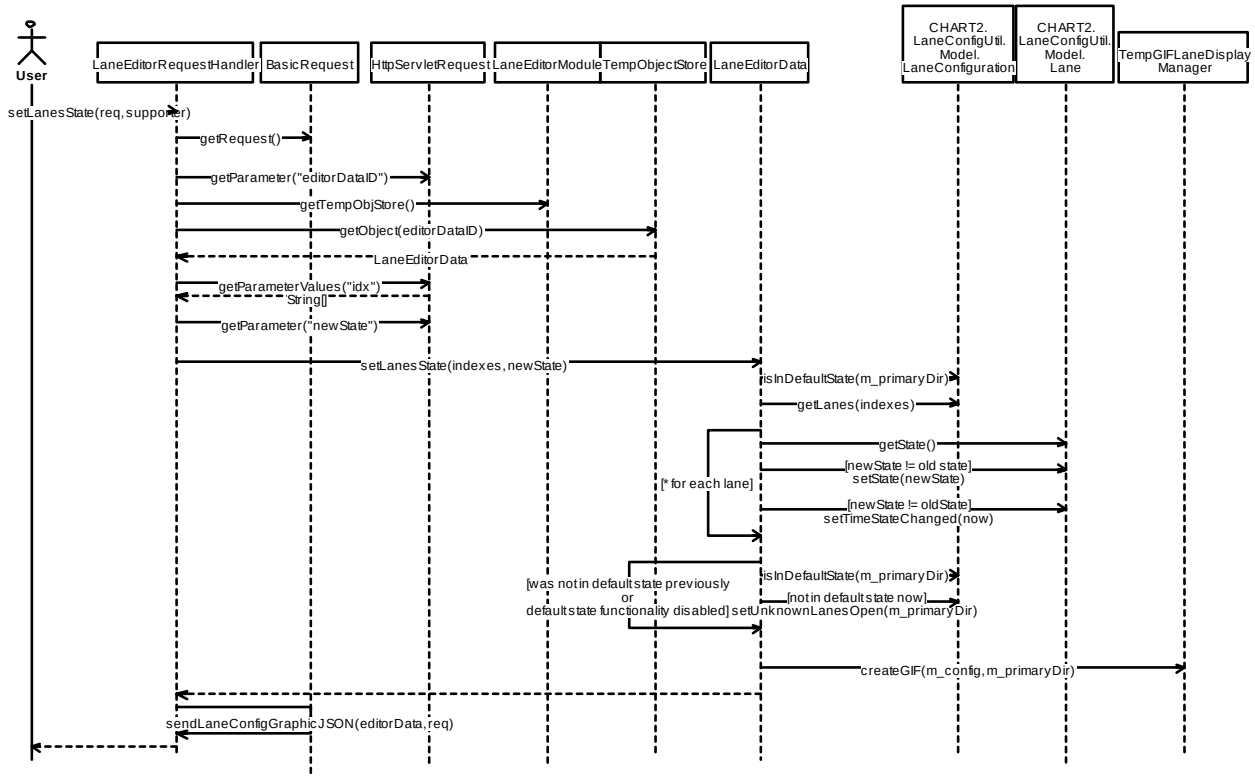


Figure 7-39 LaneEditorRequestHandler:setLanesState (Sequence Diagram)

7.6.2.15 LaneEditorRequestHandler:setLanesStateChangedTime (Sequence Diagram)

This diagram shows the processing when the lane state changed time is set for one or more lanes. The editor data ID specified in the request is used to look up the LaneEditorData object from the module's TempObjectStore. The new time and the lane index values are retrieved from request parameters, and the LaneEditorData is called to set the lanes' direction. It calls the lane configuration model to get the lanes, and for each lane it sets the state changed time for each lane. The GIF manager is called to create a new GIF image (not because the image is changing, but to generate an updated LaneDisplay object corresponding to the GIF, which has the new state changed times). The GIF image location and associated metadata are sent back in the JSON response.

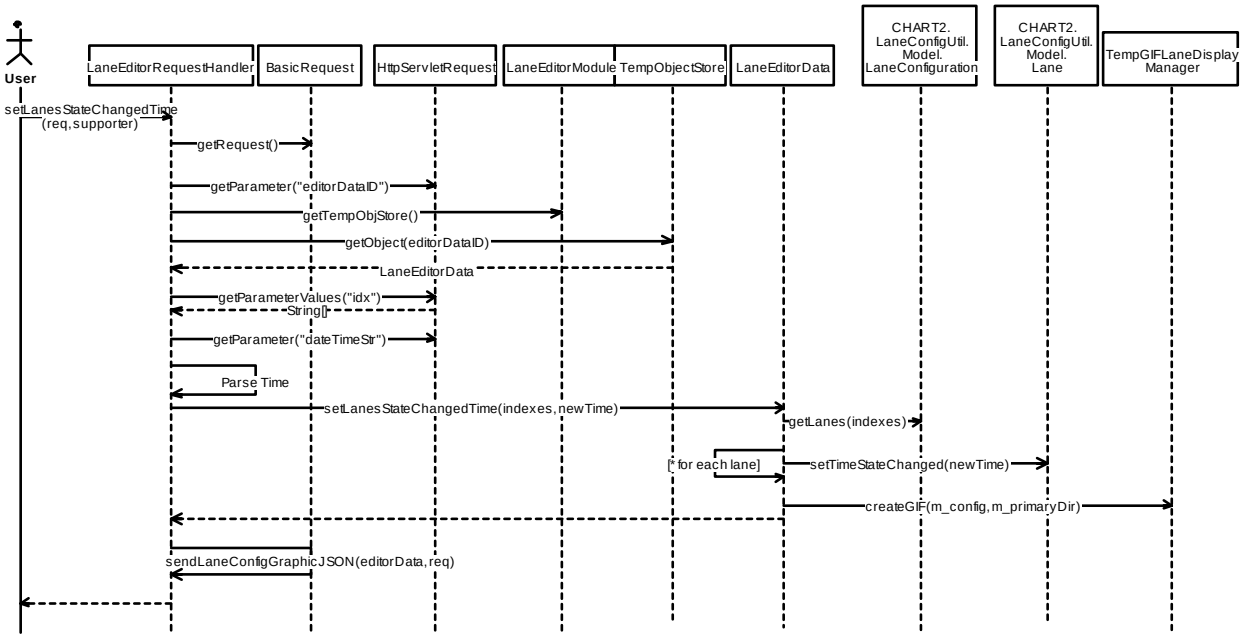


Figure 7-40 LaneEditorRequestHandler:setLanesStateChangedTime (Sequence Diagram)

7.6.2.16 LaneEditorRequestHandler:setLanesTrafficFlowDir (Sequence Diagram)

This diagram shows the processing when the lane traffic flow direction is set for one or more lanes. The editor data ID specified in the request is used to look up the LaneEditorData object from the module's TempObjectStore. The new lane direction and the lane index values are retrieved from request parameters, and the LaneEditorData is called to set the lanes' direction. It calls the lane configuration model to get the lanes, and for each lane it sets the traffic flow direction and state changed time if the traffic direction changed. A new GIF image is created to represent the configuration with the traffic flow direction changed, and the GIF image and associated metadata are sent back in the JSON response.

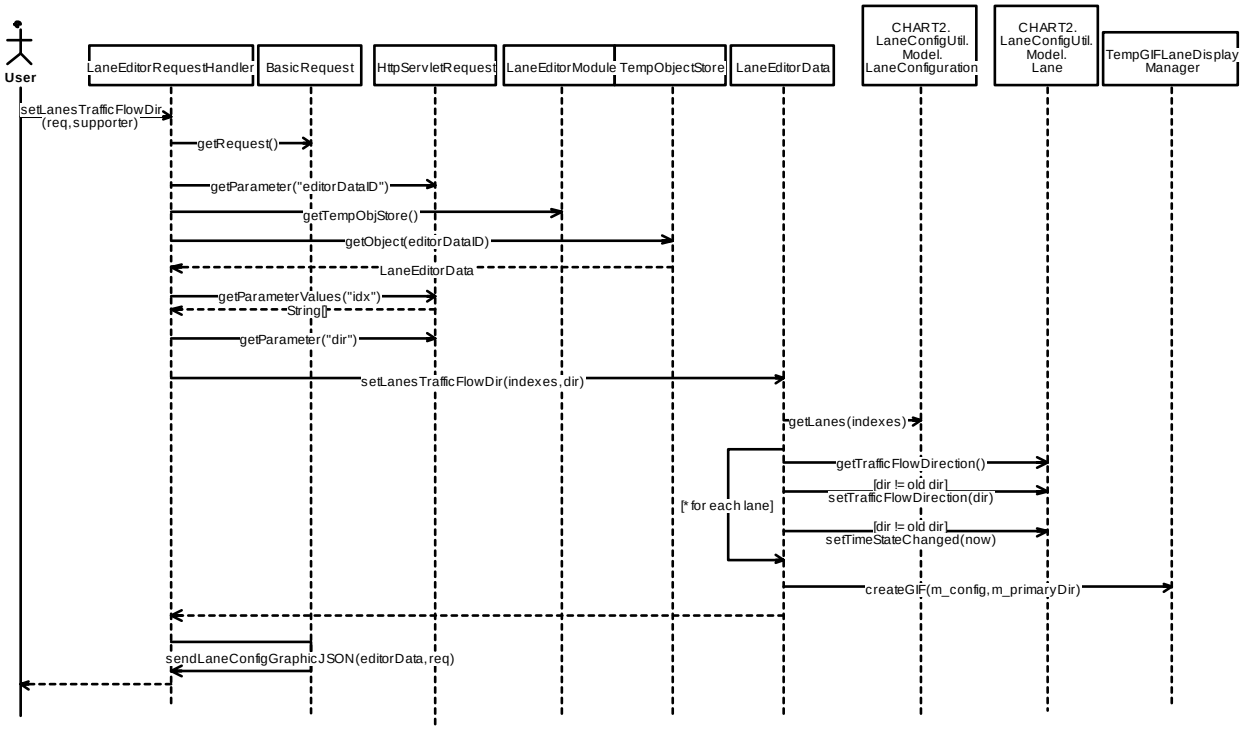


Figure 7-41 LaneEditorRequestHandler:setLanesTrafficFlowDir (Sequence Diagram)

7.6.2.16.1 LaneEditorRequestHandler: unmarshall (Sequence Diagram)

This diagram shows the processing that occurs to translate XML into java objects using the JAXB tool.

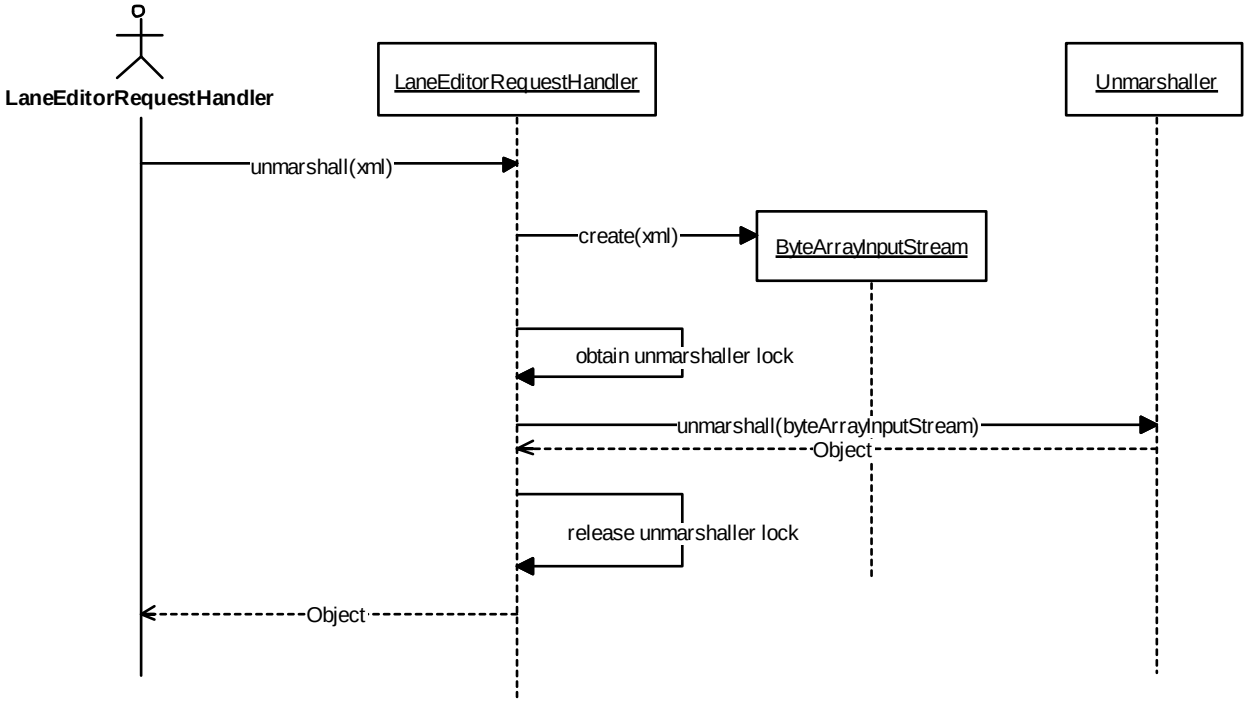


Figure 7-42 LaneEditorRequestHandler: unmarshall (Sequence Diagram)

7.6.2.17 LaneEditorRequestHandler:useNearbyLaneConfig (Sequence Diagram)

This diagram shows how a nearby lane configuration is selected for use. The editor data ID request parameter is used to look up the LaneEditorData that is stored in the module's TempObjectStore. The configuration index is read as another request parameter, and LaneEditorData is called with this index, which it uses to look up the corresponding NearbyLaneConfigInfo. The LaneConfiguration object is retrieved, and a copy is made (so that future editing will not affect the original). If the feature to initialize lanes in the primary direction to "unknown" status is enabled, the lane configuration is called to do this. A new GIF is created via the editor Data's Lane Display Manager object. Finally a JSON-encoded object representing the new lane graphic and its metadata is sent back in the response.

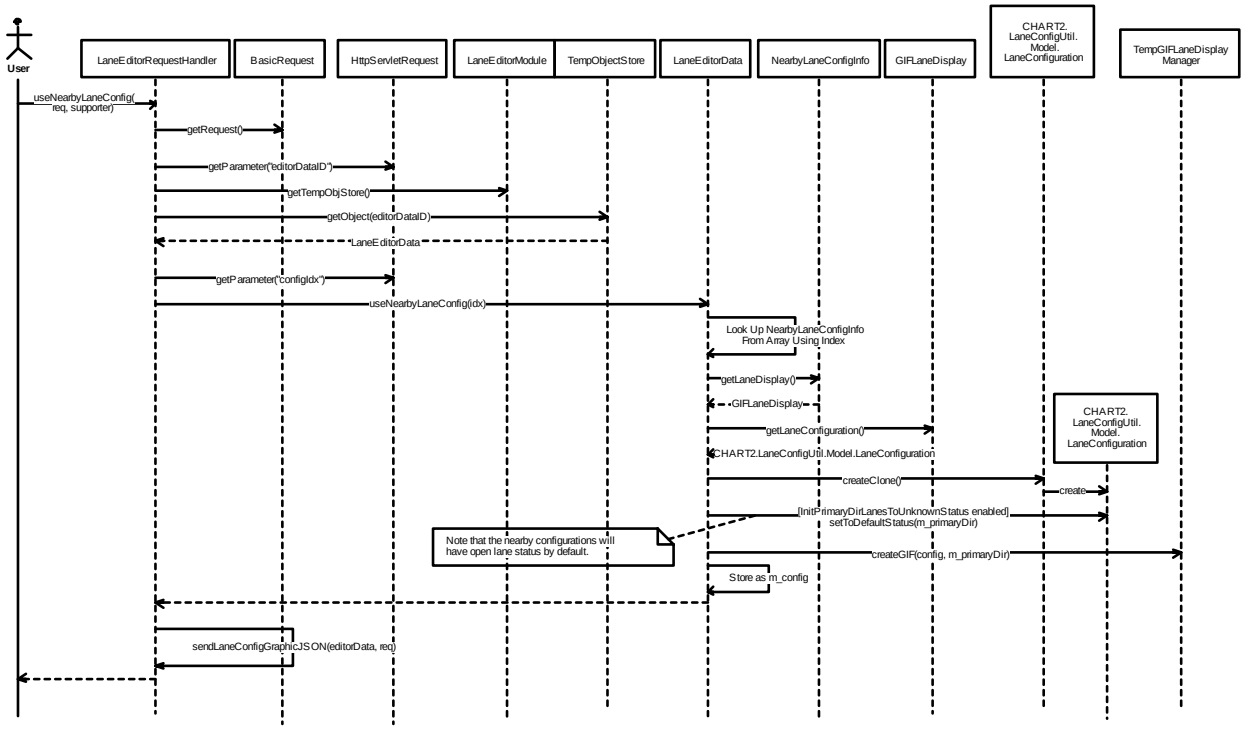


Figure 7-43 LaneEditorRequestHandler:useNearbyLaneConfig (Sequence Diagram)

7.6.2.18 LaneEditorRequestHandler:useStandardLaneConfig (Sequence Diagram)

This diagram shows how a standard lane configuration is selected for use. The editor data ID request parameter is used to look up the LaneEditorData that is stored in the module's TempObjectStore. The configuration index is read as another request parameter, and LaneEditorData is called with this index, which it uses to look up the corresponding GIFLaneDisplay object. The LaneConfiguration object is retrieved, and a copy is made (so that future editing will not affect the original). If the feature to initialize lanes in the primary direction to "unknown" status is enabled, the lane configuration is called to do this. A new GIF is created via the editor Data's Lane Display Manager object. Finally a JSON-encoded object representing the new lane graphic and its metadata is sent back in the response.

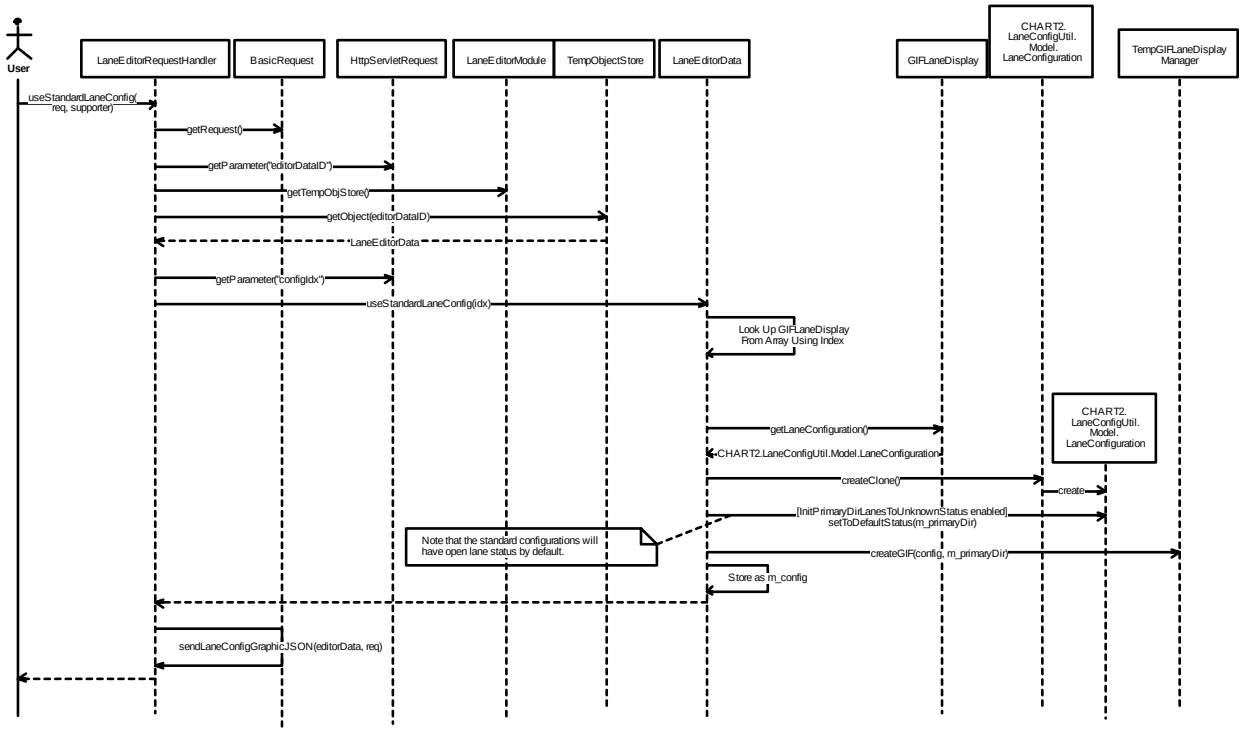


Figure 7-44 LaneEditorRequestHandler:useStandardLaneConfig (Sequence Diagram)

7.6.2.19 LaneEditorRequestHandler:viewLaneEditor (Sequence Diagram)

This diagram shows the processing that is performed when a request is received to view the lane editor. The lane editor must have already been initialized, and the ID returned from the initialization request is passed in as a parameter to this request. The LaneEditorData object which is used to persist information about the editor while the editor is in use and was created as part of editor initialization is retrieved from the temp object store. If the editor data is not found, html will be returned that displays an error message. If the editor data is found, it is placed in the context, along with the lane types available for selection when the user chooses to add a lane. The LaneEditor.vm template is used to dynamically generate html that will be returned to the browser.

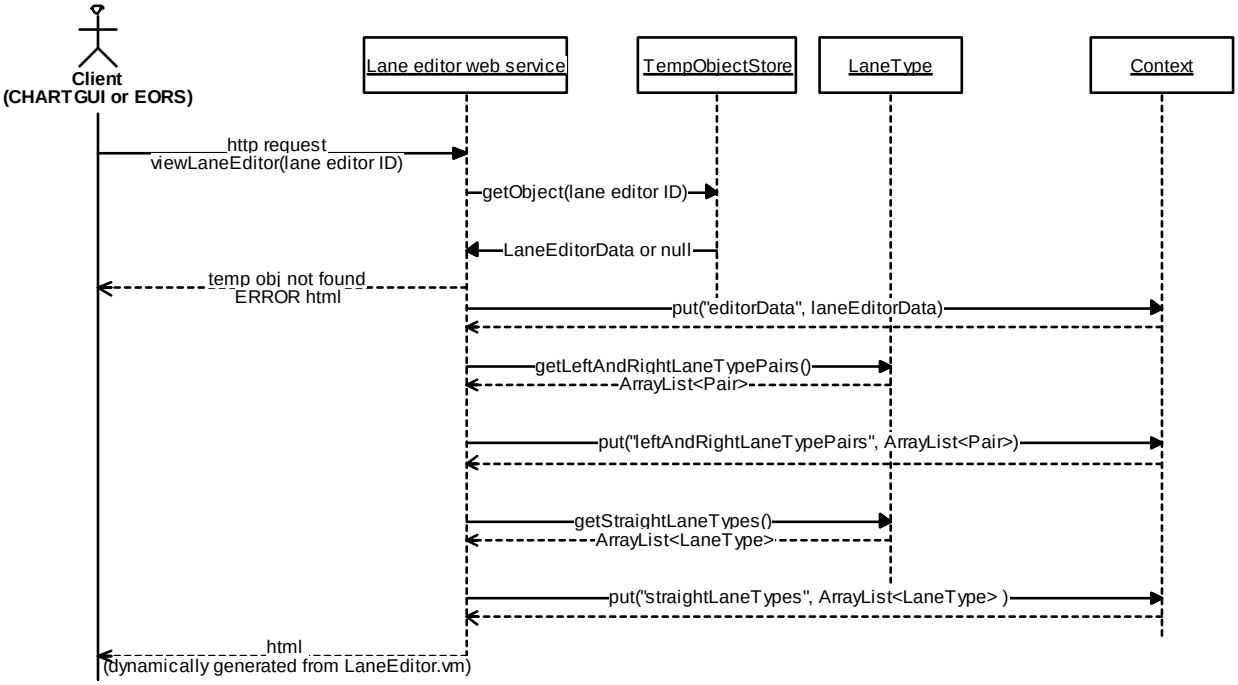


Figure 7-45 LaneEditorRequestHandler:viewLaneEditor (Sequence Diagram)

7.7 webservices.wsutil

7.7.1 Class Diagrams

7.7.1.1 webservices.util-classes (Class Diagram)

This diagram shows utility classes that are used when calling a remote REST based web service (XML over HTTP).

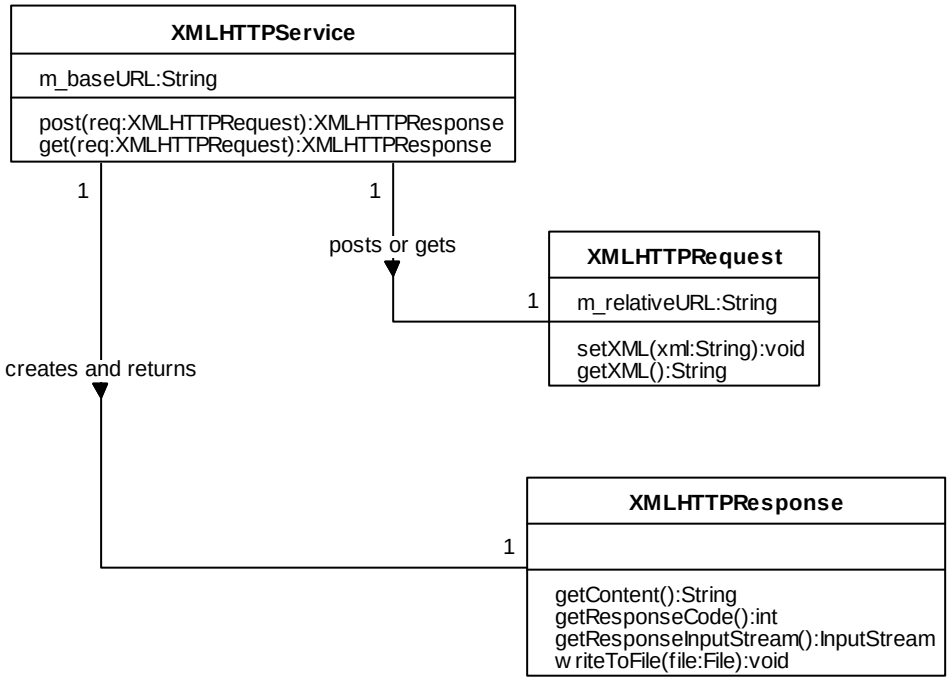


Figure 7-46 webservices.util-classes (Class Diagram)

7.7.1.1.1 XMLHTTPRequest (Class)

This class represents an HTTP request that can be sent (get or post) to an XMLHTTPService. The request has a relative URL that will be combined with the base URL of the service it is sent to and also has a method to set the request body String.

7.7.1.1.2 XMLHTTPResponse (Class)

This class represents a response that is returned from a get or post operation performed on an XMLHTTPService. It provides accessory methods for checking the response code and getting the returned response data.

7.7.1.1.3 XMLHTTPService (Class)

This class represents a remote XML/HTTP based web service at a specified URL. It supports operations to perform HTTP get and post operations on the remote service.

7.8 webservices.wsutil.jaxbutil

7.8.1 Class Diagrams

7.8.1.1 JAXBUtilClasses (Class Diagram)

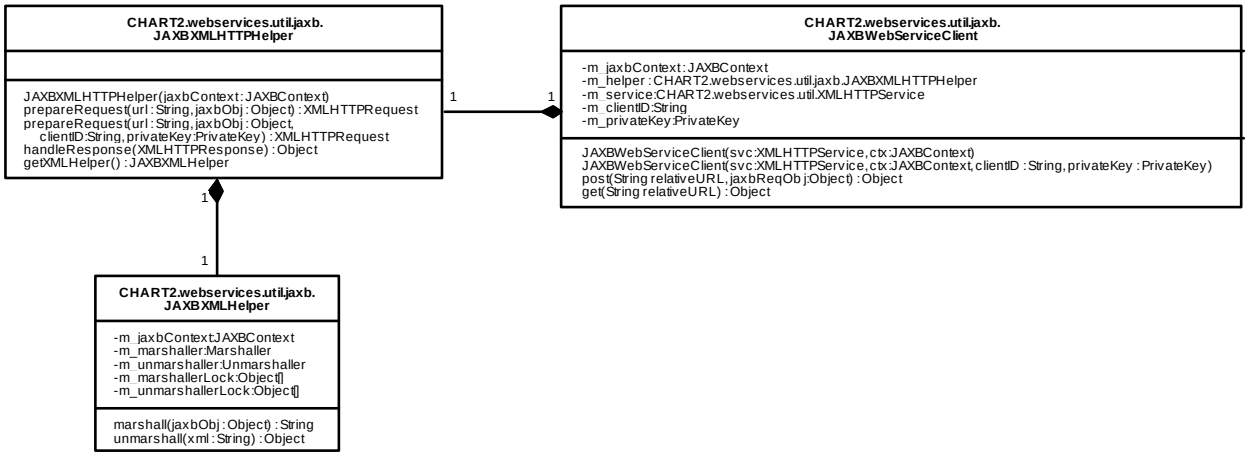


Figure 7-47 JAXBUtilClasses (Class Diagram)

7.8.1.1.1 CHART2.webservices.util.jaxb. JAXBWebServiceClient (Class)

This class is a wrapper for an XMLHTTPService that makes it easy to send/receive data to/from the service using objects that are generated via xsd and JAXB.

7.8.1.1.2 CHART2.webservices.util.jaxb. JAXBXMLHelper (Class)

This class handles marshalling and unmarshalling between XML and JAXB objects.

7.8.1.1.3 CHART2.webservices.util.jaxb. JAXBXMLHTTPHelper (Class)

This class can be used to prepare requests using JAXB objects, with or without authentication. It can also be used to extract the JAXB object from the XMLHTTPResponse. All marshalling / unmarshalling is handled internally.

7.8.2 Sequence Diagrams

7.8.2.1 JAXBWebServiceClient:get (Sequence Diagram)

This diagram shows processing performed by the JAXBWebServiceClient's get() method. This method is used to issue a request to a web service without passing XML as part of the request, and processes the response into a JAXB generated object. An XMLHTTPRequest is prepared and passed to the XMLHTTPService's get() method. The XMLHTTPResponse that is returned is passed to the JAXBXMLHTTPHelper.handleResponse() method to unmarshall the returned XML into a JAXB generated object that corresponds to the XML document root. This object is returned to the caller, who can check the type and cast as needed.

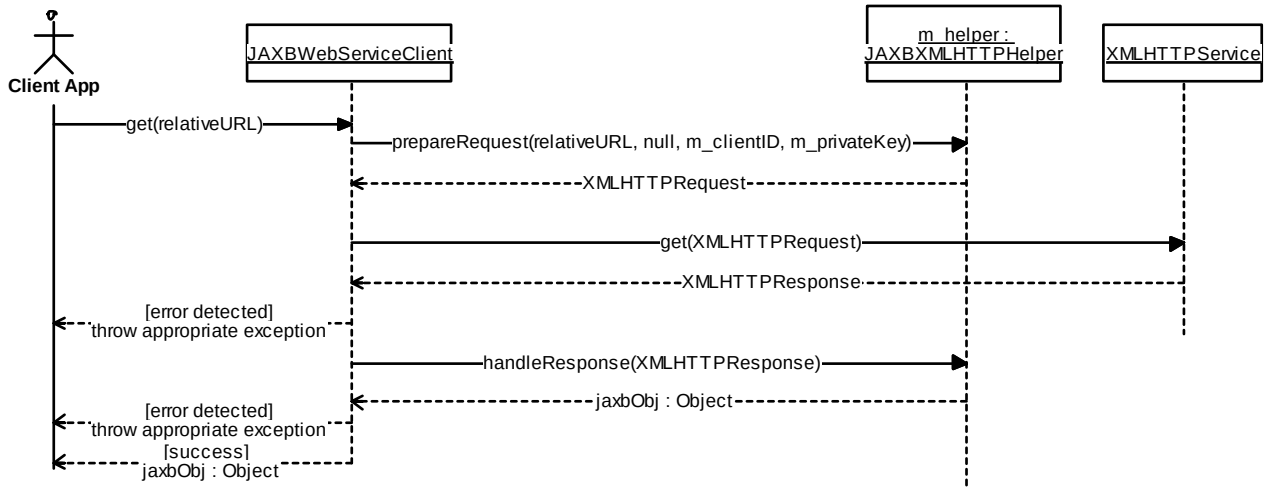


Figure 7-48 JAXBWebServiceClient:get (Sequence Diagram)

7.8.2.2 JAXBWebServiceClient:post (Sequence Diagram)

This diagram shows the processing that is performed by the JAXBWebServiceClient post() method, used to post request data to a web service as XML. The given object (which must be defined by XSD and generated by JAXB) is passed to the JAXBXMLHTTPHelper.prepareRequest() method where it is marshalled into XML and stored in a request. The post() method of the XMLHTTPService class is called to send the request to the specified URL and an XMLHTTPResponse is returned. The AXBXMLHTTPHelper.handleResponse() method is called to process the XMLHTTPResponse, unmarshalling the XML and returning the JAXB generated object that corresponds to the XML document root. This object is returned to the caller that can check its type and cast it to the appropriate class.

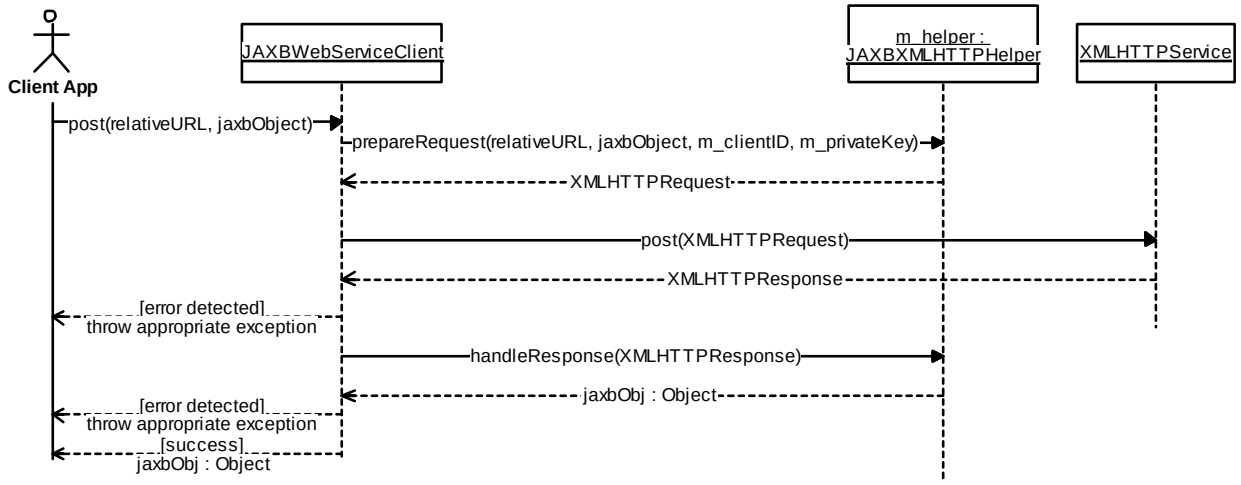


Figure 7-49 JAXBWebServiceClient:post (Sequence Diagram)

7.8.2.3 JAXBXMLHTTPHelper:handleResponse (Sequence Diagram)

This diagram shows the processing that is performed by the handleResponse() utility method. It is used to process an XMLHTTPResponse to unmarshall the response XML into a JAXB generated object.

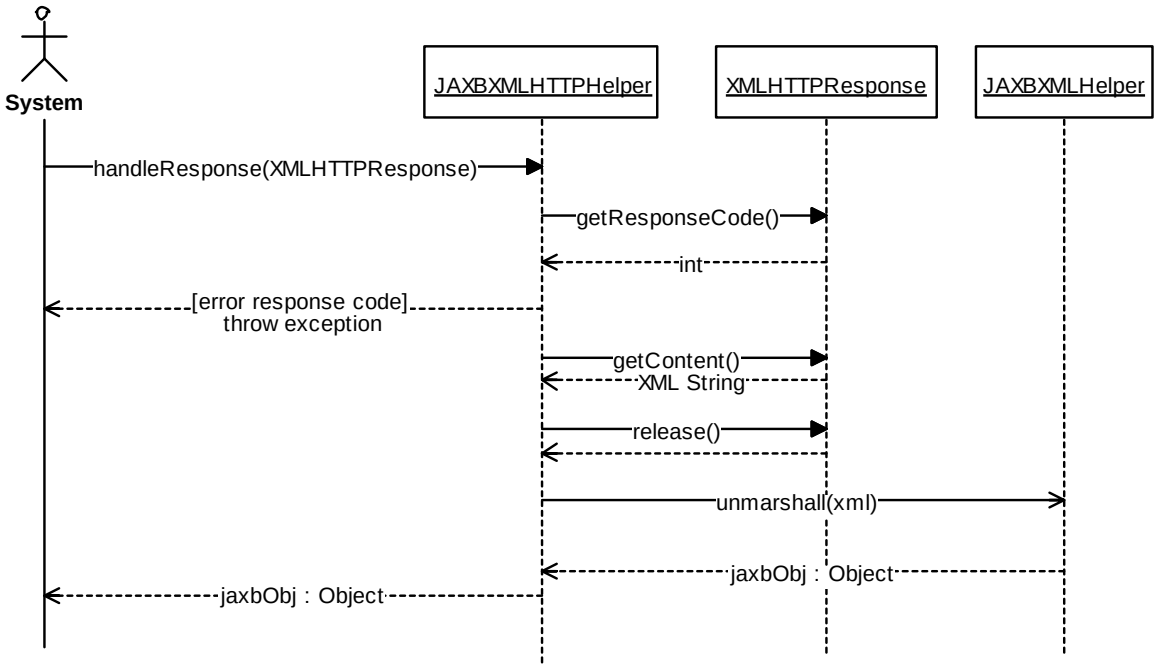


Figure 7-50 JAXBXMLHTTPHelper:handleResponse (Sequence Diagram)

7.8.2.4 JAXBXMLHTTPHelper:marshall (Sequence Diagram)

This diagram shows how a JAXB obect is converted to XML (marshalled). After creating an output stream, the marshaller lock is acquired, and if necessary the Marshaller object is created. (This will be used for any future marshalling). The Marshaller is then called to perform the work of marshalling.

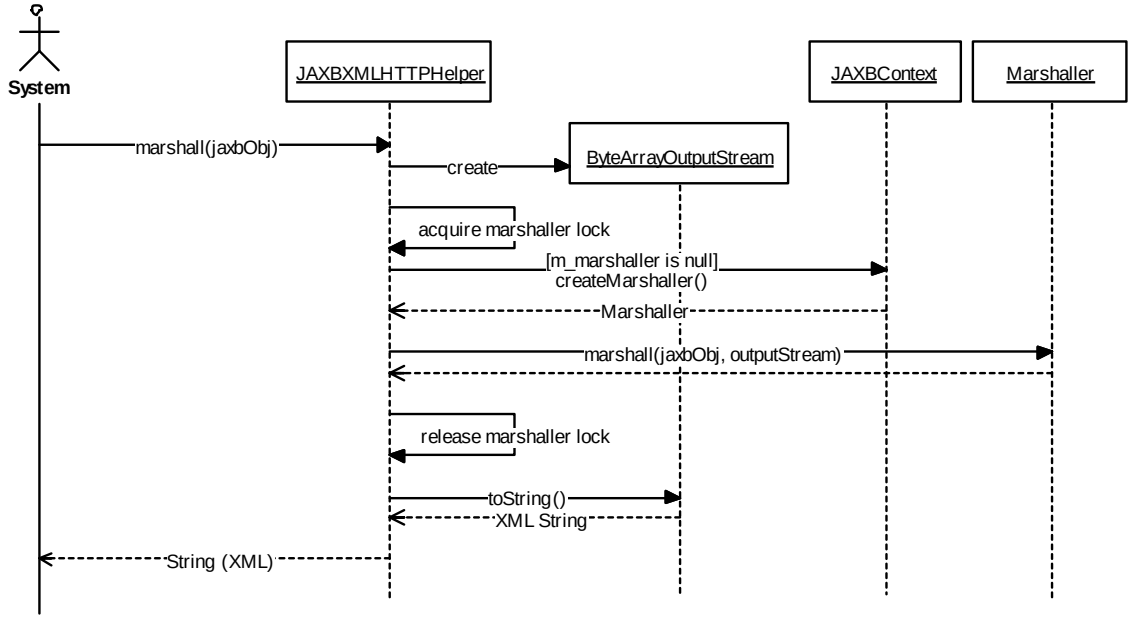


Figure 7-51 JAXBXMLHTTPHelper:marshall (Sequence Diagram)

7.8.2.5 JAXBXMLHTTPHelper:prepareRequest (Sequence Diagram)

This diagram shows the processing performed by the prepareRequest() utility method. It creates an XMLHTTPRequest and sets the clientID and private key into the request (if any). If a JAXB generated object is provided, it is marshalled into XML and also included in the request. The request is then returned to the caller.

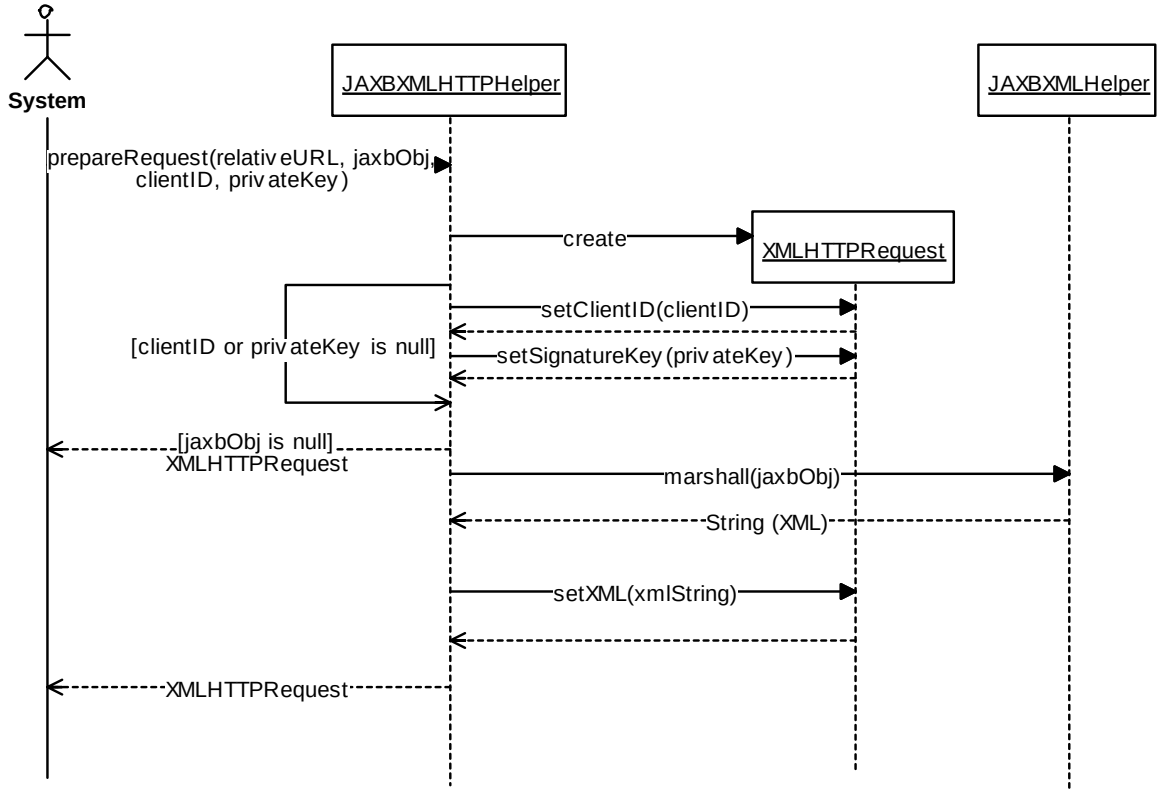


Figure 7-52 JAXBXMLHTTPHelper:prepareRequest (Sequence Diagram)

7.8.2.6 JAXBXMLHTTPHelper:unmarshall (Sequence Diagram)

This diagram shows how XML is converted into a JAXB obect (unmarshalled). After creating an input stream, the unmarshaller lock is acquired, and if necessary the Unmarshaller object is created. (This will be used for any future unmarshalling). The Unmarshaller is then called to perform the work of unmarshalling.

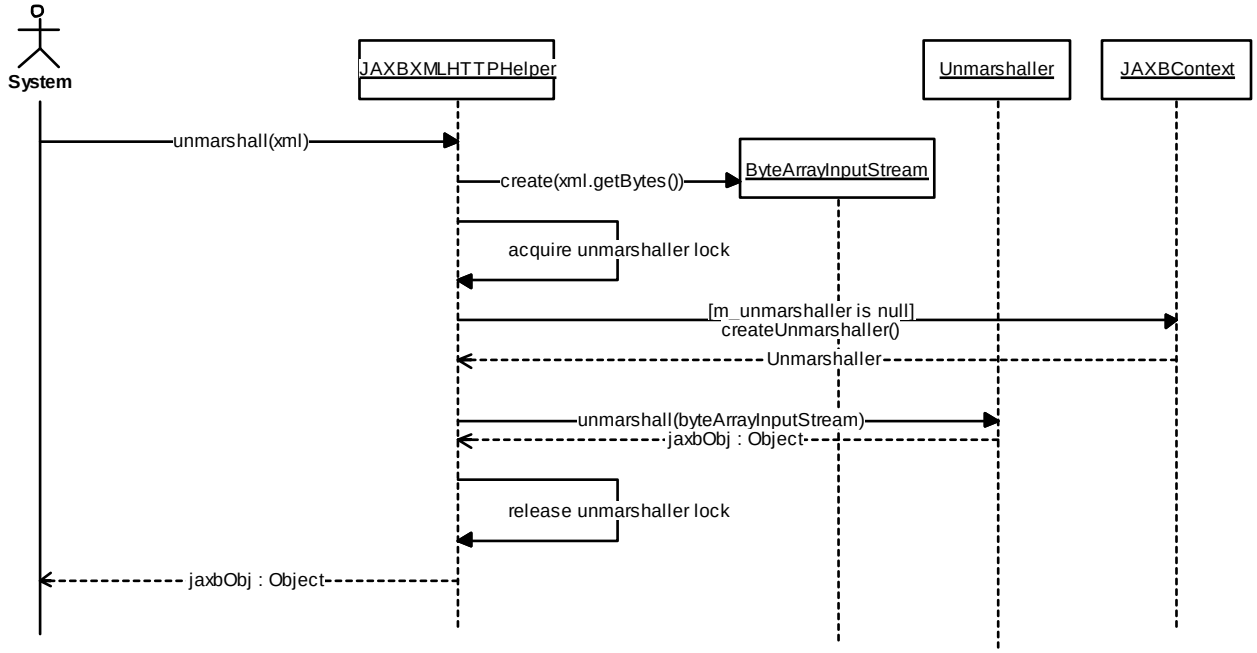


Figure 7-53 JAXBXMLHTTPHelper:unmarshall (Sequence Diagram)

7.9 xsdutil.common

7.9.1 Class Diagrams

7.9.1.1 XSDUtilCommonClasses (Class Diagram)

This diagram contains utility classes for dealing with the CHART2.xsd.laneconfig JAXB-generated classes.

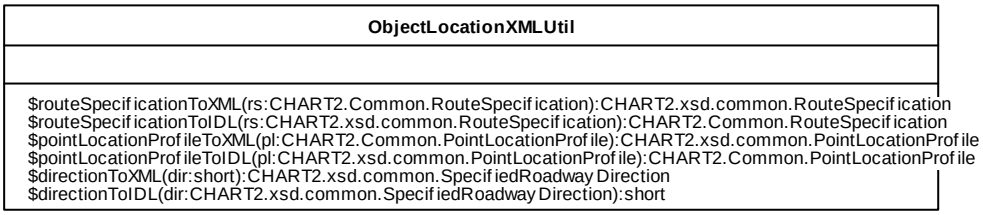


Figure 7-54 XSDUtilCommonClasses (Class Diagram)

7.9.1.1.1 ObjectLocationXMLUtil (Class)

This class contains utility methods that are helpful in converting object location data from IDL to XSD classes and vice versa.

7.10 xsdutil.laneconfig

7.10.1 Class Diagrams

7.10.1.1 XSDUtilLaneConfigClasses (Class Diagram)

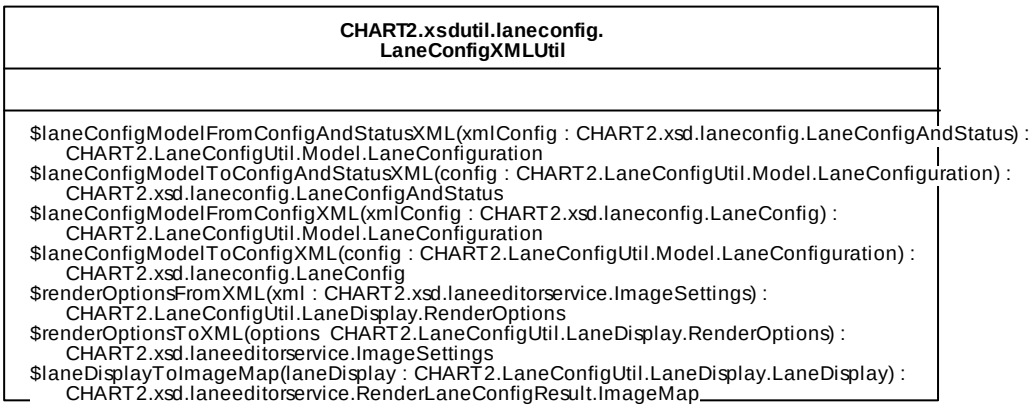


Figure 7-55 XSDUtilLaneConfigClasses (Class Diagram)

7.10.1.1.1 CHART2.xsdutil.laneconfig. LaneConfigXMLUtil (Class)

This class contains functionality to convert to / from CHART2.xsd.laneconfig classes to other (non-JAXB) classes used by CHART.

7.11 Mapping - Lane Config Web Service

7.11.1 Class Diagrams

7.11.1.1 GISLaneConfig (Class Diagram)

This diagram shows classes generated from the XSD schema and will be used to create getLaneconfigRequest, getLaneconfigResult, storeLaneConfig, requestParam and Result objects. These .net classes were generated with xsd.exe tool.

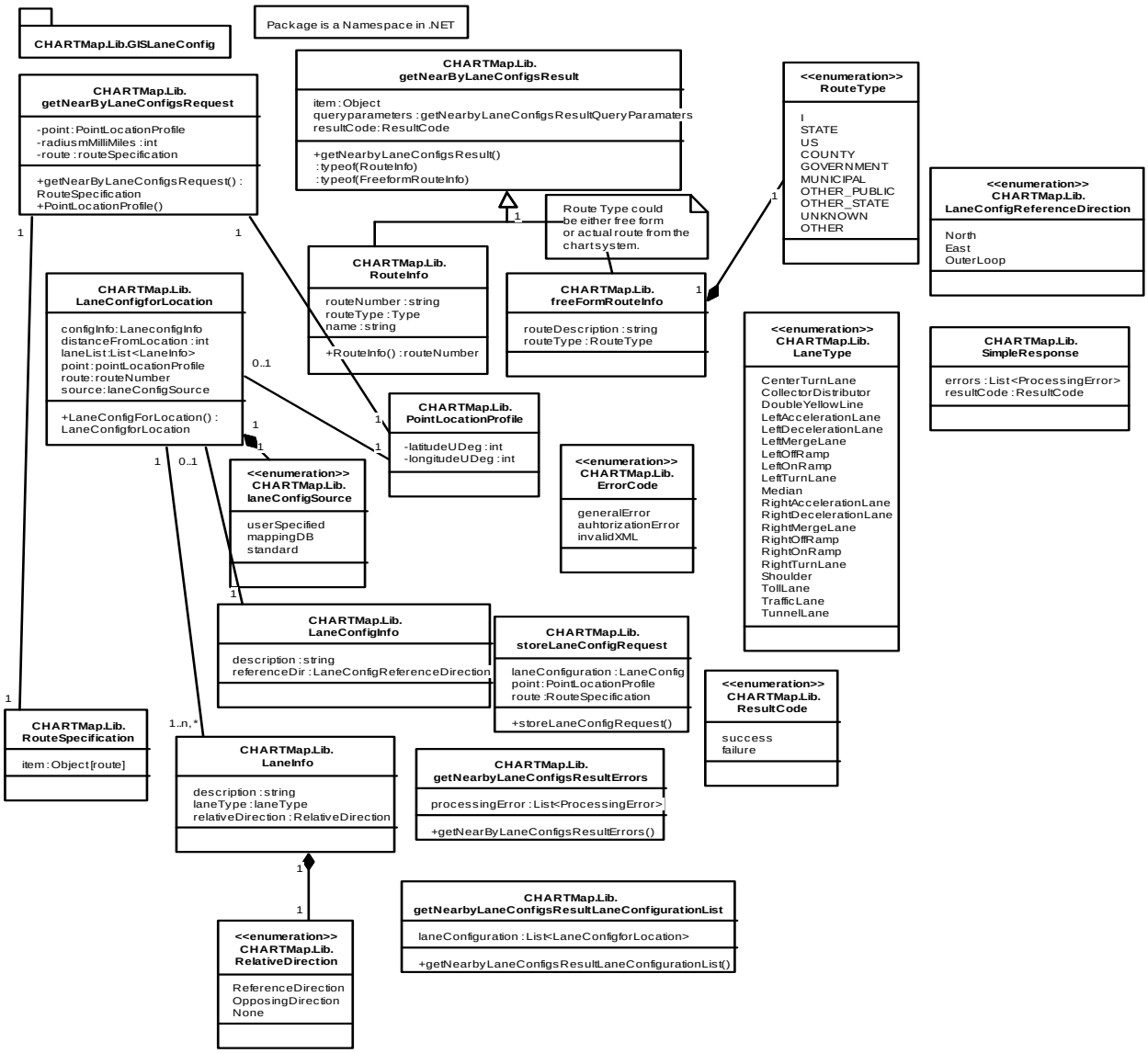


Figure 7-56 GISLaneConfig (Class Diagram)

7.11.1.1.1 <<enumeration>> CHARTMap.Lib. ErrorCode (Class)

This class will be used to generate error code with regards to general Error, Error during authorization and invalid XML.

7.11.1.1.2 <<enumeration>> CHARTMap.Lib. LaneConfigReferenceDirection (Class)

The laneConfigReferenceDirection will carry an information for reference direction per Lane.

7.11.1.1.3 <<enumeration>> CHARTMap.Lib. laneConfigSource (Class)

The laneConfigSource enum carry an information for the source of a lane config.

7.11.1.1.4 <<enumeration>> CHARTMap.Lib. LaneType (Class)

This enum class would provide the Lane Types dfined in CHART System for different lane types example : Median, Shoulder etc.

7.11.1.1.5 <<enumeration>> CHARTMap.Lib. RelativeDirection (Class)

The RelativeDirection class will carry an information for Reference and opposite direction as a part of the getLaneConfigRequest xml request.

7.11.1.1.6 <<enumeration>> CHARTMap.Lib. ResultCode (Class)

The Resultcode enum class will carry success or failure result types.

7.11.1.1.7 <<enumeration>> RouteType (Class)

This enum class would provide the Route Types in CHART System for Interstate, State, US , public and govt routes.

7.11.1.1.8 CHARTMap.Lib. freeFormRouteInfo (Class)

The freeFormRouteInfo class will be used to deserialize a free form Route object , which is part of a getNearByLaneConfigsRequest xml request.

7.11.1.1.9 CHARTMap.Lib. getNearByLaneConfigsRequest (Class)

getLaneConfigsRequest class would use point , radius in milli miles and route as a Type. The class will be used to create a getlaneconfigrequest object for lane configuration.

7.11.1.1.10 CHARTMap.Lib. getNearByLaneConfigsResult (Class)

The getNearByLaneConfigsResult class will be used to create a result object in response to getNearByLaneConfigsRequest xml request.

7.11.1.1.11 CHARTMap.Lib. getNearbyLaneConfigsResultErrors (Class)

getNearByLaneConfigsResultsError class will carry list of processing errors.

7.11.1.1.12CHARTMap.Lib. getNearbyLaneConfigsResultLaneConfigurationList (Class)

The getNearbyLaneConfigsResultLaneConfigurationList class will carry a List of Lane configurations. This class is inherited from getNearbyLaneConfigsResult.

7.11.1.1.13CHARTMap.Lib. LaneConfigforLocation (Class)

The LaneConfigforLocation class will be used to create a list object of LaneConfigInfo, distance , route as a part of a getNearByLaneConfigsResult object and in response to getNearByLaneConfigsRequest xml request.

7.11.1.1.14CHARTMap.Lib. LaneConfigInfo (Class)

The LaneConfigInfo class will carry information for lane description and reference direction for the respective lane configuration.

7.11.1.1.15CHARTMap.Lib. LaneInfo (Class)

The laneInfo class will carry an information for lane description , lane types and relative direction.

7.11.1.1.16CHARTMap.Lib. PointLocationProfile (Class)

The pointlocationprofile class will carry information for Latitude and longitude. This class is inherited from storeLaneConfigRequest and getLaneConfigRequest.

7.11.1.1.17CHARTMap.Lib. RouteInfo (Class)

The RouteInfo class will be used to deserialize a Route object , which is part of a getNearByLaneConfigsRequest xml request.

7.11.1.1.18CHARTMap.Lib. RouteSpecification (Class)

The RouteSepecification class will carry an object Route.

7.11.1.1.19CHARTMap.Lib. SimpleResponse (Class)

The SimpleResponse class defines errors (processing error List) and result code (type) class. The response will be added to the getLaneConfigResult object, if request is failed to process or a database related exception thrown.

7.11.1.1.20CHARTMap.Lib. storeLaneConfigRequest (Class)

The storeLaneConfigRequest class {type) will carry an object for storing lane configuration which includes lane configurations, location and route information.

7.11.1.2 GISLaneConfigHelper (Class Diagram)

This diagram shows classes for the GIS Lane Config web service, a web service that provides the ability to retrieve and store lane configurations based on location.

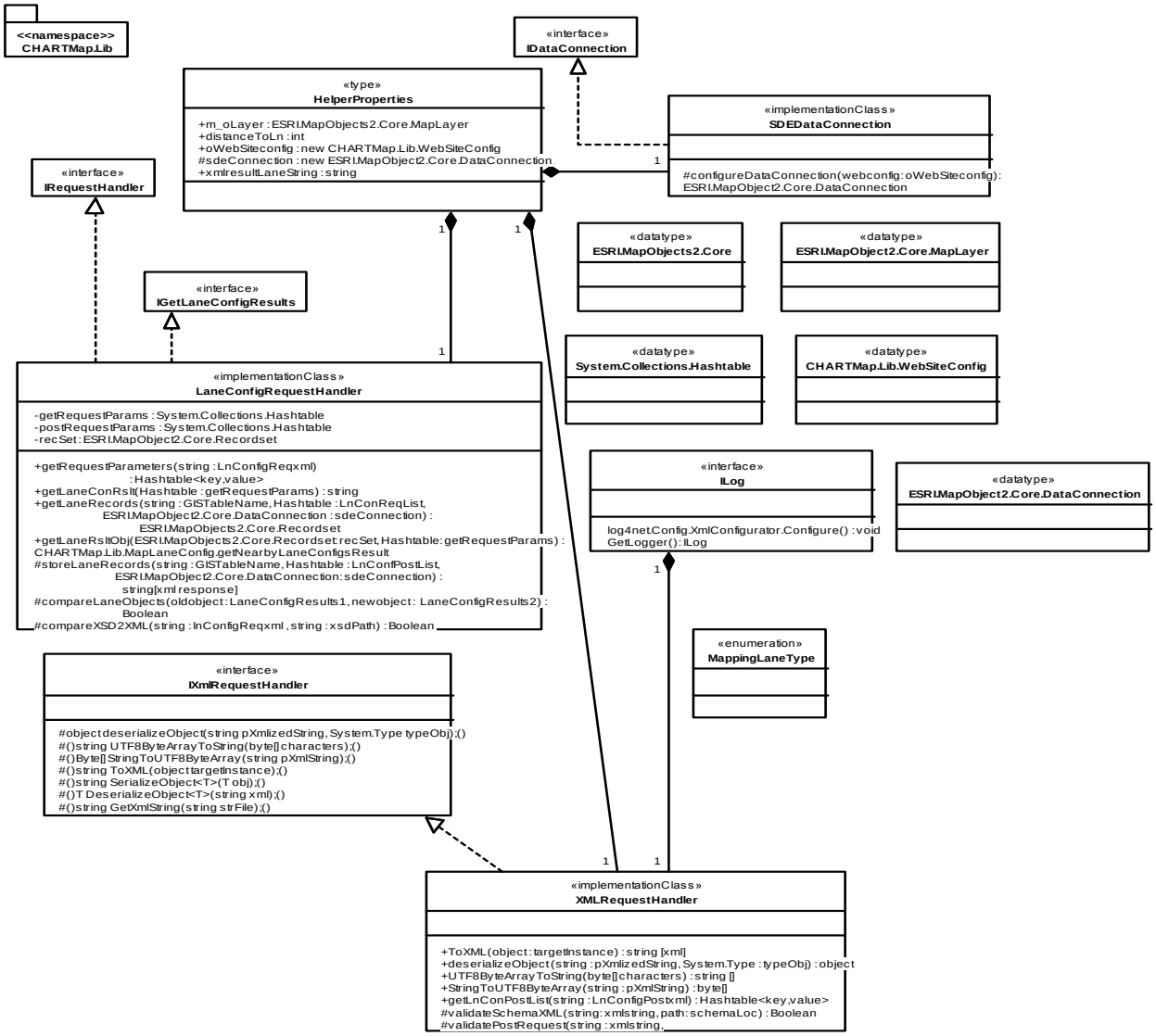


Figure 7-57 GISLaneConfigHelper (Class Diagram)

7.11.1.2.1 CHARTMap.Lib.WebSiteConfig (Class)

This data type is derived from the .net WebConfiguration properties of Systems.configurations. The first WebConfig instance will be created from the Systems.configurations class.

7.11.1.2.2 ESRI.MapObject2.Core.DataConnection (Class)

The DataConnection data type is derived from the ESRI MapObject assembly. Type will be used for Dataconnection string to connect to ArcSDE.

7.11.1.2.3 ESRI.MapObject2.Core.MapLayer (Class)

This data type is derived from ESRI Mapobject.core library and will contain DataConnection and MapLayer objects.

7.11.1.2.4 ESRI.MapObjects2.Core (Class)

This data type is derived from the ESRI MapObject .net library. The first MapObject instance will be created from Core class.

7.11.1.2.5 HelperProperties (Class)

The properties of the Helper class can be accessed through the LaneConfigRequestHandler and xmlRequestHandler class .

7.11.1.2.6 IDataConnection (Class)

This interface defines the dataconnection methods that each module must implement in order to connect to the SDE Database types within the web service framework.

7.11.1.2.7 IGetLaneConfigResults (Class)

This interface will be used to implement LaneConfigRequestHandler. The interface contains methods to implement for LaneConfigRequestHandler.

7.11.1.2.8 ILog (Class)

This ILog interafce is derived from the .net Log4Net assembly. The implementation will contain WriteLog() methods and will be used to write log entries during database connection, deserialize. serialize processes.

7.11.1.2.9 IRequestHandler (Class)

This class is a request handler (.net Interface) to intialize the request.

7.11.1.2.10 IXmlRequestHandler (Class)

This interface will contain methods to process LaneConfig XMLs and Objects.

7.11.1.2.11 LaneConfigRequestHandler (Class)

The LaneConfigRequestHandler class will contain methods to process LaneConfiguration request XML, process to convert dataTypes, serialize and deserialize LaneConfig Objects.

7.11.1.2.12 MappingLaneType (Class)

The enumerations for Lane Types defined in the Mapping Lane Config Centerline table.

7.11.1.2.13 SDEDataConnection (Class)

This class will connect to the ArcSDE Database using connection string from the

configuration settings. The database connections are configurable from the webconfig file. The Dataconnection class will be used from the ESRI MapObjects API library. The connection will be maintained for each user session and will be closed once database related codes are finished processing.

7.11.1.2.14System.Collections.Hashtable (Class)

This data type is derived from the .net System.collections namespace. The first HashTable instance will be created from the Collections class.

7.11.1.2.15XMLRequestHandler (Class)

This class will implement methods to process LaneConfig XMLs and Objects.

7.11.2 Sequence Diagrams

7.11.2.1 LaneConfigGISWebService:getLaneConfiguration (Sequence Diagram)

This sequence diagram shows the processes that is performed when a request is received to the GISLaneConfiguration web service as getLaneConfigurations. The request object will be accepted in XML format. The request object will be deserialized and request parameters will be stored in Hashtable (.net object). The request parameters will be used to query the lane configurations table and are explained in detail in getLaneConRslt sequence diagram. The errors during the processes will be reported back to the Requester as a part of Result XML.

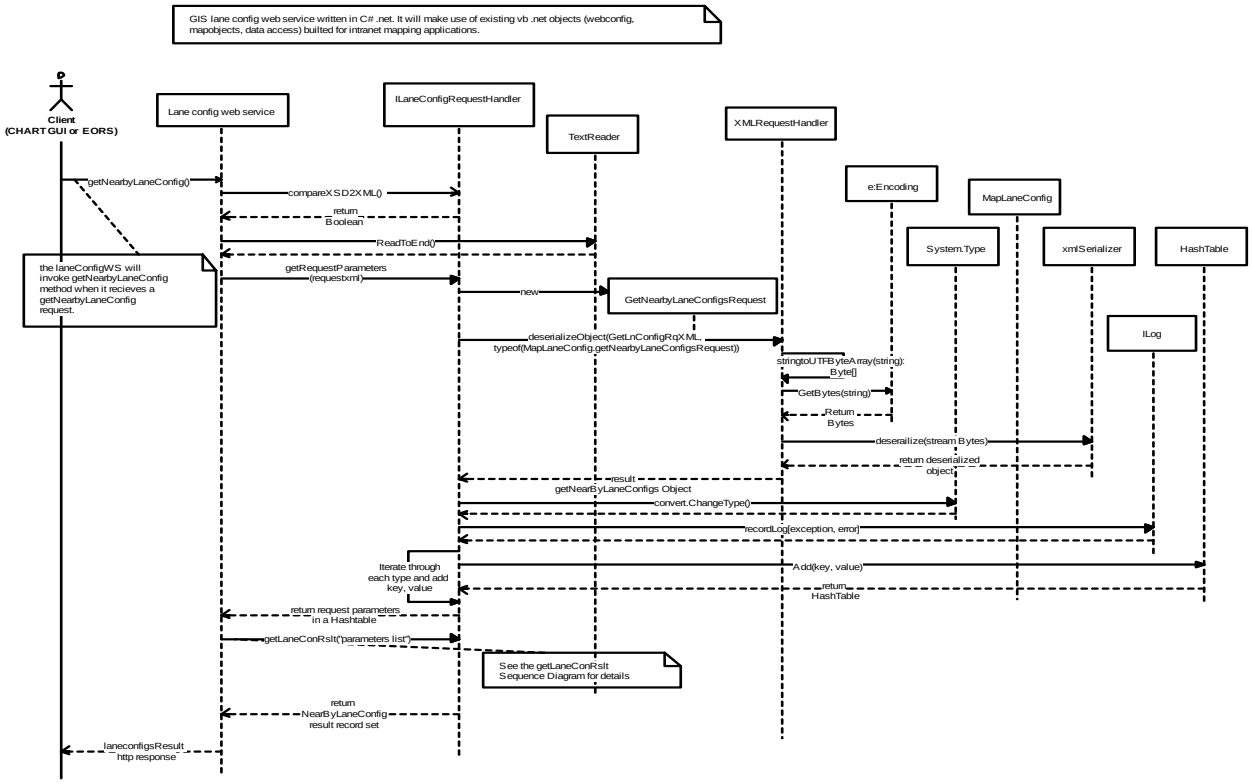


Figure 7-58 LaneConfigGISWebService:getLaneConfiguration (Sequence Diagram)

7.11.2.2 LaneConfigGISWebService:getLaneConRslt (Sequence Diagram)

This sequence diagram shows the processing that will be performed after invoking getLaneConRslt method. The processing will include connecting to the SDE data table with Mapobject (core ESRI MapObjects library). The lane configuration records will be returned after searching through the Spatial Table. The search criteria for querying lane configuration table will use input distance, location and route information (if available).

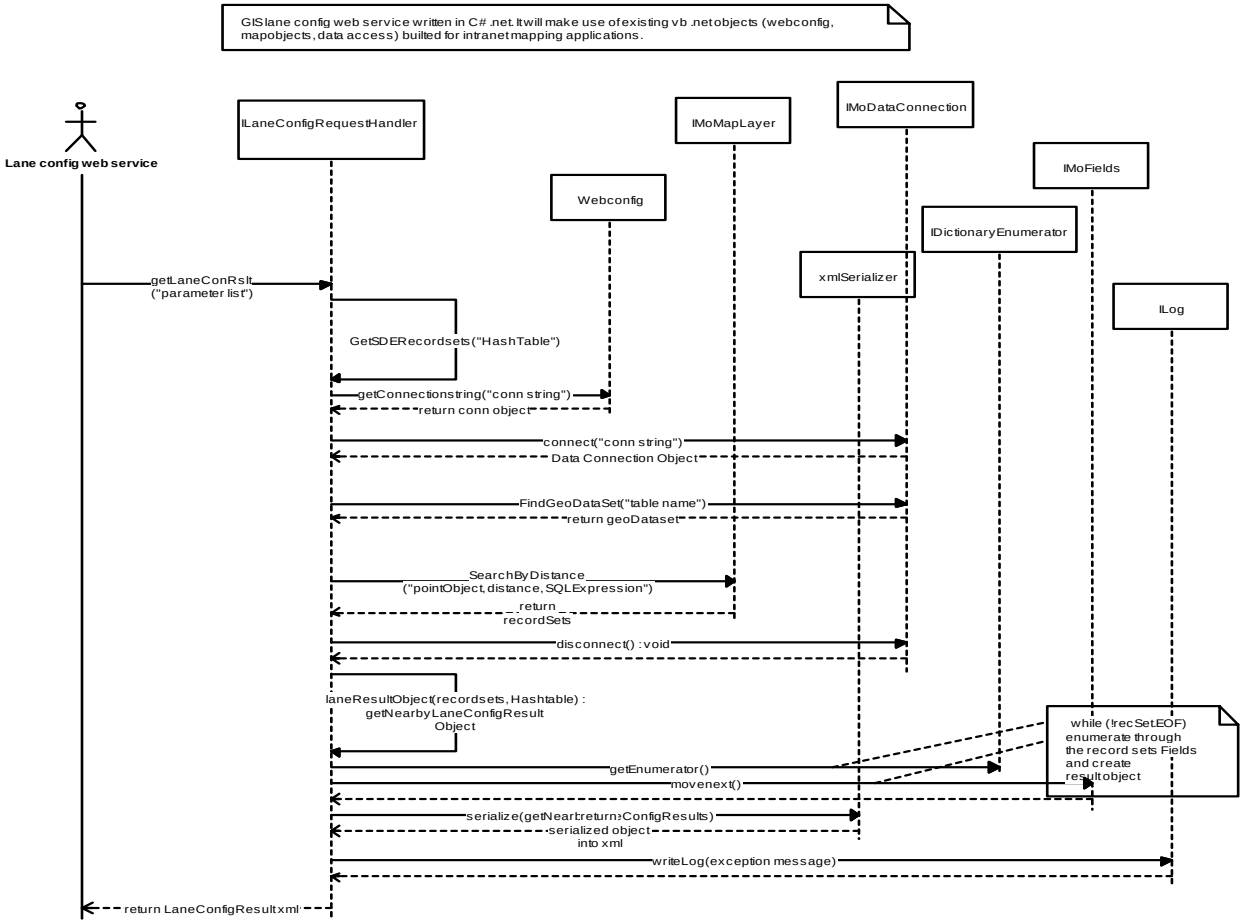


Figure 7-59 LaneConfigGISWebService:getLaneConRslt (Sequence Diagram)

7.11.2.3 LaneConfigGISWebService:storeLaneConfiguration (Sequence Diagram)

This diagram shows the processing that will be performed when a request is received to store a lane configuration for a location. The process includes authenticating the request by generating a signature using the client's public key and the XML data being submitted, and comparing it to the signature provided by the requesting client. If the signature matches, then the service will proceed to compare the Schema with the Request XML and then to deserialize the storeLane config xml request. A call is then made to storeLaneRecords() to store the lane configuration into the database.

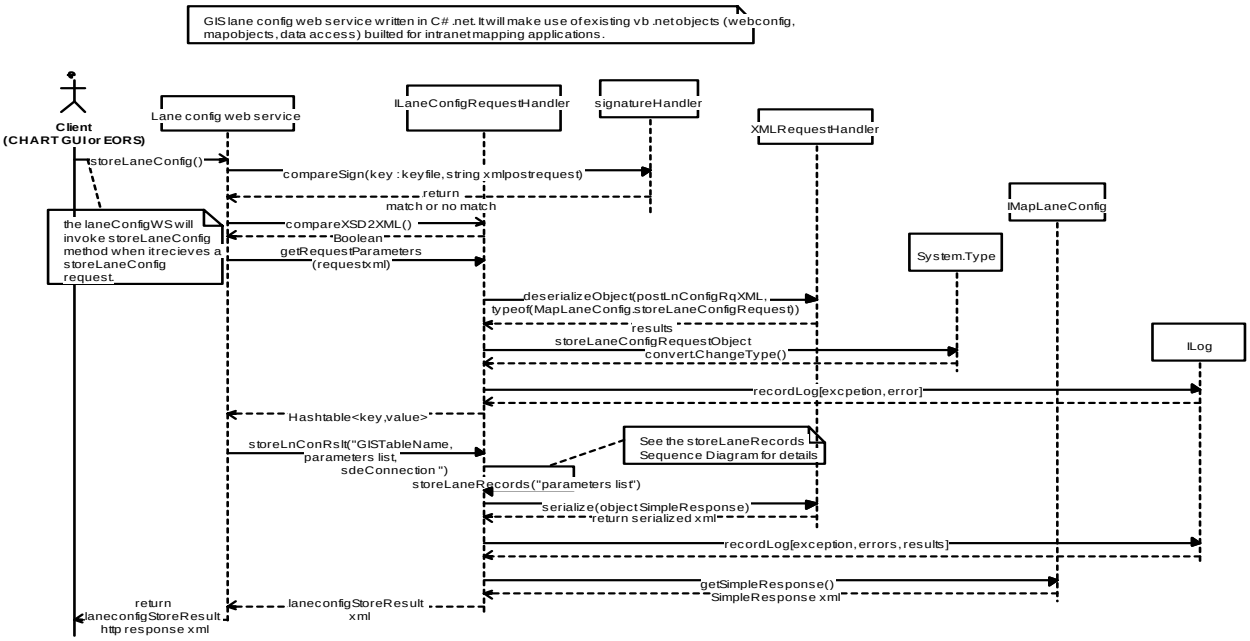


Figure 7-60 LaneConfigGISWebService:storeLaneConfiguration (Sequence Diagram)

7.11.2.4 LaneConfigGISWebService:storeLaneRecords (Sequence Diagram)

This diagram shows the processing that will be performed by the storeLaneRecords() method to store a lane configuration in the database. The process will include connecting to an SDE data table with Mapobject (core ESRI Mapobjects library). The Recordsets will be returned by searching the Spatial Table by using distance, location and query expression. The returned recordsets will then be compared with the existing Store Lane Config Request object. If the location returned by the returned recordset matches the Store Lane Config Request location, the new Lane Config will be stored in the Spatial Table. The new timestamp will be added to the Table and the Lane Config will be stored as a User Specified Lane configuration. The result XML will include the result code of success or failure messages.

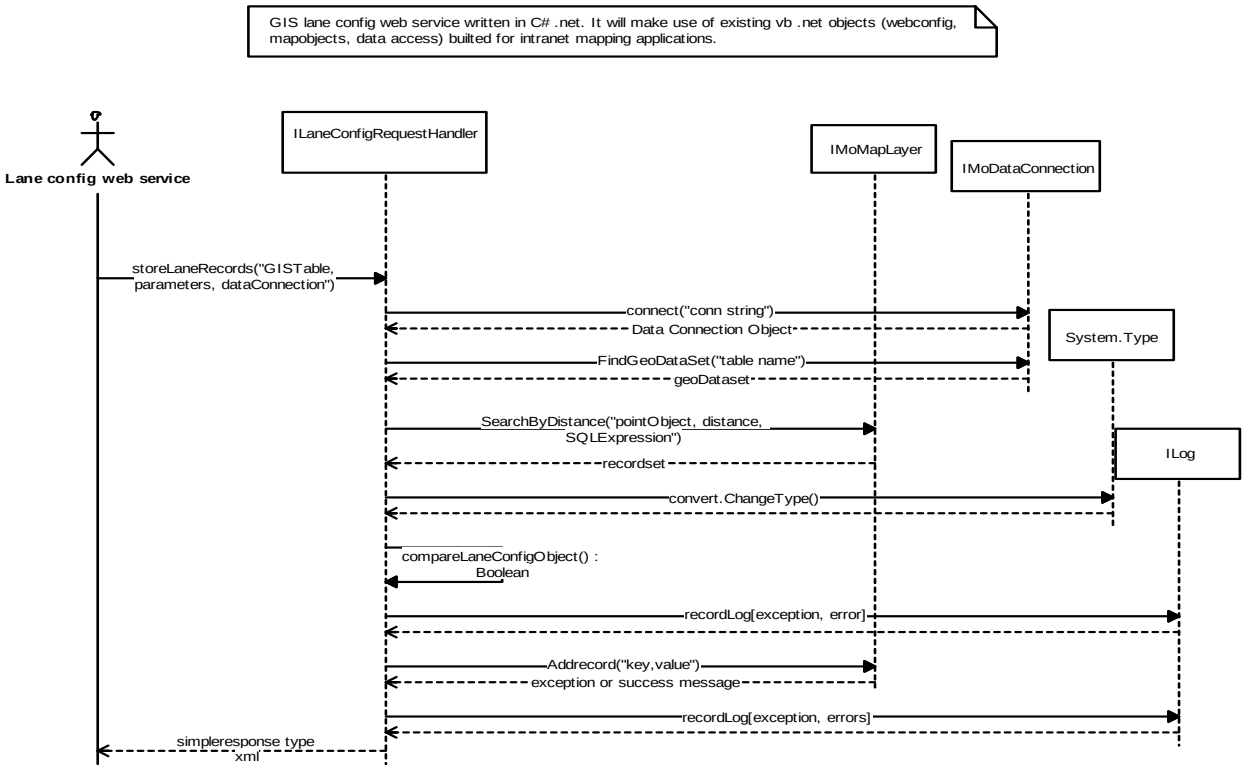


Figure 7-61 LaneConfigGISWebService:storeLaneRecords (Sequence Diagram)

8 Use Cases – Map Between Features

The use case diagrams depict new functionality for the CHART R6 Between Features update for the Integrated Map, and to identify existing features that will be enhanced. The use case diagrams for this feature exist in the Tau design tool in the Release6 area. The sections below indicate the title of the use case diagrams that apply to this feature.

8.1 CHART

8.1.1 R6HighLevel (Use Case Diagram)

This diagram shows the high level use cases for features added or modified as part of R6.

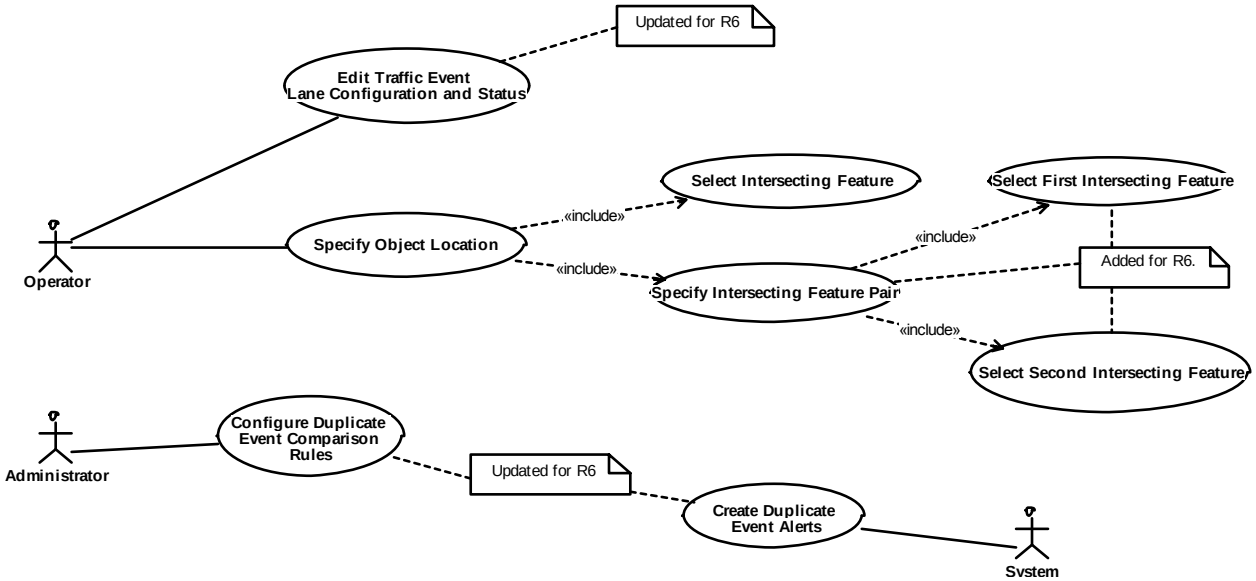


Figure 8-1 R6HighLevel (Use Case Diagram)

8.1.1.1 Configure Duplicate Event Comparison Rules (Use Case)

A user with proper functional rights can configure the rules used when comparing traffic events for the purpose of duplicate event detection. Combinations of event types which can be compared as duplicates can be configured. For one example, Incidents can be considered to be duplicates of other Incidents and Congestion Events, but no other event types. For another example, Safety Events can be configured to never be considered as duplicates of any other events (including even other Safety Events). The duplicate max comparison time can be configured. New for R6: The comparison distance can be configured. If two comparable events are geographically located within this distance, have the same primary route and the same direction, they will be considered duplicates.

8.1.1.2 Create Duplicate Event Alerts (Use Case)

The system creates a Duplicate Event Alert when two Traffic Events have the same location data and their Traffic Event Types are defined as being able to be compared (see the Configure Duplicate Events Comparison Rules use case in the Configure Alerts use case diagram). Two events will be considered to have the same location when they are geographically located with a configurable number of miles (in thousandths) from one another, they are located on the same route and have the same direction.

8.1.1.3 Edit Traffic Event Lane Configuration and Status (Use Case)

An operator with the manage traffic events user right may edit the lane status of a traffic event, including changing direction for a particular lane. This only applies to open Planned Roadway Closures, Incidents, and Special Events.

8.1.1.4 Select First Intersecting Feature (Use Case)

The operator will select an intersecting feature marking the beginning of the interval on the roadway. If geolocation information is available for this feature it will be used as the geolocation data for the object.

8.1.1.5 Select Intersecting Feature (Use Case)

A user may select an intersecting feature along a primary route when defining the location of an object. The user may specify that the location is at, past, prior to, east of, west of, north of, or south of the selected feature.

8.1.1.6 Select Second Intersecting Feature (Use Case)

The operator will select an intersecting feature marking the end of the interval on the roadway containing the object. If no geolocation information is available for the intersecting feature marking the beginning of the interval but geolocation data is available for this feature, it will be used as the geolocation data for the object.

8.1.1.7 Specify Intersecting Feature Pair (Use Case)

When specifying an object location, a user may specify that an object is located between two intersecting features on a route by using the Proximity value of 'FROM_TO' or 'BETWEEN'. The user then will select a first and second intersecting feature, marking the beginning and end of the interval on the roadway containing the object. Each intersecting feature (start and end) may be an exit, route or milepost. The start and end intersecting features do not need to be of the same type. The same intersecting feature cannot be specified for both the first and second intersecting feature. When the intersecting feature pair is specified, the map view visible when editing an object location will pan and zoom to show both intersecting features if geolocation data is available for both feature.

8.1.1.8 Specify Lane Configuration (Use Case)

The user shall be able to specify the configuration of a roadway at a certain point, including the types of lanes included and the position of lanes relative to each other. The lane configuration editor will be initialized with the lane configuration and status provided during initialization of the editor if provided. Otherwise it will automatically pre-select the first lane configuration in the list of available lane configurations (which is sorted as specified in Select Lane Configuration).

8.1.1.9 Specify Object Location (Use Case)

A user may specify the location of an object with the aid of a system map. This process can involve selecting a state, county or region, primary route, and intersecting feature. The system will suggest a location description based on the selections the user makes. The user may override the suggested location description if desired, but will be warned when doing so and again before submitting the location form. During the process of setting an object location, a user may press a button to reset the form. Doing this will cause all previously entered location data to be lost and the location marker(s) will be cleared from the map.

9 Detailed Design – Map Between Features

9.1 Human-Machine Interface

9.1.1 "Between" as a Location Proximity

This section describes the map and form that are used when setting the location of a traffic event or device. The map and form have been changed to allow a user to specify a location that is between two intersecting features. This can be done when creating a new traffic event using the Event Launcher on the Home Page, when editing the location of an existing traffic event or when editing the location of a device.

9.1.1.1 New Location Proximity Values

Two new location proximity values have been added to the form used to specify a location: "BETWEEN" and "FROM-TO". These proximities are used to describe a location that is either between two intersecting features or that includes the segment of road from one intersecting feature to another. The only difference in the way these proximities function in the form and the map is the resulting location description. For example, a location with a proximity value of "BETWEEN" will have a location description similar to "I-270 BETWEEN EXIT 4 MONTROSE RD AND EXIT 5 FALLS RD" and a proximity value of "FROM-TO" will have a location description similar to "I-270 FROM EXIT 4 MONTROSE RD TO EXIT 5 FALLS RD".

9.1.1.2 Specifying a Second Intersecting Feature

The form and map default to the standard behavior of only allowing one intersecting feature if a proximity value other than "BETWEEN" or "FROM-TO" is selected as shown in Figure 9-1 below. The form displays an area to specify the intersecting feature within the "Intersecting Feature" tab.

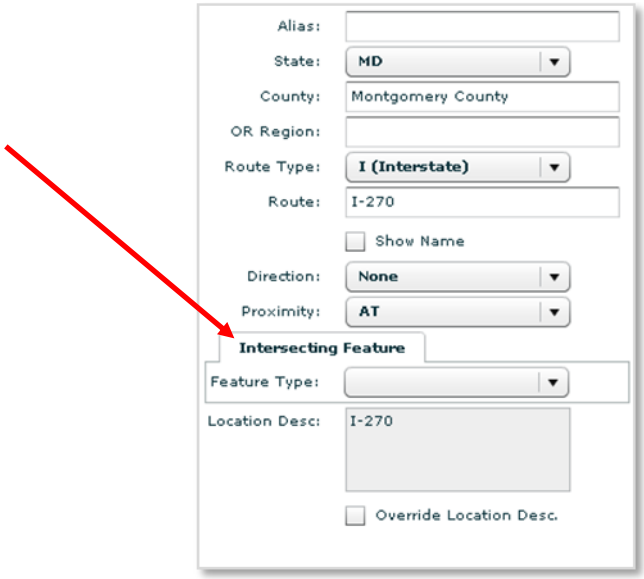


Figure 9-1 Location Form Showing Default Behavior

The new location proximity values ("BETWEEN" and "FROM-TO") have been added to the form and can be selected from the Proximity drop-down box as shown in Figure 9-2 below.

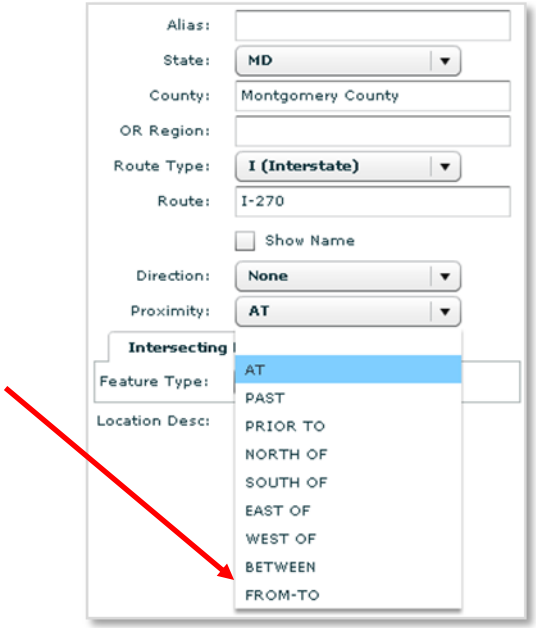


Figure 9-2 Location Form Showing New Values for Proximity

When a proximity value of either "BETWEEN" or "FROM-TO" is selected, the form displays an area to specify the second intersecting feature as shown in Figure 9-3 below. The "Intersecting Feature" tab becomes the "Feature 1" tab and an additional tab labeled "Feature 2" is visible and contains an area to specify the second intersecting feature.

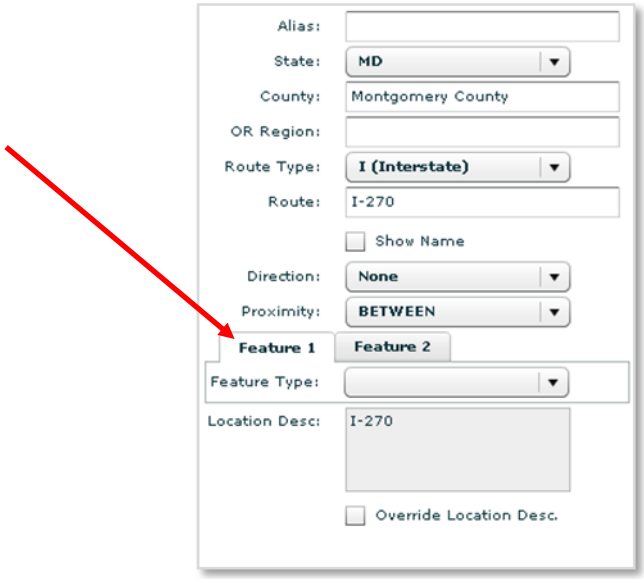


Figure 9-3 Location Form Showing Tabs for Feature 1 and Feature 2

Any combination of feature types can be used for the intersecting features as shown in Figure 9-4below. The only limitation is that the location for Feature 1 and Feature 2 cannot be identical (i.e. the same intersecting feature type and intersecting feature location cannot be specified for both Feature 1 and Feature 2).

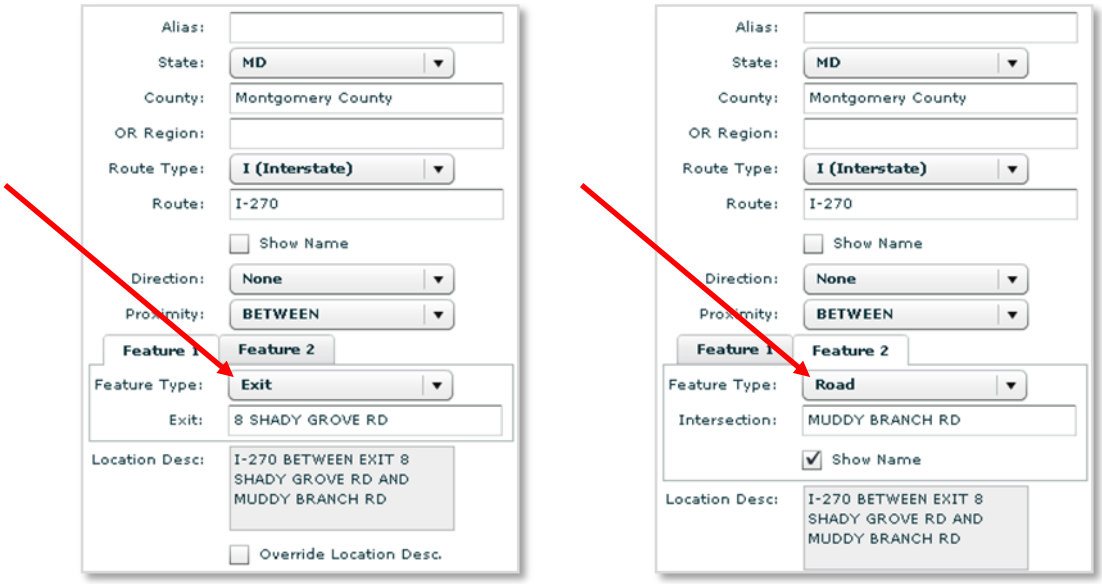


Figure 9-4 Location Form Showing Feature 1 as an Exit and Feature 2 as a Road

The beginning and ending intersecting features cannot be the same feature. For example, the same milepost cannot be used for Feature 1 and Feature 2 as shown in below.

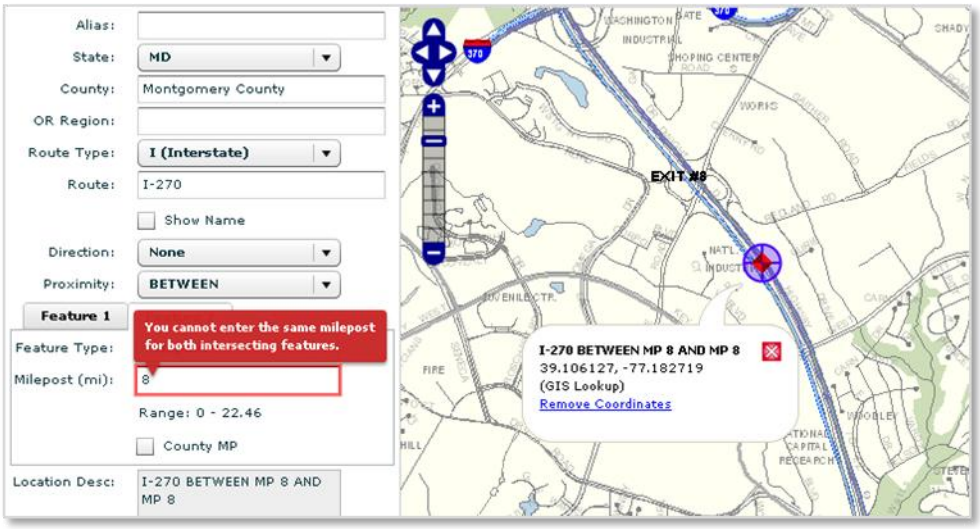


Figure 9-5 Location Form Showing Error Message Due to Identical Intersecting Features

9.1.1.3 Feature Markers on the Map

When a proximity value other than "BETWEEN" or "FROM-TO" is selected, the map defaults to the standard behavior of displaying the crosshairs marker at the object location (if a latitude/longitude is available) as shown in Figure 9-6 below.

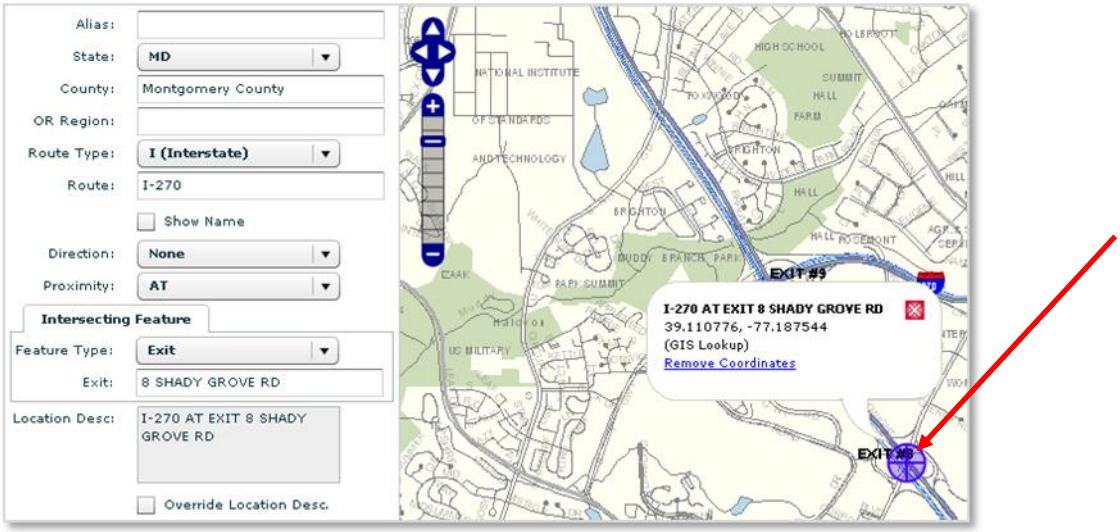


Figure 9-6 Location Form and Map Showing Default Behavior

When a proximity value of either "BETWEEN" or "FROM-TO" is selected, the map displays both the crosshairs marker at the object location (if a latitude/longitude is available) as well as the feature marker at the feature locations (if a latitude/longitude is available) as shown in Figure 9-7 below.

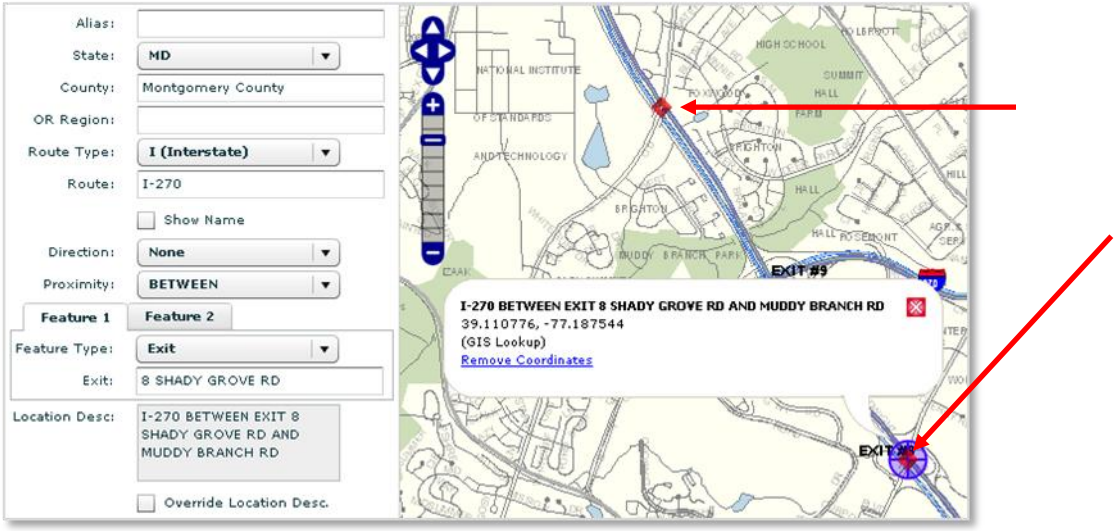


Figure 9-7 Location Form and Map Showing Cross-Hair Marker and Feature Markers

9.1.2 Set Traffic Event Duplicate Comparison Rules

This section describes changes to the Set Traffic Event Duplicate Comparison Rules feature. For R6, the server side logic used to detect duplicate traffic events for the purpose of generating Duplicate Event Alerts will now use latitude/longitude to compare an event's proximity to another event. Only events within this proximity (specified in miles) will be considered as duplicates. The Set Traffic Event Duplicate Comparison Rules page has been updated to allow entry of the proximity setting as shown in Figure 9-8 below.

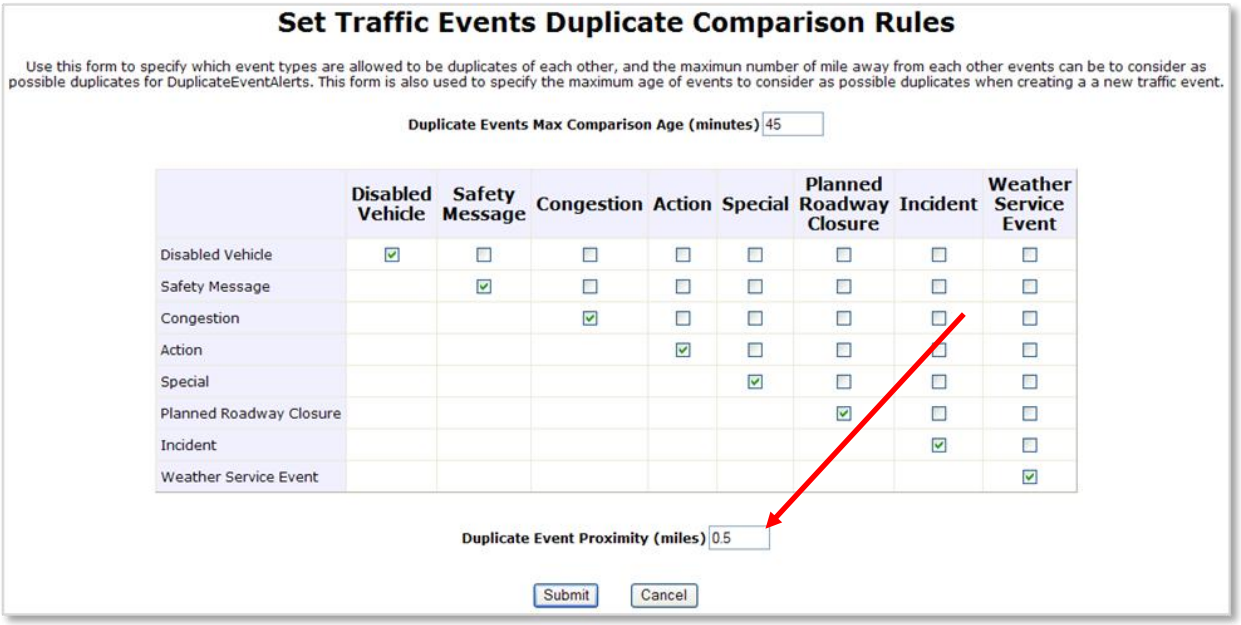


Figure 9-8 Set Traffic Event Duplicate Comparison Rules Page

The Duplicate Event Proximity should be entered in miles as a decimal as shown in Figure 9-9 below. For example, a user would enter 0.25 for 1/4 mile.

Figure 9-9 The Duplicate Event Proximity Value

9.2 CHART Common

9.2.1 Class Diagrams

9.2.1.1 Common2 (Class Diagram)

This class diagram shows classes used by many other modules within the CHART System. This diagram supplements the “Common” Class Diagram, showing additional classes which cannot fit on the original “Common” Class Diagram.

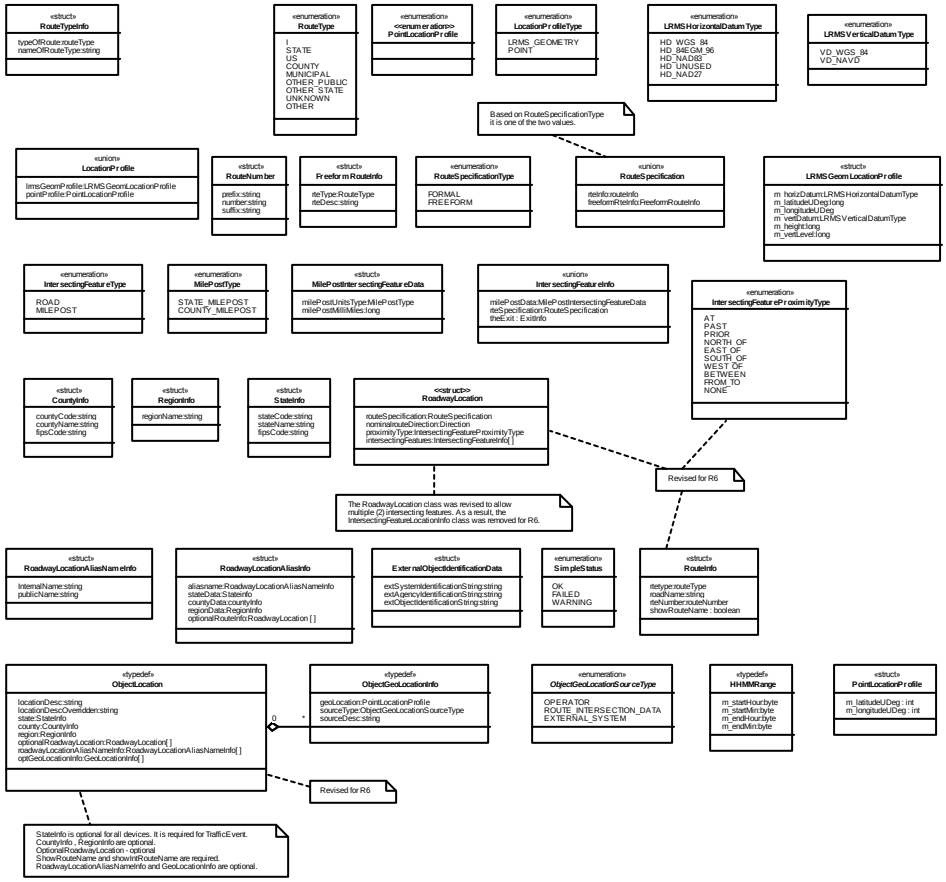


Figure 9-10 Common2 (Class Diagram)

9.2.1.1.1 <<enumeration>> PointLocationProfile (Class)

The PointLocationProfile struct represents a simple lat / lon point location.

9.2.1.1.2 <<struct>> RoadwayLocation (Class)

This structure has the information to define the roadway location of CHART objects like devices and traffic events.

9.2.1.1.3 CountyInfo (Class)

This structure contains information about a county.

9.2.1.1.4 ExternalObjectIdentificationData (Class)

This structure is used to hold data which identifies the external source of an external object which has been imported into CHART.

9.2.1.1.5 FreeformRouteInfo (Class)

Information specifying a route when only the route type and route description are known, such as may be the case when a route is entered by the user.

9.2.1.1.6 HHMMRange (Class)

This structure defines a time duration.

9.2.1.1.7 IntersectingFeatureInfo (Class)

This union provides auxiliary data for identifying an intersecting feature along a given roadway.

9.2.1.1.8 IntersectingFeatureProximityType (Class)

This enumeration represents a direction relative to an intersecting feature on a roadway for defining a location on the roadway. If no intersecting feature has been defined, the direction is NONE.

9.2.1.1.9 IntersectingFeatureType (Class)

The type of intersecting feature which is used to define a point along a given roadway.

9.2.1.1.10 LocationProfile (Class)

Data included in an LRMS geometry location profile.

9.2.1.1.11 LocationProfileType (Class)

Defines all supported location profiles for GeoLocatable objects in the system.

9.2.1.1.12 LRMSGeomLocationProfile (Class)

Data included in an LRMS geometry location profile.

9.2.1.1.13 LRMSHorizontalDatumType (Class)

This enum lists the values that can be used for horizontal datum for the LRMS (Location Referencing Message Specification) Geometry profile using TMDD proscribed values for loc_ext_horizontal_datum. (Reference TMDD Vol II Annex June 2004).

9.2.1.1.14 LRMSVerticalDatumType (Class)

This enum lists the values that can be used for vertical datum for the LRMS (Location Referencing Message Specification) Geometry profile using TMDD proscribed values for loc_ext_vertical_datum. (Reference TMDD Vol II Annex June 2004).

9.2.1.1.15 MilePostIntersectingFeatureData (Class)

This structure defines a milepost location along a given roadway, with a milepost measurement in terms of the specified milepost type.

9.2.1.1.16 MilePostType (Class)

This enumeration lists the type of milepost units.

9.2.1.1.17 ObjectGeoLocationInfo (Class)

This structure defines the geographical location of a CHART object.

9.2.1.1.18 ObjectGeoLocationSourceType (Class)

This structure defines the source of geolocation information of an object (a CHART entity).

9.2.1.1.19 ObjectLocation (Class)

This structure defines the location of CHART objects like devices and traffic events. StateInfo, CountyInfo, RegionInfo, RoadwayLocation, RoadwayLocationAliasNameInfo and GeoLocationInfo fields are optional.

9.2.1.1.20 PointLocationProfile (Class)

This struct represents a geographical point defined as a latitude / longitude pair. The latitude and longitude are defined in micro degrees.

9.2.1.1.21 RegionInfo (Class)

This structure contains information about a region.

9.2.1.1.22 RoadwayLocationAliasInfo (Class)

This structure contains the aliases for locations. For example, an alias can describe the Fort McHenry Tunnel where the alias would be FMT.

9.2.1.1.23 RoadwayLocationAliasNameInfo (Class)

This structure contains information on the two names of an alias for a roadway location.

9.2.1.1.24 RouteInfo (Class)

Information for specifying a route when the components of the route number information (and optionally the route name) are known. The showRouteName flag indicates whether to show the route name (instead of the route number) when displaying the description of the route.

9.2.1.1.25 Route Number (Class)

A route number, which may consist of an alphanumeric prefix, a number, and an alphanumeric suffix.

9.2.1.1.26 RouteSpecification (Class)

This union specifies a route using either a formal definition or a freeform text definition.

9.2.1.1.27 RouteSpecificationType (Class)

This enum indicates whether a route is specified using the formal definition (as is the case when the route number components are known), or whether the route was entered as freeform text.

9.2.1.1.28 RouteType (Class)

This enumeration is used to specify the classification of a road (interstate, MD, etc.)

9.2.1.1.29 RouteTypeInfo (Class)

This structure contains information about the classification type of a road.

9.2.1.1.30 SimpleStatus (Class)

This enum defines simple status values.

9.2.1.1.31 StateInfo (Class)

This structure contains information about a State.

9.3 GUI – Flex – Edit Location (chartlite/Flex/editlocation)

9.3.1 Class Diagrams

9.3.1.1 FlexLocationClasses (Class Diagram)

This diagram shows location-related classes.

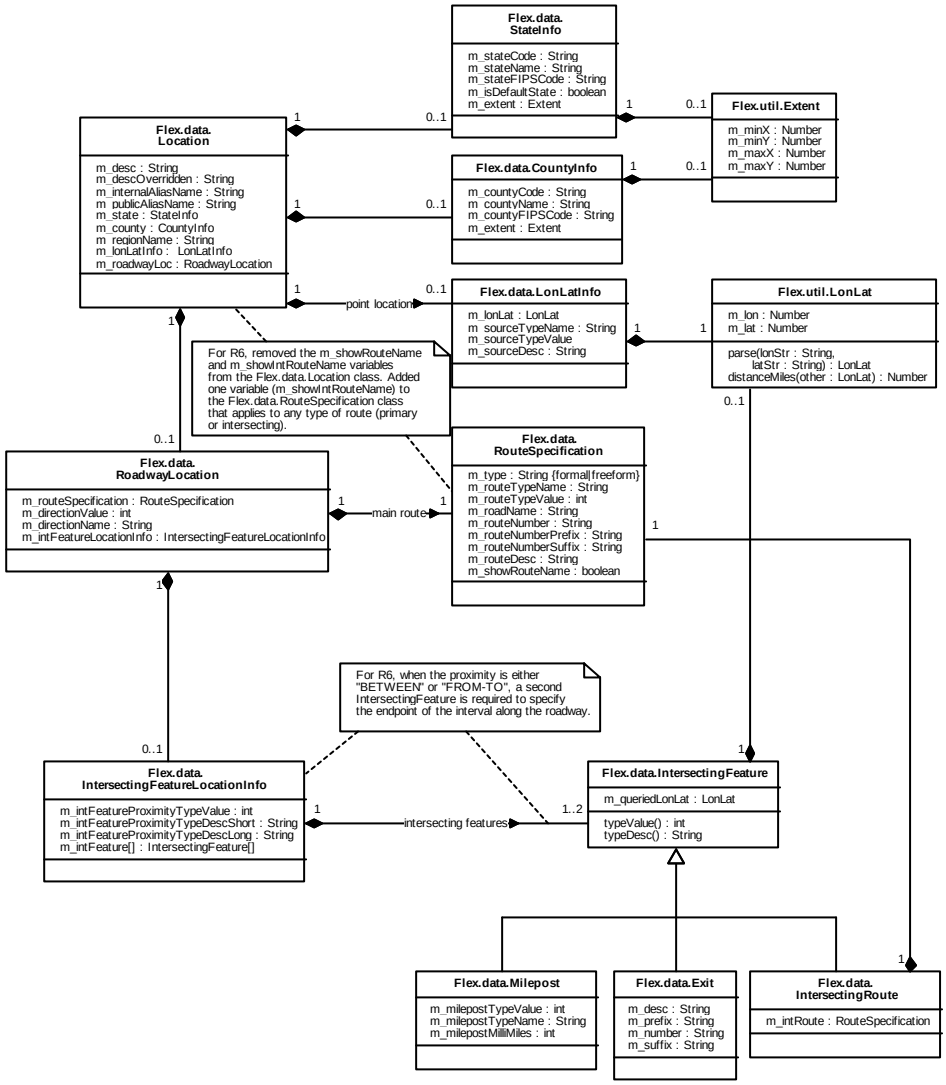


Figure 9-11 FlexLocationClasses (Class Diagram)

9.3.1.1.1 Flex.data. IntersectingFeatureLocationInfo (Class)

This class contains information about a location along a roadway relative to an intersecting feature (or two features, if the location is between two points on the roadway).

9.3.1.1.2 Flex.data. IntersectingRoute (Class)

This class represents an intersection of the main route with an intersecting route. The intersection may or may not be "at grade" - it can be a stop sign, stop light, or overpass.

9.3.1.1.3 Flex.data. Location (Class)

This class contains data describing a single location. The location may be a wide area (such as an entire state, county, or region) or it may be an entire route or some portion of a route

(one or both directions, a single point relative to a feature along the roadway, or an interval along the roadway). Most or all fields are optional, and are blank or null if not applicable.

9.3.1.1.4 Flex.data. RoadwayLocation (Class)

This class represents a location along a roadway / route. It may represent the entire route, one direction of the route, a specific point, or an interval along the roadway.

9.3.1.1.5 Flex.data. RouteSpecification (Class)

This class contains information identifying a route. This may include the route number information (and optionally the route name) or a free form route description. The showRouteName flag indicates whether to show the route name (instead of the route number) when displaying the description of the route.

9.3.1.1.6 Flex.data. StateInfo (Class)

This contains information about a U.S. state. The extent and isDefaultState may be set to null/false if not known or applicable.

9.3.1.1.7 Flex.data.CountyInfo (Class)

This class contains information about a county. The extent may be null if not known or applicable.

9.3.1.1.8 Flex.data.Exit (Class)

This class represents an exit along a route.

9.3.1.1.9 Flex.data.IntersectingFeature (Class)

This class represents an intersecting feature along a roadway. It may contain a longitude / latitude pair queried from the roadway database if the data is available; otherwise the coordinates will be null.

9.3.1.1.10 Flex.data.LonLatInfo (Class)

This class contains information about a longitude / latitude pair and the source of the coordinates.

9.3.1.1.11 Flex.data.Milepost (Class)

This class represents a state or county milepost.

9.3.1.1.12 Flex.util.Extent (Class)

Represents a bounding rectangle for X and Y.

9.3.1.1.13 Flex.util.LonLat (Class)

This class represents a longitude / latitude pair.

9.3.1.2 GUIFlexComponentsClasses (Class Diagram)

This diagram shows the Flex components used to specify an object location.

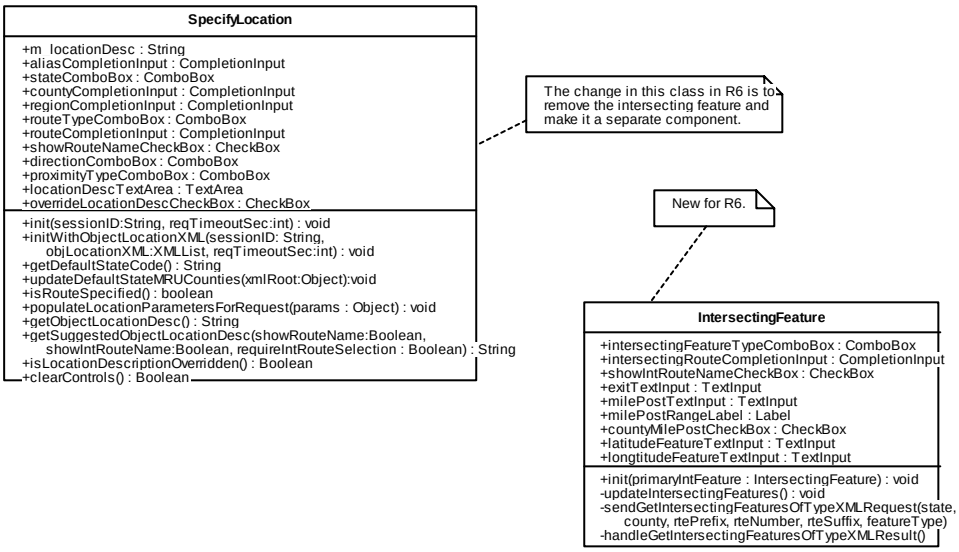


Figure 9-12 GUIFlexComponentsClasses (Class Diagram)

9.3.1.2.1 IntersectingFeature (Class)

This is a Flex control that allows a user to specify an intersecting feature for an object location. This control is used as a component in the SpecifyLocation control. A user first selects a feature type of Road, Exit, or Milepost. For an intersecting feature of type Road, a user can select from a list of intersecting roads (or enter a road name/number as freeform text). A user may also indicate if the road name or road number should be used for the description. For an intersecting feature of type Exit, a user can select from a list of exits (or enter an exit number as freeform text). For an intersecting feature of type Milepost, a user can enter a milepost number and indicate if it is a state milepost.

9.3.1.2.2 SpecifyLocation (Class)

This is the Flex control used to specify an object location. This form allows a user to specify a location using a combination of state, county, region, primary route, direction, proximity, and intersecting feature. For a proximity of BETWEEN or FROM-TO, two intersecting features may be selected. A user may also specify a location by selecting from a list of existing location aliases.

9.3.2 Sequence Diagrams

9.3.2.1 flex.shared.components:SpecifyLocation.updateIntFeatures (Sequence Diagram)

This diagram shows the processing performed when a user enters information about an object location in the Flex form with a proximity value of "BETWEEN" or "FROM-TO". The user first specifies the state, county, route type, primary route, and proximity in the Flex form (using the ComboBox and CompletionInput controls). By selecting a proximity value of "BETWEEN" or "FROM-TO", the user is able to define 2 intersecting features. When the first intersecting feature type is selected, the first list of intersecting features is updated based on the selected type. If the intersecting feature information is not available in the Flex form cache, a request is made to the GUI Servlet to get the intersecting features of the selected type and update the Flex form cache. When the second intersecting feature type is selected, the second list of intersecting features is updated based on the selected type. At this point the intersecting feature information should be available in the Flex form cache. If not, a request is made to the GUI Servlet to get the intersecting features of the selected type and update the cache.

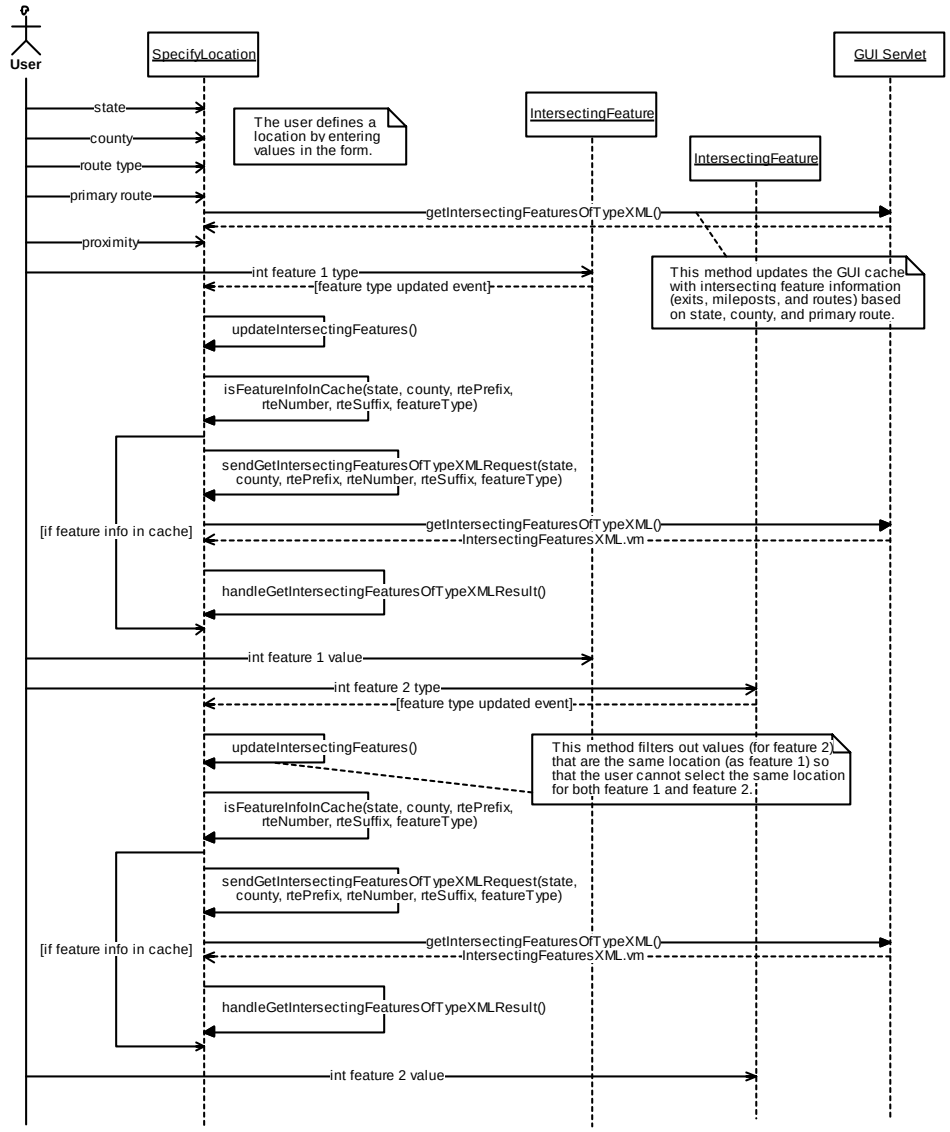


Figure 9-13 flex.shared.components:SpecifyLocation.updateIntFeatures (Sequence Diagram)

9.4 GUI – Javascript – Open Layers (chartlite/JavaScript/OpenLayers)

9.4.1 Class Diagrams

9.4.1.1 MapViewSpecificClasses (Class Diagram)

This diagram shows classes related to specific views of the map. This includes extensions of the CHARTLayers.CHARTMap class.

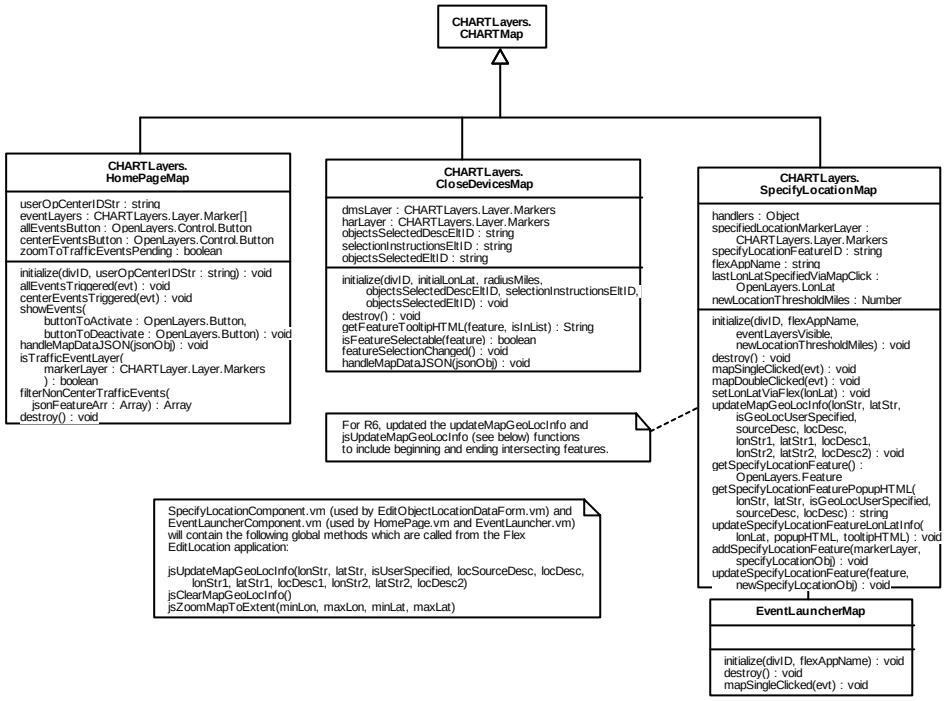


Figure 9-14 MapViewSpecificClasses (Class Diagram)

9.4.1.1.1 CHARTLayers. CHARTMap (Class)

This provides common functionality for all maps used in the CHART GUI, beyond the map functionality supported by OpenLayers.

9.4.1.1.2 CHARTLayers. CloseDevicesMap (Class)

This class represents the Close Devices Map that is shown on the Traffic Event Details page.

9.4.1.1.3 CHARTLayers. HomePageMap (Class)

This class represents the map on the Home Page.

9.4.1.1.4 CHARTLayers. SpecifyLocationMap (Class)

This class is used for specifying the location of an object. It supports a feature (and marker) to represent the location specified by the user. When an object location is specified with a proximity of "BETWEEN" or "FROM-TO", the map supports a marker at the beginning and ending features. The map interacts with a Flex application containing a Specify Location form in both directions, to allow the map to update the form and vice versa. This map will support panning and zooming.

9.4.1.1.5 EventLauncherMap (Class)

This map extends the SpecifyLocationMap to add functionality specific to the Event

Launcher.

9.4.2 Sequence Diagrams

9.4.2.1 SpecifyLocation:jsUpdateMapGeoLocInfo (Sequence Diagram)

This diagram shows how a location specified in the EditLocation Flex application is used to update the Specify Location Map or Event Launcher map. The EditLocation Flex application calls jsUpdateMapGeoLocInfo() via an external interface. (This JavaScript method will be defined in the SpecifyLocationComponent and EventLauncherComponent Velocity template files.) The SpecifyLocationMap object is called to update the location info. This builds HTML for the Specify Location feature's tooltip and popup, and calls updateSpecifyLocationFeatureLonLatInfo() which creates an object containing the data for the Specify Location feature, and calls the marker layer's updateFeatures() method to create / update the Specify Location feature and marker. If the “BETWEEN” or “FROM-TO” proximity value is selected, HTML will be built for the beginning and ending feature as well. An object will be created for both the beginning and ending feature and the call to the marker layer's updateFeatures() method will create / update the beginning and ending features and markers. Then, if the location is changed from the previous map click, the map is panned / zoomed (otherwise this is feedback from a map click, so the map is not panned / zoomed). The feature's popup is hidden or shown depending on whether the popup HTML is empty.

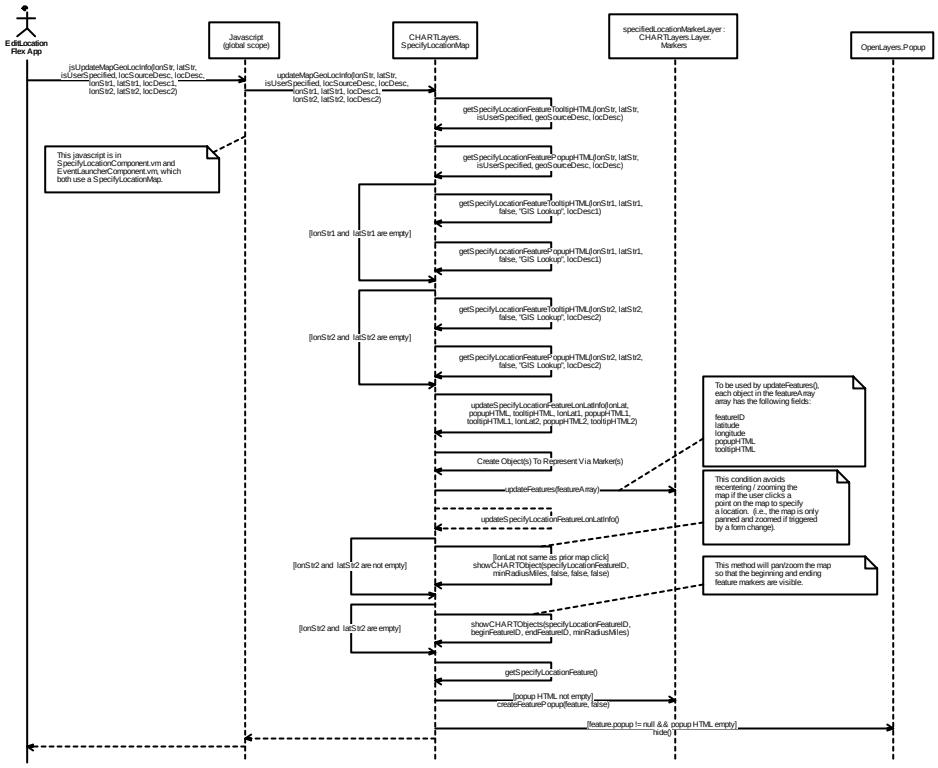


Figure 9-15 SpecifyLocation:jsUpdateMapGeoLocInfo (Sequence Diagram)

9.5 Utility Package

9.5.1 Class Diagrams

9.5.1.1 UtilityClasses2 (Class Diagram)

This Class Diagram shows various utility classes related to log entries that are used by GUI and servers.

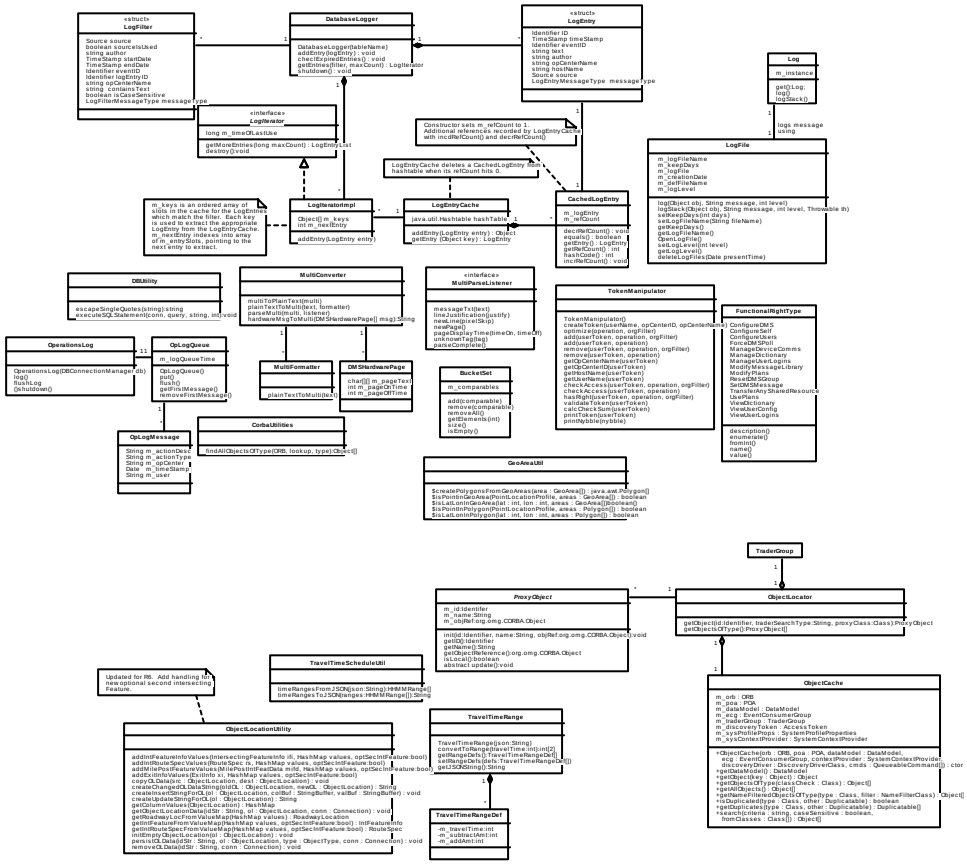


Figure 9-16 UtilityClasses2 (Class Diagram)

9.5.1.1.1 BucketSet (Class)

This class is designed to contain a collection of comparable objects. All of the objects added to this collection must be of the same concrete type. Each element in the collection has an associated counter which tracks how many times this element has been added. It is then possible to get only the elements which have been added to the collection n times where n is a positive integer value. This class is very useful for creating GUI menu's for multiple objects as it allows all objects to insert their menu items and then allows the user

to get only those items which all objects inserted.

9.5.1.1.2 CachedLogEntry (Class)

This class represents a reference-counting object stored in a memory-efficient LogEntryCache. The object of this class encapsulates the stored log entry and adds a reference count.

9.5.1.1.3 CorbaUtilities (Class)

This class is a collection of static CORBA utility methods that can be used by both server and GUI for CORBA Trader service transactions.

9.5.1.1.4 DatabaseLogger (Class)

This class represents a generic database logger which can be used to log and retrieve information from the database. This class also provides a mechanism for the user to filter and retrieve logs that meet a specific criteria.

9.5.1.1.5 DBUtility (Class)

This class contains methods that allow interaction with the database.

9.5.1.1.6 DMSHardwarePage (Class)

This class holds data that specifies the layout of one page of a DMS message on the actual DMS hardware. A two dimensional array that is the same size as the sign's display (rows and columns) specifies the character displayed in each cell, including blank if the cell has no character. This format maps well to the way DMS protocols return the current message being displayed in a status query. This class can then be passed to a MultiConverter object to convert the message into MULTI format.

9.5.1.1.7 FunctionalRightType (Class)

This class acts as an enumeration that lists the types of functional rights possible in the CHART2 system. It contains a static member for each possible functional right.

9.5.1.1.8 GeoAreaUtil (Class)

This class contains static methods used for searching GeoAreas for specific Lat/Lons. The actual search of a GeoArea is done by converting a chart GeoArea to a java.awt.Polygon then using that object's contains (lat,lon) method to make the determination. Several helper methods front the Polygon based methods for flexibility.

9.5.1.1.9 Log (Class)

Singleton log object to allow applications to easily create and utilize a LogFile object for system trace messages.

9.5.1.1.10 LogEntry (Class)

This class represents a typical log entry that is stored in the database. This can be a general Communications Log entry or it can be a historical entry for a Traffic Event. Some Traffic Event actions (opening, closing, etc.) are logged in the Communications Log as well as in the history of the specific Traffic Event.

9.5.1.1.11 LogEntryCache (Class)

The LogEntryCache caches log entries returned from a database query which are in excess of the requestor-specified maximum number of entries to return at one time. The LogIterator stores references to the LogEntry objects thus cached, and requests additional objects as needed. The LogEntryCache uses reference counting to prevent storing duplicate copies of LogEntry objects, and it deletes LogEntry objects when they are no longer needed.

9.5.1.1.12 LogFile (Class)

This class creates a flat file for writing system trace log messages and purges them at user specified interval. The log files created by this class are used for system debugging and maintenance only and are not to be confused with the system operations log which is modeled by the OperationsLog class.

9.5.1.1.13 LogFilter (Class)

This class is used to specify the criteria to be used when getting entries from the Communications Log. The caller would create an object of this type specifying the criteria that each log entry must match in order to be returned.

9.5.1.1.14 LogIterator (Class)

This class represents an iterator to iterate through a collection of log entries. If a retrieval request results in more data than is reasonable to transmit all at once, one clump of entries is returned at first, together with a LogIterator from which additional data can be requested, repeatedly, until all entries are returned or the user cancels the operation.

9.5.1.1.15 LogIteratorImpl (Class)

The LogIteratorImpl implements the LogIterator interface; that is, it does the actual work which clients can request via the LogIterator interface. The LogIteratorImpl stores data relating to cached LogEvents for a single retrieval request, and implements the client request to get additional clumps of data pertaining to that request.

9.5.1.1.16 MultiConverter (Class)

This class provides methods which perform conversions between the DMS MULTI mark-up language and plain text. It also provides a method which will parse a MULTI message and inform a MultiParseListener of elements found in the message.

9.5.1.1.17 MultiFormatter (Class)

This interface must be implemented by classes which convert plain text DMS messages to MULTI formatted messages.

9.5.1.1.18 MultiParseListener (Class)

A MultiParseListener works in conjunction with the MultiConverter to allow an implementing class to be notified as parsing of a MULTI message occurs. An exemplary use of a MultiParseListener would be the MessageView window which will need to have the MULTI message parsed in order to display it as a pixmap.

9.5.1.1.19 ObjectCache (Class)

The ObjectCache is a wrapper for the DataModel. It provides access to DataModel methods to find objects in the data model, delegating those methods to the DataModel itself. It also provides additional methods of finding name filtered objects and discovering "duplicate" objects (as defined by an isDuplicateOf() method of the Duplicatable interface).

9.5.1.1.20 ObjectLocationUtility (Class)

This class contains utility functionality for dealing with ObjectLocation objects.

9.5.1.1.21 ObjectLocator (Class)

This class is used to provide access to proxy objects stored in the CHART object cache (which have been discovered by the DiscoveryDriver tasks).

9.5.1.1.22 OperationsLog (Class)

This class provides the functionality to add a log entry to the Chart II operations log. At the time of instantiation of this class, it creates a queue for log entries. When a user of this class provides a message to be logged, it creates a time-stamped OpLogMessage object and adds this object to the OpLogQueue. Once queued, the messages are written to the database by the queue driver thread in the order they were queued.

9.5.1.1.23 OpLogMessage (Class)

This class holds data for a message to be stored in the system's Operations Log.

9.5.1.1.24 OpLogQueue (Class)

This class is a queue for messages that are to be put into the system's Operations Log. Messages added to the queue can be removed in FIFO order.

9.5.1.1.25 ProxyObject (Class)

This class is a base class for many types of proxy objects store in the CHART object cache (which have been discovered by the DiscoveryDriver tasks), used to provide a standard set of access methods for the proxy objects.

9.5.1.1.26 TokenManipulator (Class)

This class contains all functionality required for user rights in the system. It is the only code in the system which knows how to create, modify and check a user's functional rights. It encapsulates the contents of an octet sequence which will be passed to every secure method. Secure methods should call the checkAccess method to validate the user. Client processes should use the check access method to verify access and optimize to reduce the size of the sequence to only those rights which are necessary to invoke the secure method. The token contains the following information. Token version, Token ID, Token Time Stamp, Username, Op Center ID, Op Center IOR, functional rights

9.5.1.1.27 TraderGroup (Class)

This class provides a facade for trader lookups that allows application level code to be unaware of the number of CORBA trading services that the application is using or the details of the linkage between those services.

9.5.1.1.28 TravelTimeRange (Class)

This class is a utility that can convert a travel time into a travel time range. It is constructed using the travel time range definitions as specified by a JSON string stored in the system profile. The convertToRange method can then be called get the low and high range values for a specific travel time.

9.5.1.1.29 TravelTimeRangeDef (Class)

This class holds data for a travel time range definition. It has methods that can convert this object to a JSON object for persistence in the system profile, and to allow its data to be loaded from a JSON object when depersisting from the system profile.

9.5.1.1.30 TravelTimeScheduleUtil (Class)

This class provides utility methods related to travel time schedules.

9.5.1.2 LocationRelatedProxyClasses (Class Diagram)

This class diagram shows traffic related proxy objects impacted for R6. IDL modification will cause some proxy objects to be removed, added or updated.

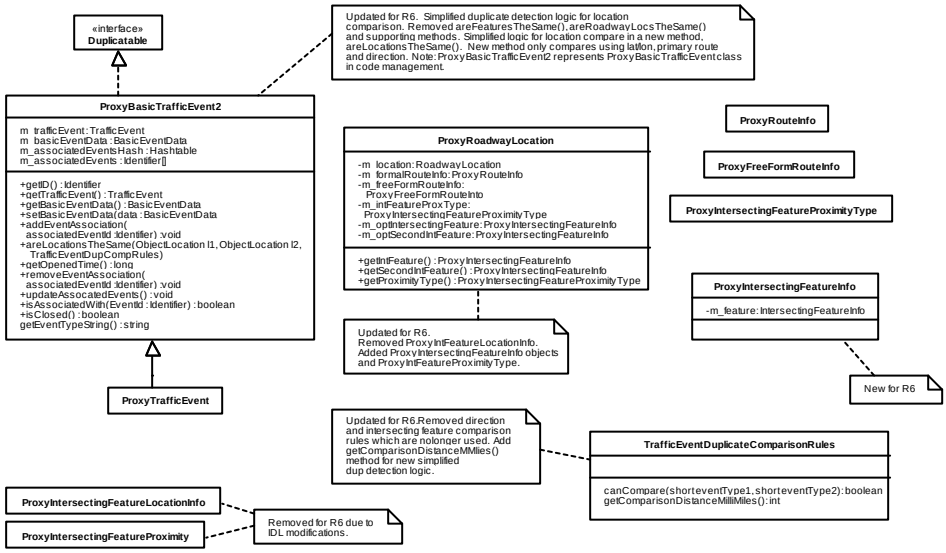


Figure 9-17 LocationRelatedProxyClasses (Class Diagram)

9.5.1.2.1 Duplicatable (Class)

This java interface is implemented by classes which have sense of being "duplicated" within the CHART system. This allows the ObjectCache to search for duplicates of any Duplicatable object. This is different from "equals()" or "compareTo()". To cite two examples: Alerts within CHART are duplicates if they refer to the same objects within CHART (but do not have the same Alert ID, which is more closely associated with "equals()"). Traffic Events within CHART are duplicates if they have the same location (but do not have the same Traffic Event ID).

9.5.1.2.2 ProxyBasicTrafficEvent2 (Class)

This class is used as a proxy for traffic events existing in all traffic event services (including the local service). The proxy traffic events cached are not complete copies of the traffic events, because the full range of data is not needed. The ProxyBasicTrafficEvent data consists of BasicEventData and associated events only (this is why the names of these objects contain the word "Basic", e.g., DiscoverBasicTrafficEventClassesCommand. These proxy traffic events allow every traffic event service in the system to have some knowledge of every traffic event in the entire system, for the purpose of detecting duplicate traffic events.

9.5.1.2.3 ProxyFreeFormRouteInfo (Class)

This class is used as a proxy for FreeFormRouteInfo IDL generated classes.

9.5.1.2.4 ProxyIntersectingFeatureInfo (Class)

This class is used as a proxy for IntersectingFeatureInfo IDL generated classes.

9.5.1.2.5 ProxyIntersectingFeatureLocationInfo (Class)

Removed for R6. Formerly used to a Proxy for IntersectingFeaturelocationInfo this no longer exists.

9.5.1.2.6 ProxyIntersectingFeatureProximity (Class)

Removed for R6. Formerly used to a Proxy for IntersectingFeatureProximity this no longer exists.

9.5.1.2.7 ProxyIntersectingFeatureProximityType (Class)

This class is used as a proxy for IntersectingFeatureProxyType IDL generated class.

9.5.1.2.8 ProxyRoadwayLocation (Class)

This class is used as a proxy for the RoadwayLocation IDL generated class. It wraps location information and provides convenience methods used to simplify coding.

9.5.1.2.9 ProxyRouteInfo (Class)

This class is used as a proxy for RouteInfo IDL generated classes.

9.5.1.2.10 ProxyTrafficEvent (Class)

This class is used as a proxy for traffic events existing in all traffic event services (including the local service). The ProxyTrafficEvent is derived from the ProxyBasicTrafficEvent and it is a more complete copy of the traffic events used only when the full range of data is needed. The ProxyTrafficEvent data consists of LaneConfiguration, RPIs, Participants, etc.

9.5.1.2.11 TrafficEventDuplicateComparisonRules (Class)

The TrafficEventDuplicateComparisonRules class is a helper class that provides access to system profile properties that represent rules used when comparing Traffic events for the purpose of detecting duplicate events and generating DuplciateEventAlerts.

9.5.2 Sequence Diagrams

9.5.2.1 ProxyBasicTrafficEvent2:areLocationsTheSame (Sequence Diagram)

The ProxyBasicTrafficevent.areLocationsTheSame() private method is called by the isDuplicateOf() method which is called during duplicate event detection processing. This method does a simple compare of two locations and returns true if the locations are considered the same, false otherwise. The primary routes and directions are compared. Then the lat/longs of the two events are compared to determine if they are within a configurable distance of one another. If not, the location are considered different. If either location is missing lat/lon, primary route, or direction the locations are not considered the same.

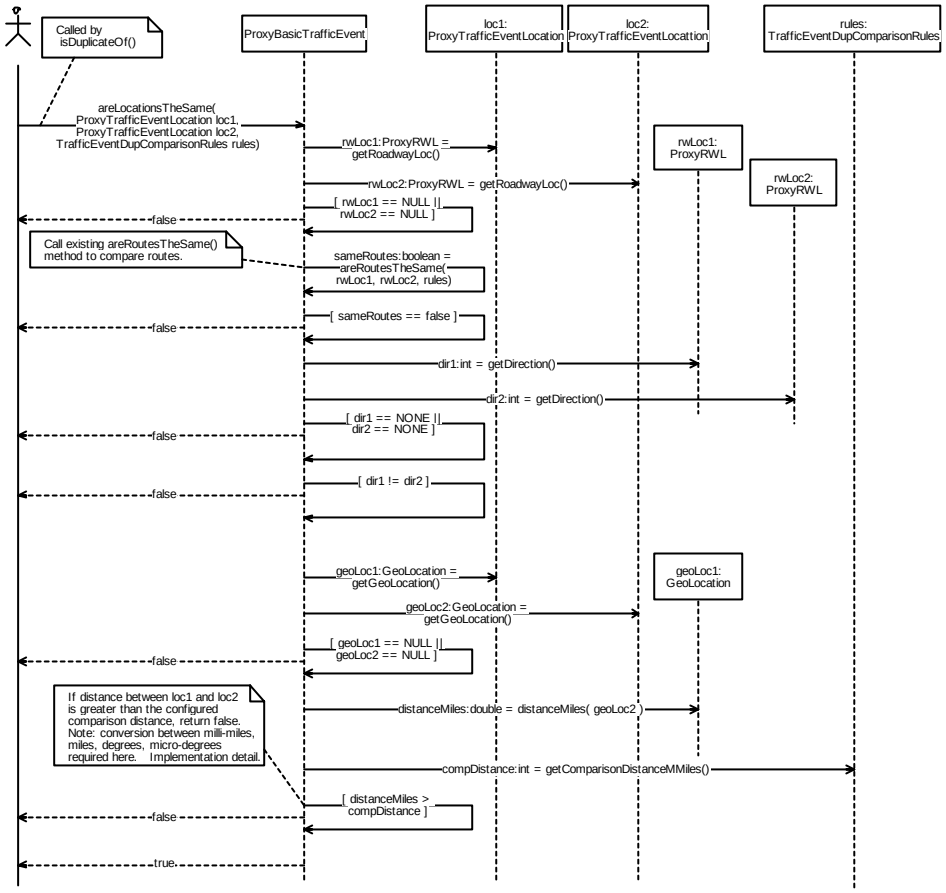


Figure 9-18 ProxyBasicTrafficEvent2:areLocationsTheSame (Sequence Diagram)

10 Use Cases – External TSS

The use case diagrams depict new functionality for the CHART R6 External TSS feature and also identify existing features that will be enhanced. The use case diagrams for this feature exist in the Tau design tool in the Release6 area. The sections below indicate the title of the use case diagrams that apply to this feature.

10.1 CHART

10.1.1 Provide Data to External Systems (Use Case Diagram)

This diagram shows uses of the system related to providing data to external system. R6 CHART provides Traffic Event, DMS, HAR, SHAZAM, TSS, and CCTV to external systems. In addition, R6 provides external client management for the authentication of external users.

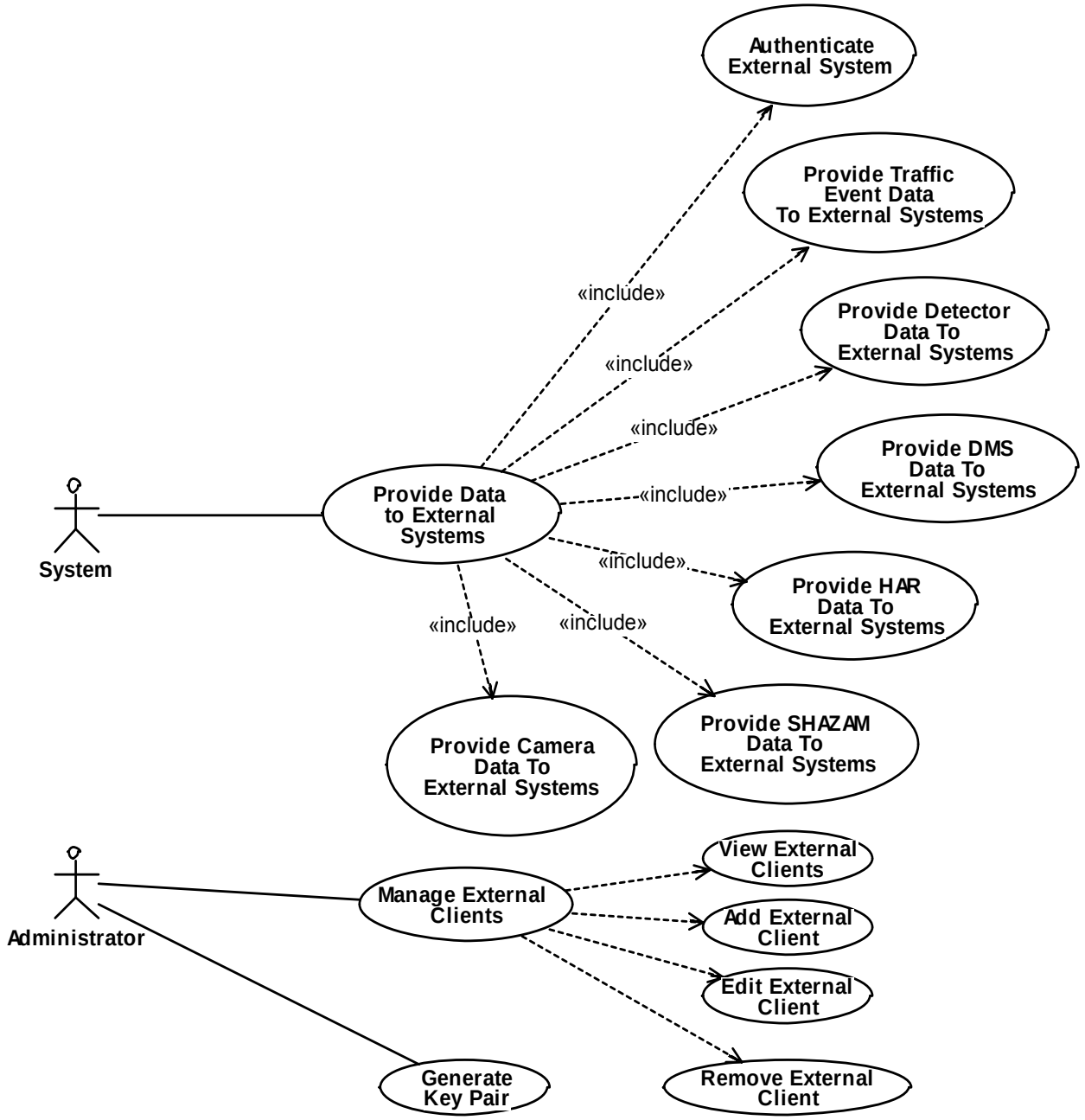


Figure 10-1 R6 ProvideDataToExternalSystems (Use Case Diagram)

10.1.1.1 Add External Client (Use Case)

The system shall allow an administrator to add an external client to the CHART system. When doing so, the administrator will specify data pertaining to the client including a client ID to be used by the external system and a public key used to validate data signed by the external system. The administrator will specify whether the external system is a supplier and/or consumer of CHART data, and if it is a consumer, one or more CHART Roles whose user rights will determine the data accessible to the external system. The

administrator will also be able to specify a name and description of the external system, contact information for a person responsible for managing the external system.

10.1.1.2 Authenticate External System (Use Case)

The system shall authenticate external system connections to validate that they are authorized to connect to CHART and to enforce rights regarding the data the external system is permitted to access. Each external system owner will be provided a private key from a public/private key pair generated within the CHART system. Each request from an external system will include the system's CHART client ID (as configured within CHART) and a digital signature of the request data, created using the private key provided by the CHART administrator. The CHART system will validate each request signature using the client's public key.

10.1.1.3 Edit External Client (Use Case)

The system shall allow an administrator to edit the data associated with an external client, as described in the Add External Client use case.

10.1.1.4 Generate Key Pair (Use Case)

The system shall allow an administrator to generate a public/private key pair for use in controlling access to the CHART external interface. The private key is to be given (offline) to the owner of the external system wishing to gain access to CHART data. The public key is to be used by the administrator when adding the external client to the CHART system that corresponds to the external system that wishes to retrieve data from CHART.

10.1.1.5 Manage External Clients (Use Case)

The system shall allow an administrator to manage the external clients that are permitted to retrieve data from CHART via its external system interface.

10.1.1.6 Provide Camera Data to External Systems (Use Case)

The system shall provide Camera data to external systems using the TMDD standard format, with CHART extensions as needed. External systems can obtain an inventory and status of all CHART Camera devices, or the ones that have changed in a certain look back time period. They can obtain updates to the Camera devices (including the current status) periodically with on-demand request or by subscribing to receive updates at a specified web service URL.

10.1.1.7 Provide Data to External Systems (Use Case)

The system shall provide access to external systems via a web service to allow them to receive data that the CHART system makes available to third parties. One or more Roles assigned to each external client will be used to determine the data the client will be permitted to access. All requests made by external systems shall be validated against

published XSD. CHART will return a response XML document for each request. The XML returned will contain an error code and error text for invalid requests, and will return the requested data for valid, authorized requests. The response XML shall be formatted as specified in published XSD.

10.1.1.8 Provide Detector Data to External Systems (Use Case)

The system shall provide detector data to external systems. The system shall enforce granular, organization based user rights to allow the level of detail provided for a detector to be controlled. Two user rights will be used to determine if a detector's detailed volume, speed, and occupancy (VSO) data is exported, only a speed range, or no VSO data. When VSO data is provided for a detector, it will include the data for zone groups and for each zone within the group. The detector data will be provided using the TMDD standard, with CHART extensions as needed. External systems can obtain an inventory and status of all CHART system detectors, or the ones that have changed in a certain look back time period. They can obtain updates to the detector data (including the status) periodically with on-demand request or by subscribing to receive updates at a specified web service URL.

10.1.1.9 Provide DMS Data to External Systems (Use Case)

The system shall allow external systems to receive data pertaining to DMSs using the TMDD standard, with CHART extensions to the standard as needed. External systems can obtain an inventory and status of all CHART DMS devices, or the ones that have changed in a certain look back time period. They can obtain updates to the DMS devices (including the current status) periodically with on-demand request or by subscribing to receive updates at a specified web service URL.

10.1.1.10 Provide HAR Data to External Systems (Use Case)

The system shall allow external systems to receive data pertaining to HARs using the TMDD standard, with CHART extensions to the standard as needed. External systems can obtain an inventory and status of all CHART HAR devices, or the ones that have changed in a certain look back time period. They can obtain updates to the HAR devices (including the current status) periodically with on-demand request or by subscribing to receive updates at a specified web service URL.

10.1.1.11 Provide SHAZAM Data to External Systems (Use Case)

The system shall provide SHAZAM (beacons) data to external systems using the TMDD standard format, with CHART extensions as needed. External systems can obtain an inventory and status of all CHART SHAZAM devices, or the ones that have changed in a certain look back time period. They can obtain updates to the SHAZAM devices (including the current status) periodically with on-demand request or by subscribing to receive updates at a specified web service URL.

10.1.1.12 Provide Traffic Event Data to External Systems (Use Case)

The system shall allow external systems to receive traffic event data from CHART. The CHART system shall enforce granular, organization based user rights to determine the level of detail that may be seen by each event. User rights will control whether or not the incident type of "fatality" is exported within the event name and the incident type field. A cleansed version of the incident type and event name that substitutes personal injury for fatality will be exported to external clients that don't have the proper user right. A separate user right determines whether or not event history for an event is exported. The traffic event data is exported using the SAE ATIS J2354 standard format with CHART extensions as needed. External systems can obtain a list of traffic events (an inventory) that includes either all events in the system or the ones that have changed in a certain look back time period. They can receive updates by polling to receive a new inventory periodically or by subscribing to receive updates at a specified web service URL.

10.1.1.13 Remove External Client (Use Case)

The system shall allow an administrator to remove an external client from the system, effectively preventing them from accessing CHART's external interface to retrieve data. The system will prompt the user for confirmation before removing the client.

10.1.1.14 View External Clients (Use Case)

The system shall allow an administrator to view the external clients allowed to retrieve CHART data via its external interface.

10.2 Mapping

10.2.1 CHART Intranet Map User Login (Use Case Diagram)

This diagram shows uses of the system related to logging into the Intranet map. In CHART R6 a user will login to the Intranet map to get specific views of TSS data. Otherwise a default view of TSS data is provided.

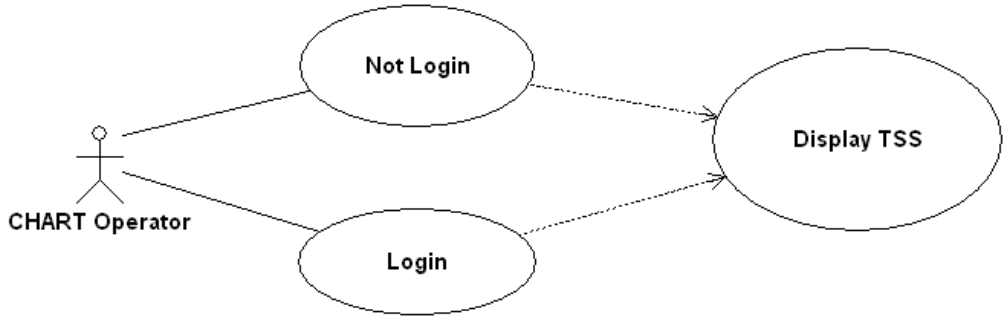


Figure 10-2 Intranet Map User Login (Use Case Diagram)

10.2.1.1 Not Login (Use Case)

By default, a user does not have to login to the Intranet Map. By not logging in they receive the rights associated with a default user set up by the CHART administrator. It is expected this user will have rights to view everything except objects with restrictions. For the R6 delivery this likely means non-logged in users will not be able to view detailed speeds for NAVTEQ detectors.

10.2.1.2 Login (Use Case)

The system shall allow an operator to authenticate their username and password to obtain their full set of rights. To do this, the operator clicks on the "Login to CHART" option under the "Display" drop down menu. A login screen prompts the operator to provide their user name and password. After submitting the login information, the system determines the user's rights via the web-based user management service. If the user is authenticated, the system returns to the default home screen, otherwise, a warning message is prompted.

10.2.1.3 Display TSS (Use Case)

Detectors (Traffic Sensor Systems) display on the Intranet Map using arrows to depict the underlying direction of roadway travel. When a user mouse's over a detector's arrow, a tooltip displays detailed information about that detector including detector location, the last report date and time, roadway direction, and owning organization. The tooltip also displays speed indications depending on the user's rights to view an organization's detectors:

Summary speed values (speed range): if the user does not have the ViewVSODetailedRight

Detailed speed values (actual speed): if the user has the ViewVSODetailedRight

There is no Intranet Map support for the ViewVSOSummaryRight because the customer does not envision a case when a detector would be displayed without at least a speed range.

If no user rights can be obtained due to an internal system problem, all users will at least be given detailed rights to view all TSSs owned by the SHA organization as these are not expected to ever be restricted for any user.

11 Detailed Design – External TSS

11.1 Human-Machine Interface

11.1.1 CHART Intranet Mapping GUI

The CHART Intranet Mapping GUI includes changes to the Menu.

11.1.1.1 Login to CHART

To login to CHART, an operator mouse's over to the "Display" dropdown menu and selects the "Login to CHART" option.

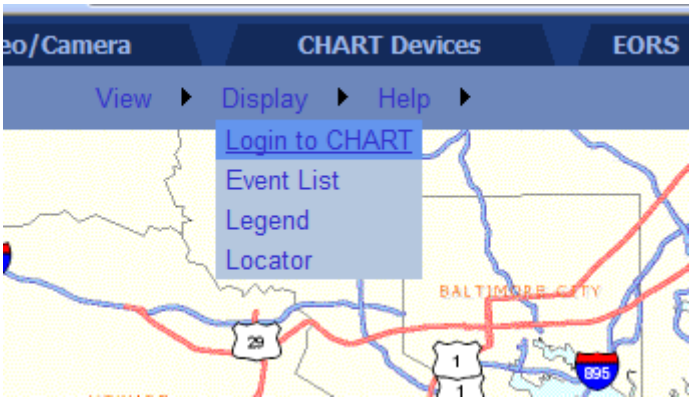


Figure 11-1 Login to CHART

The login screen is prompted which allows an operator to enter his or her CHART login. If the login is successful, the system remembers the user's session and page and then redirects them back to the Intranet Mapping application default screen. An error message is displayed if login is unsuccessful. If operator decided not to login, he or she can click on the "Go to Intranet Mapping without Logging into CHART" link and redirect back to the Intranet Mapping application default screen.

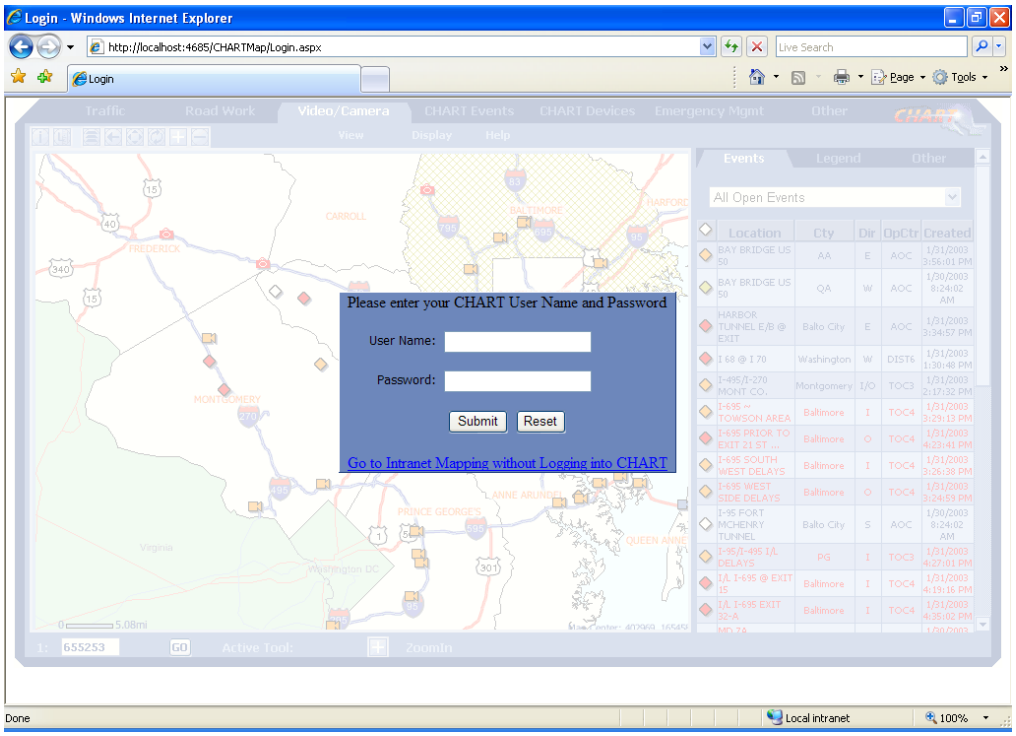


Figure 11-2 Login Screen

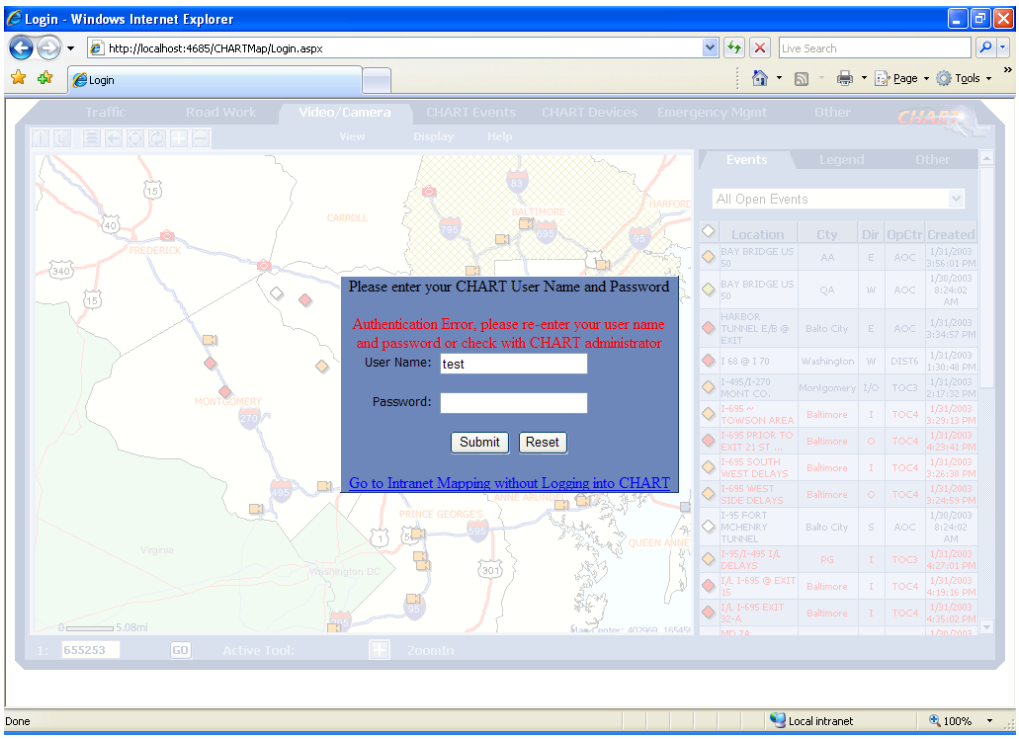


Figure 11-3 Login Screen Error

11.1.1.2 Display External TSS

Two new categories "Speed Ranges" and "Organizations" have been added under the "Road Speed Sensor" in the Legend. "Organizations" is initially collapsed to reduce the space in the legend.

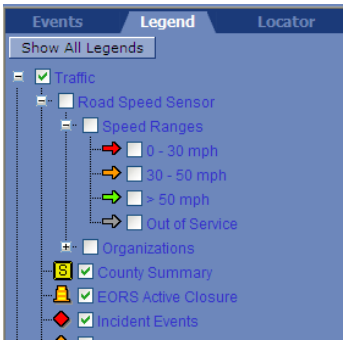


Figure 11-4 New Road Speed Sensor Legend

When the Road Speed Sensor category is selected, all Speed Ranges and the SHA Organization are automatically selected by default. A user can explicitly turn on the individual organizations by selecting the check box next to the name or, alternatively, they can select all organizations by checking the "Organization" check box.

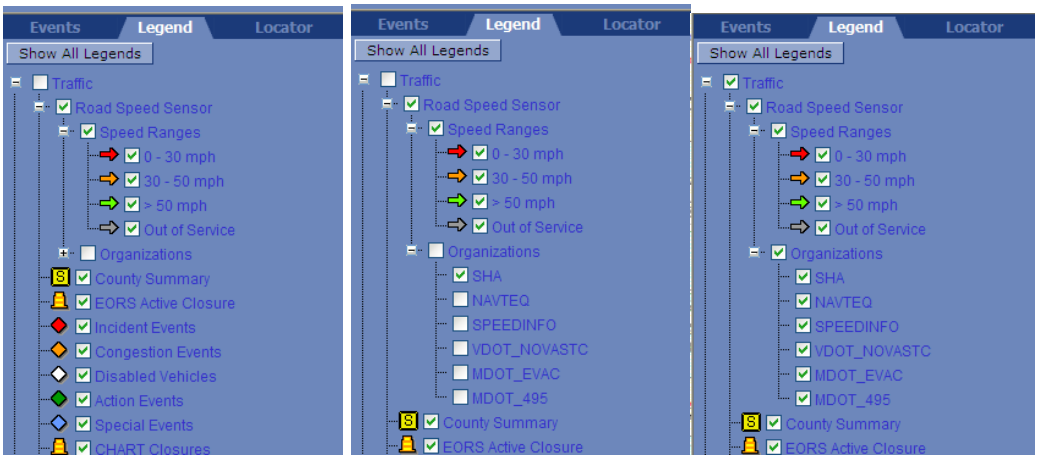


Figure 11-5 Turn on Road Speed Sensor layer

The user can optionally filter out the display of the speed sensors on the map by the owning organization or by the speed range by checking/unchecking the owning organization or the speed range in the legend.

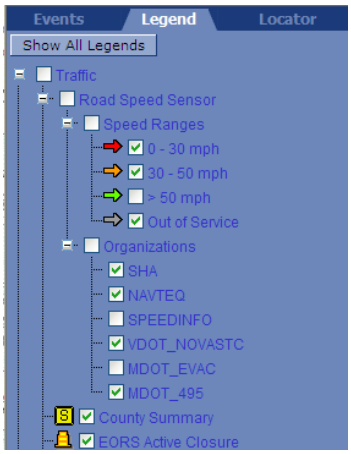


Figure 11-6 Filter TSS Display by Owning Organization and Speed Range

11.2 Mapping

11.2.1 Class Diagrams

11.2.1.1 Configuration - Intranet Map (Class Diagram)

This is a base class to capture CHART system configurations

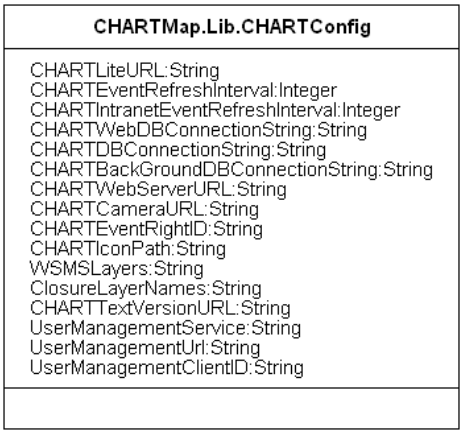


Figure 11-7 Intranet Map Configuration (Class Diagram)

11.2.1.1.1 CHARTMap.Lib.CHARTConfig (Class)

This is a base class for capturing CHART system configurations.

11.2.1.2 User Validation - Intranet Map (Class Diagram)

This is a base class to handle user authentication. This class contains methods to validate user against the CHART User Management Service and it also contains methods to handle the response.

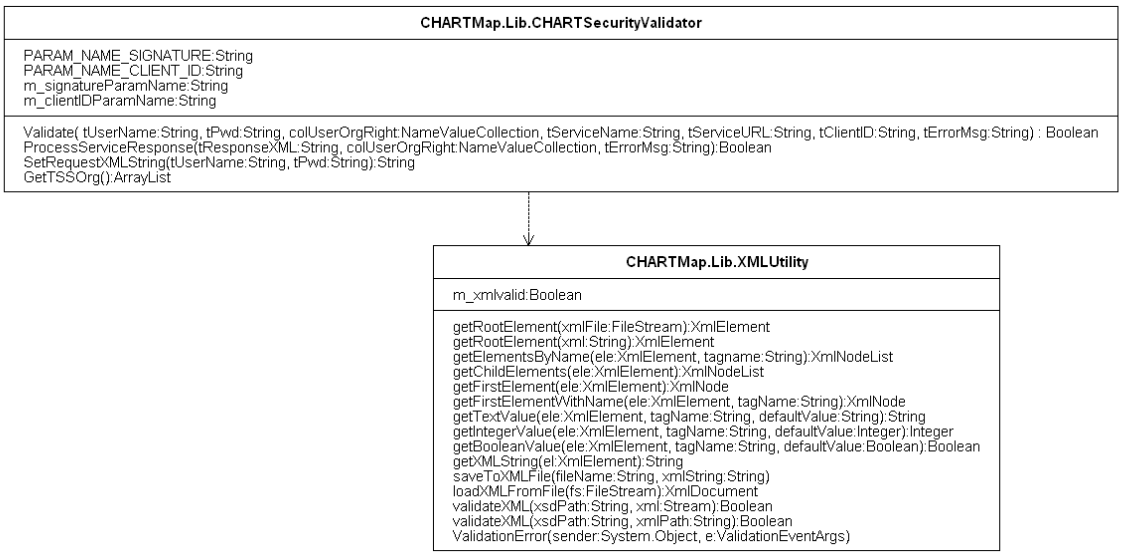


Figure 11-8 Intranet Map User Validation (Class Diagram)

11.2.1.2.1 CHARTMap.Lib.CHARTSecurityValidator (Class)

This is a base class to handle user authentication. This class contains methods to validate user against the CHART User Management Service and it also contains methods to handle the response.

11.2.1.2.2 CHARTMap.Lib.XMLUtility (Class)

This is a base utility class to handle XML. This class contains methods to validate the xml response from the user management and methods to retrieve values from the xml.

11.2.1.3 User Rights – Intranet Map (Class Diagram)

This is a base class to store a list of CHARTFunctionalRights. This class contains methods to serialize and deserialize a list of CHARTFunctionalRights in xml and methods to save to or load from xml file.

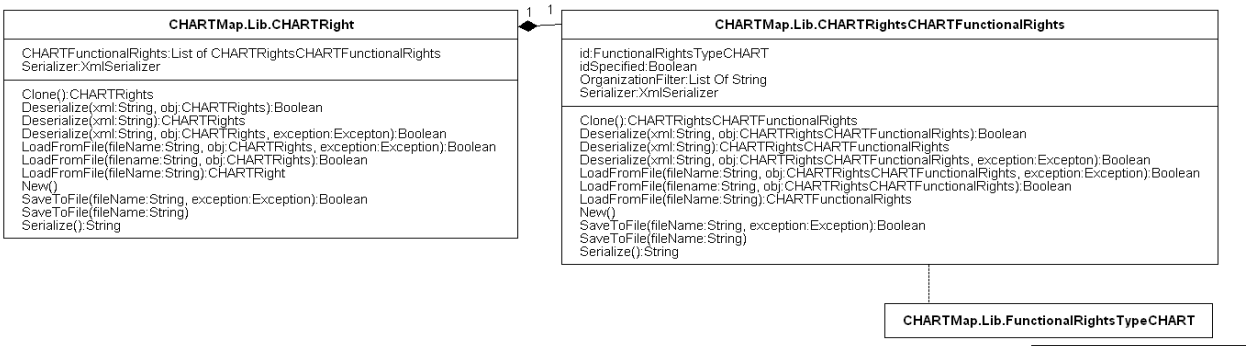


Figure 11-9 Intranet Map User Rights (Class Diagram)

11.2.1.3.1 CHARTMap.Lib.CHARTRights (Class)

This is a base class to store a list of CHARTFunctionalRights. This class contains methods to serialize and deserialize a list of CHARTFunctionalRights in xml and methods to save to or load from xml file.

11.2.1.3.2 CHARTMap.Lib.CHARTRightsCHARTFunctionalRights (Class)

This is a base class to capture the CHARTFunctionalRights. This class contains methods to serialize and deserialize CHARTFunctionalRights in xml and methods to save to or load from xml file.

11.2.1.3.3 CHARTMap.Lib.FunctionalRightsTypeCHART (Class)

This is the base class that contains the constant attributes of FunctionalRightsTypeCHART.

11.2.1.4 Response Data – Intranet Map (Class Diagram)

This is a base class to capture the ResponseMethod and ResponseTypeValue. This class contains methods to serialize and deserialize the ResponseData in xml and methods to save to or load from xml file.

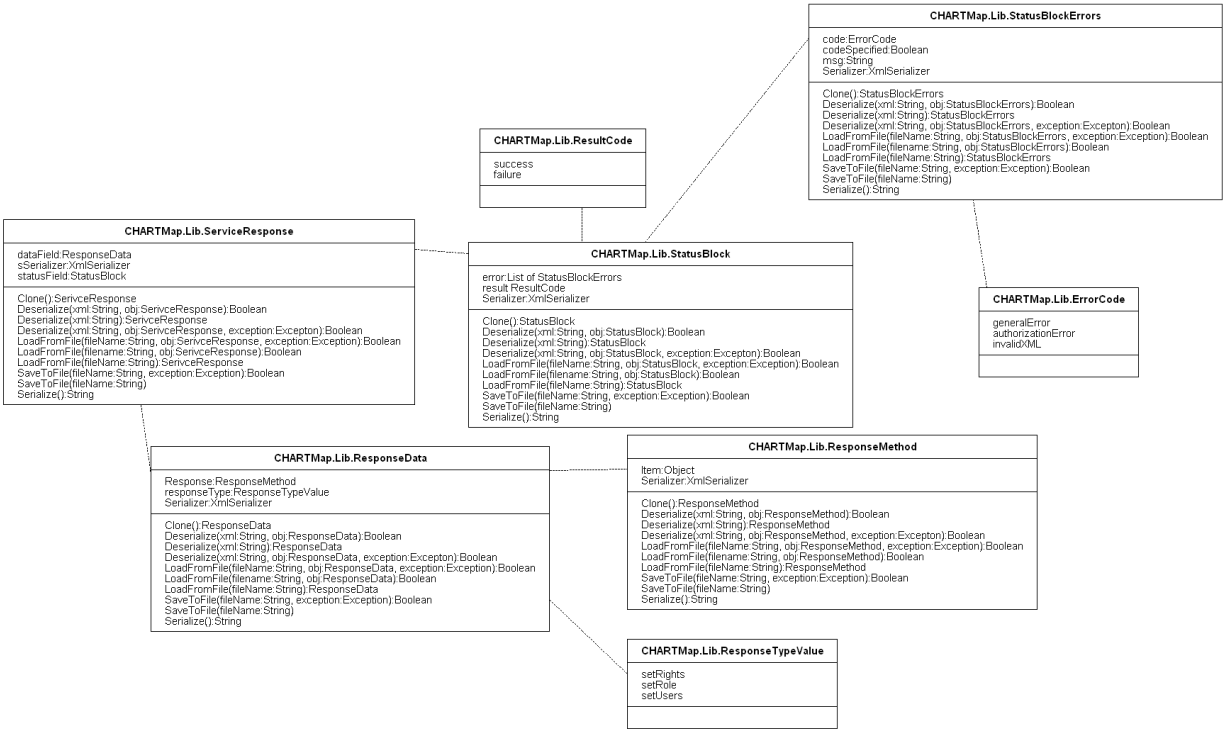


Figure 11-10 Intranet Map Response Data (Class Diagram)

11.2.1.4.1 CHARTMap.Lib.ServiceResponse (Class)

This is a base class to capture the ResponseData and StatusBlock. This class contains methods to serialize and deserialize the ServiceResponse in xml and methods to save to or load from xml file.

11.2.1.4.2 CHARTMap.Lib.ResponseData (Class)

This is a base class to capture the ResponseMethod and ResponseTypeValue. This class contains methods to serialize and deserialize the ResponseData in xml and methods to save to or load from xml file.

11.2.1.4.3 CHARTMap.Lib.ResponseMethod (Class)

This is a base class to capture the Response. This class contains methods to serialize and deserialize the Response in xml and methods to save to or load from xml file.

11.2.1.4.4 CHARTMap.Lib.ResponseTypeValue (Class)

This is the base class that contains the constant attributes of ResponseTypeValue.

11.2.1.4.5 CHARTMap.Lib.StatusBlock (Class)

This is a base class to capture a list of StatusBlockErros and ResultCode. This class contains methods to serialize and deserialize the StatusBlock in xml and methods to save to

or load from xml file.

11.2.1.4.6 CHARTMap.Lib.ResultCode (Class)

This is the base class that contains the constant attributes of ResultCode.

11.2.1.4.7 CHARTMap.Lib.StatusBlockErrors (Class)

This is a base class to capture a list of ErrorCode, the codeSpecified value, and error message. This class contains methods to serialize and deserialize the StatusBlockErrors in xml and methods to save to or load from an xml file.

11.2.1.4.8 CHARTMap.Lib.ErrorCode (Class)

This is the base class that contains the constant attributes of ErrorCode.

11.2.1.5 Client Request - Intranet Map (Class Diagram)

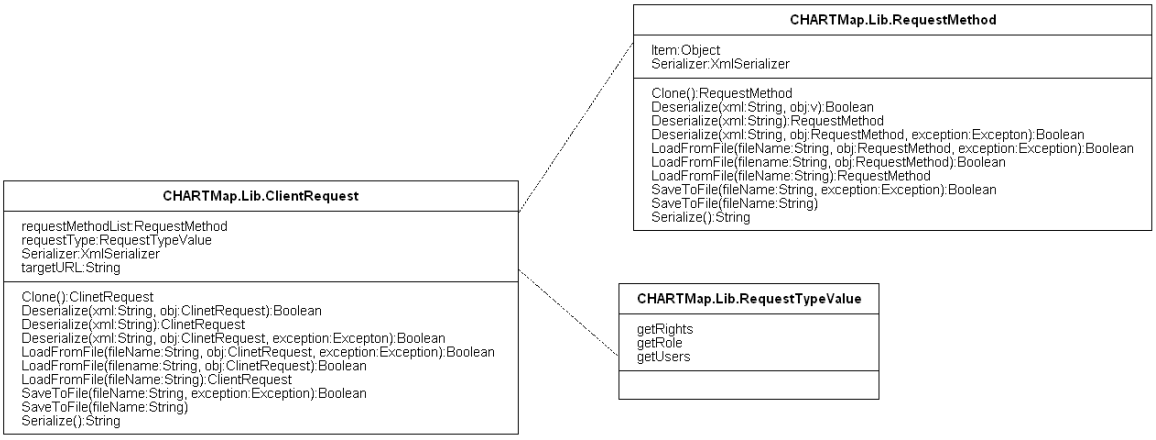


Figure 11-11 Intranet Map Client Request (Class Diagram)

11.2.1.5.1 CHARTMap.Lib. ClientRequest (Class)

This is a base class to capture the RequestMethod, RequestTypeValue and the targetURL. This class contains methods to serialize and deserialize the ClientRequest in xml and methods to save to or load from xml file.

11.2.1.5.2 CHARTMap.Lib.RequestMethod (Class)

This is a base class to capture the Request. This class contains methods to serialize and deserialize the RequestMethod in xml and methods to save to or load from xml file.

11.2.1.5.3 CHARTMap.Lib.RequestTypeValue (Class)

This is the base class that contains the constant attributes of RequestTypeValue.

This class diagram defines a WebServiceModule used for providing a web service interface for Exporting User Functional Rights data. It utilized the Chart WebService framework. The UserManagerHandler is the main class responsible for providing methods to retrieve information in an exportable form.

11.2.2 Sequence Diagrams

11.2.2.1 Login to CHART – Intranet Map (Sequence Diagram)

This diagram shows the processing that occurs when a request is received to login to CHART. The process starts by getting the CHART configurations with the CHARTConfig object and creates a CHARTSecurityValidator object where it takes the user name, password and CHART configurations and serializes it using the ClientRequest object. The process then submits the request to the web-based user management service. Once the response is returned, the process validates the response using the XMLUtility object. If the response validates, the process deserializes the response into a ResponseData object and calls the ProcessServiceResponse method to handle the response. If the user is validated, the Login page saves the user session, writes the user information to log file, and redirects the user to the Intranet Map homepage. Otherwise, the user is prompted for error.

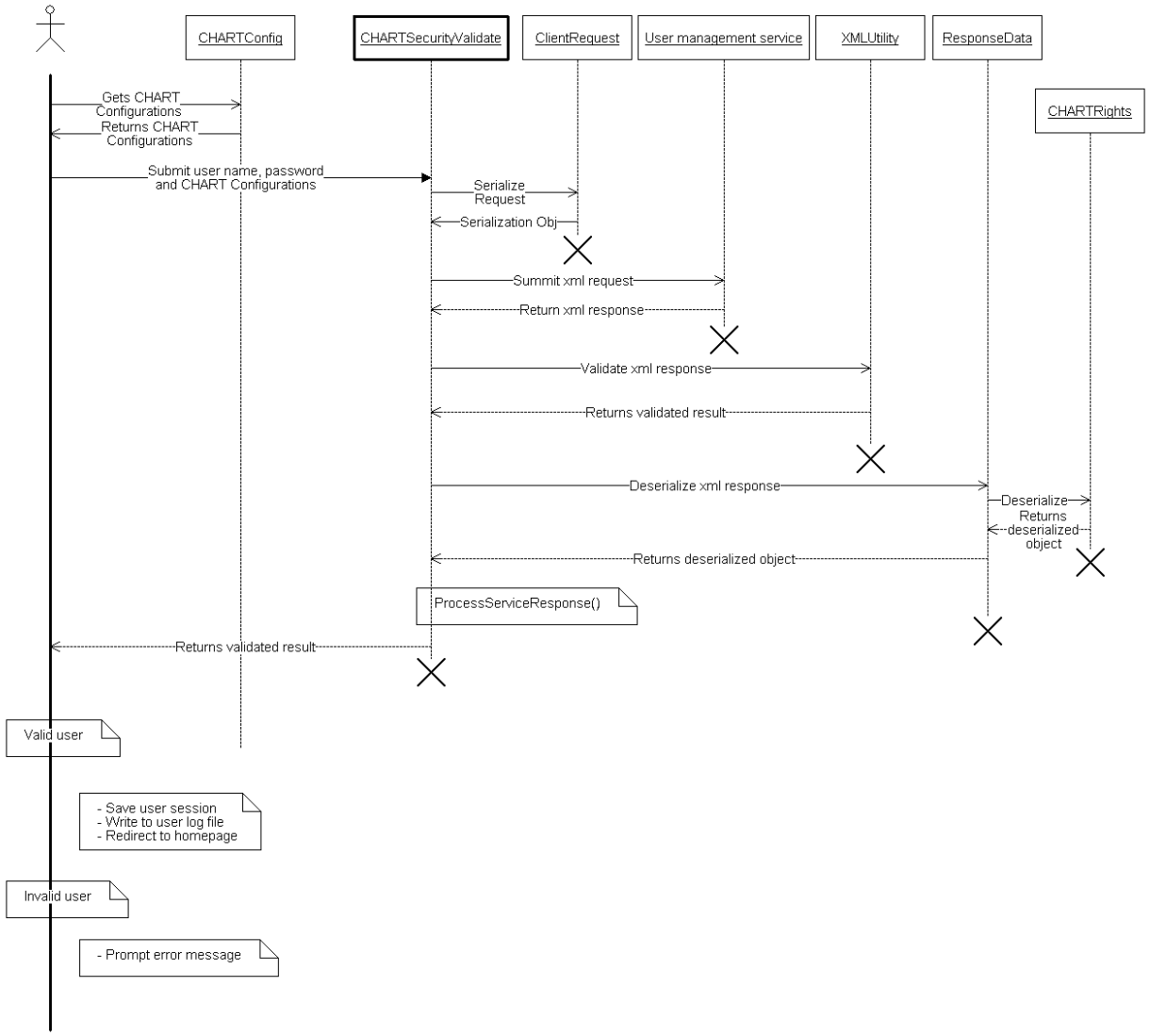


Figure 11-12 Intranet Map Login (Sequence Diagram)

11.2.2.2 Keep Session Alive (Sequence Diagram)

This diagram shows the processing of keeping the user session alive. The process starts by creating a hidden frame (i.e. KeepSessionAlive) in the application; the hidden frame is configured to do the initial authentication for the default user and it also handles the authentication during session refresh behind the scene. In the Page_Load event of the KeepSessionAlive, the process starts by getting the CHART configurations with the CHARTConfig object and creates a CHARTSecurityValidator object where it takes the user name, password and CHART configurations and serializes them using the ClientRequest object. The process then submits the request to the web-based user management service. Once the response is returned, the process validates the response using the XMLUtility object. If the response is valid, the response is deserializes into a ResponseData object and calls the ProcessServiceResponse method to handle the response. Once the response is received, the KeepSessionAlive page calls the OutputTSSInfo() method to write the both TSS organizations and its rights to the client. When the KeepSessionAlive page is loaded, the default page then retrieves the user's organization right for each TSS organization.

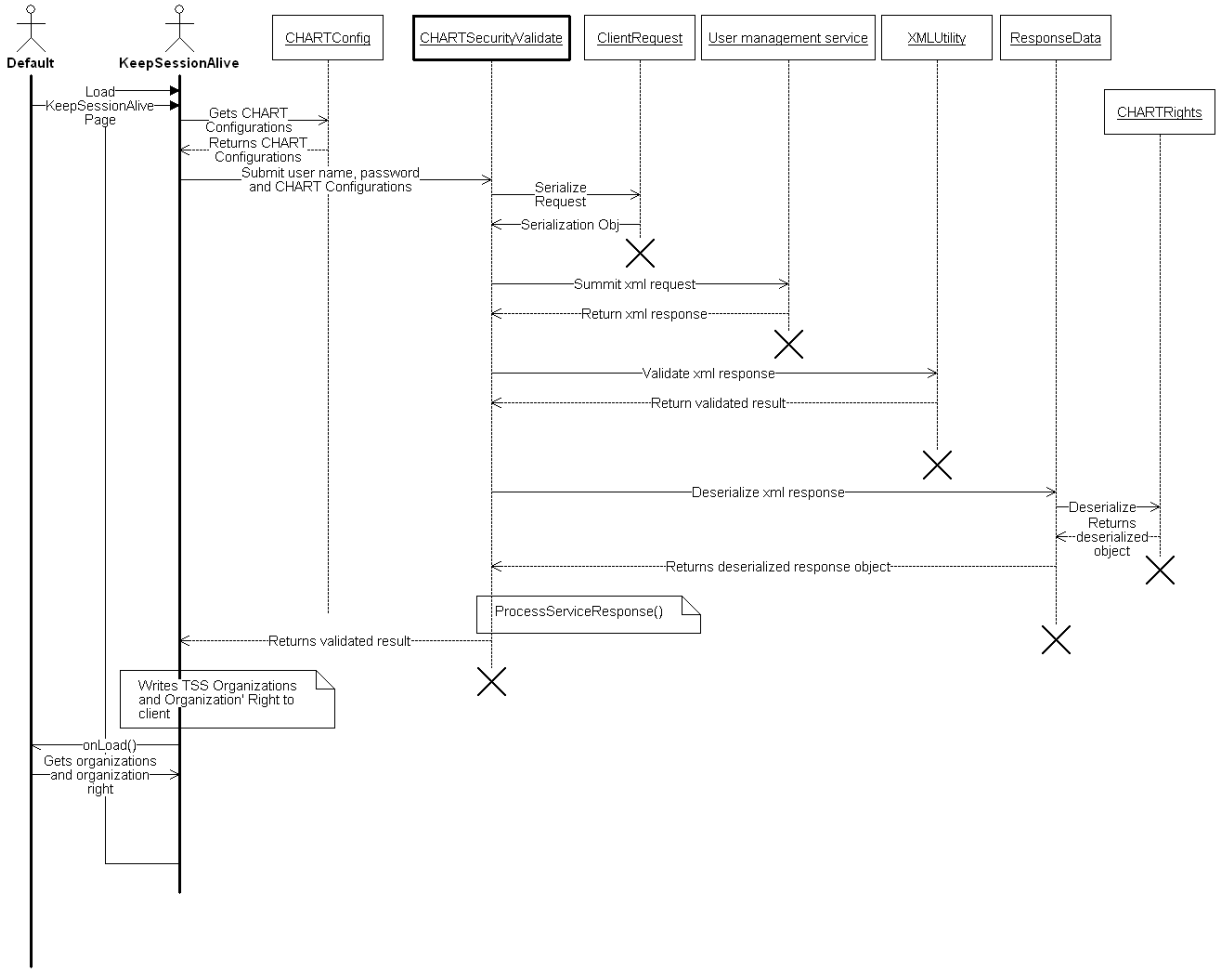


Figure 11-13 Intranet Map Keep Session Alive (Sequence Diagram)

11.3 CHART Data Exporter Web Service

11.3.1 Class Diagrams

11.3.1.1 WSUserManagementModule (Class Diagram)

This class diagram defines a web service module used for providing a web service for exporting user management data such as a list of users and their rights. It utilizes the CHART web service framework. UserManagerRequestHandler is the main class responsible for knowing how to obtain user management information from the internal CORBA User Management Service. It also provides methods for providing user management data in an exportable form.

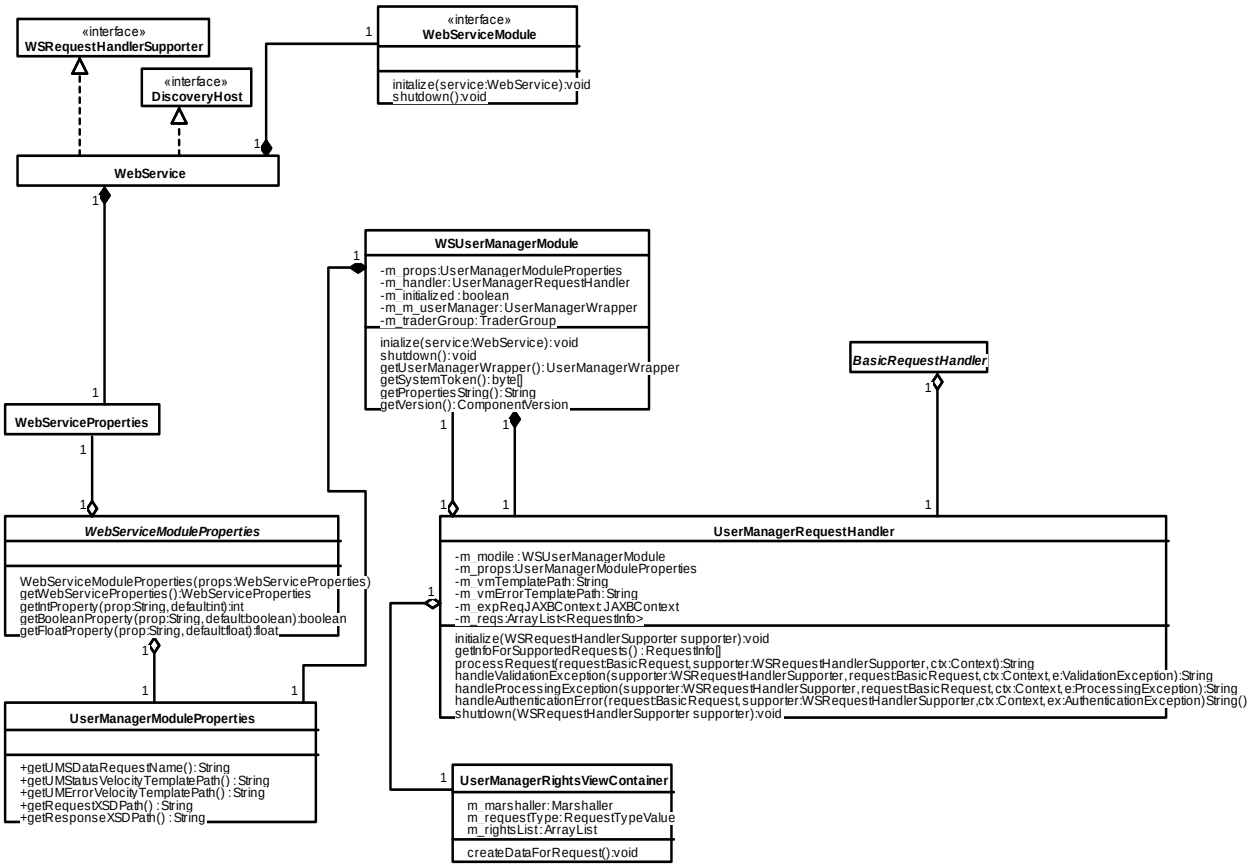


Figure 11-14 WSUserManagerModuleClasses (Class Diagram)

11.3.1.1.1 BasicRequestHandler (Class)

This abstract base class provides an implementation of the WSRequestHandler.processRequest() method that provides optional XML validation

against specified XSD files and optional digital signature verification as well. It is intended to be used by request handlers that plan to take XML in and return XML to the requesting client.

11.3.1.1.2 DiscoveryHost (Class)

This interface defines the methods that the DiscoveryManager relies on. It must be implemented by any class that will create a DiscoveryManager.

11.3.1.1.3 UserManagerModuleProperties (Class)

The UserManagementModuleProperties class provides access methods for properties used by the WSUserManagementModule. It extends the WebServiceModuleProperties class which allows access to other properties available from the WebService Framework.

11.3.1.1.4 UserManagerRequestHandler (Class)

This class extends the BasicRequestHandler to support web-based requests for user services. In addition to supporting all the required validation and authentication methods, it also interacts with the UserManagementModule via the UserManagementWrapper to authenticate user credentials and obtain user rights.

11.3.1.1.5 UserManagerRightsViewContainer (Class)

The UserManagerRightsViewContainer class wraps a ProxyUserManagement object and provides a view of the proxy object specific to UserManagement requests. These objects are used by the Velocity Engine which is made available by the WebService Framework. Velocity will apply a defined UserManagement velocity template to the rights list to generate the XML response to UserManagement web requests.

11.3.1.1.6 WebService (Class)

This class is the core of each Web Service. It extends the VelocityServlet base class and implements the Service CORBA interface so that Web Service servlets can be administered in the same manner as other CHART service applications.

11.3.1.1.7 WebServiceModule (Class)

This interface defines the methods that each module must implement in order to run within the web service framework.

11.3.1.1.8 WebServiceModuleProperties (Class)

This abstract base class provides a base for WebServiceModule implementation classes to extend in order to get access to their configuration properties.

11.3.1.1.9 WebServiceProperties (Class)

This class provides convenient access to the java Properties object that contains

configuration data for the web service and its modules.

11.3.1.1.10WSRequestHandlerSupporter (Class)

This interface defines the methods that will be available to every WSRequestHandler when it is invoked by the framework. It defines the services that the framework will make available to the handlers.

11.3.1.1.11WSUserManagerModule (Class)

The WSUserMangerModule implements the WebServiceModule interface and provides user management functions via the WebService framework.

11.3.1.2 WSDMSExportModule (Class Diagram)

This class diagram defines a WebServiceModule used for providing a web service interface for Exporting device data. The DMS export module is used as a generic example for how all device data is exported. It utilizes the Chart WebService framework. The DMSExportHandler is the main class responsible for maintaining a cache of DMS related objects and providing methods to retrieve information in an exportable form. Note: the DMSExportHandler is not WebService specific and could be used in the context of the Chart ServiceApplication framework if needed.

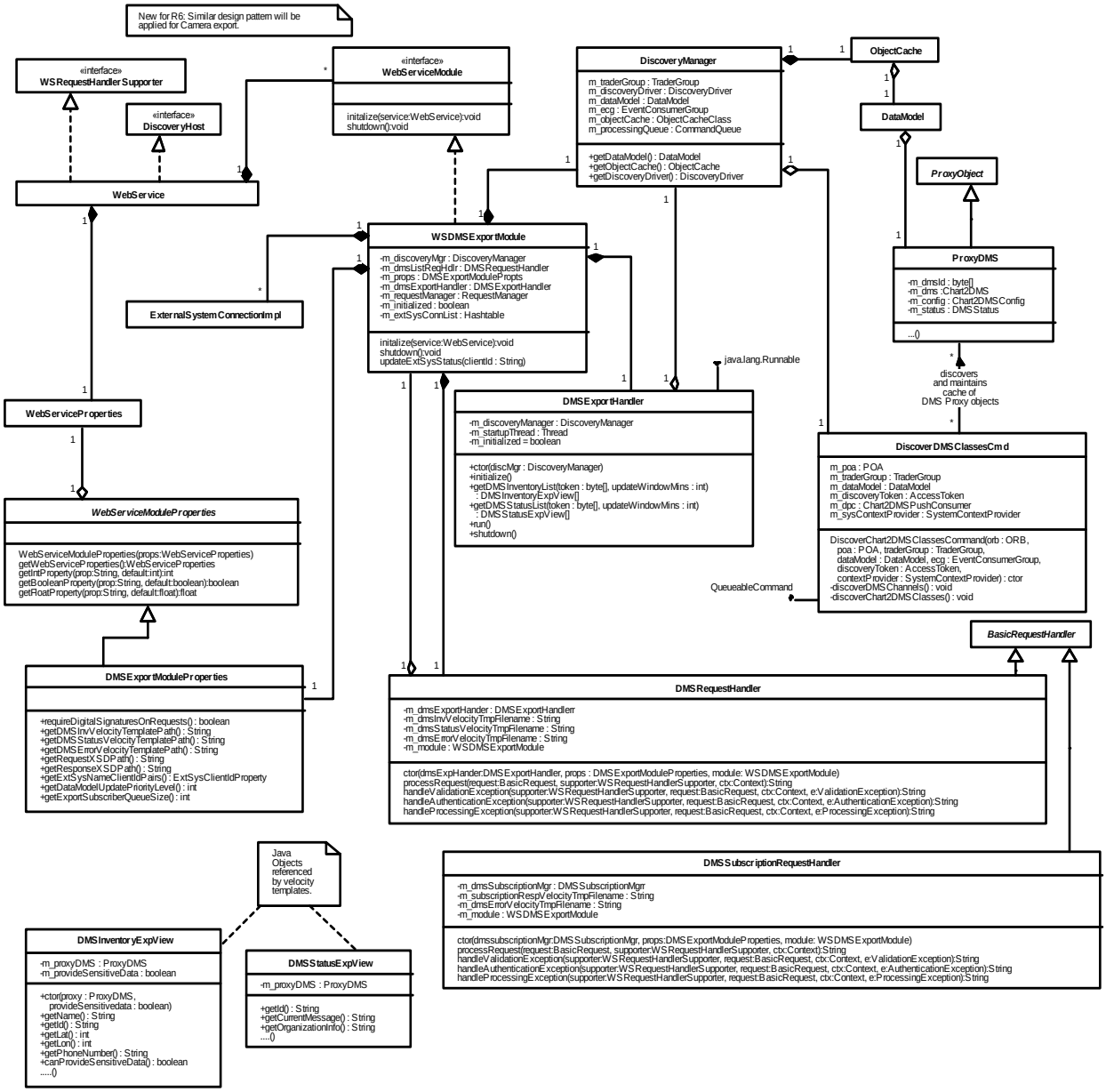


Figure 11-15 WSDMSExportModuleClasses (Class Diagram)

11.3.1.2.1 BasicRequestHandler (Class)

This abstract base class provides an implementation of the WSRequestHandler.processRequest() method that provides optional XML validation against specified XSD files and optional digital signature verification as well. It is intended to be used by request handlers that plan to take XML in and return XML to the calling client.

11.3.1.2.2 DataModel (Class)

The data model class serves as a collection of objects. It provides an efficient lookup

mechanism for locating any object, and methods which allow for the retrieval of all objects of a particular type. Additionally, this class provides the ability to attach observer objects which are notified when objects are added to or removed from the model. Objects may also notify the DataModel that they have been modified. The model will periodically notify all attached observers of the changes to objects in the model.

11.3.1.2.3 DiscoverDMSClassesCmd (Class)

The DiscoverChart2DMSClassesCmd class is responsible for discovering Chart2DMS and Chart2DMSFactory corba objects, wrapping those objects in proxy classes and adding those objects to the DiscoveryManager's Object Cache. This class also listens to appropriate corba events and updates the Object cache accordingly.

11.3.1.2.4 DiscoveryHost (Class)

This interface defines the methods that the DiscoveryManager relies on. It must be implemented by any class that will create a DiscoveryManager.

11.3.1.2.5 DiscoveryManager (Class)

This SystemContextProvider interface defines some of the functionality required by a class which provides discovery services for CHART services. It is used by both the CHART GUI and the CHART backend services. A class which implements this interface must provide "get" accessor methods for the system profile properties, the data model, and the main processing queue for a service, for instance. It also provides access to the root deployment path and dynamic image path, which is used only by the CHART GUI. For the CHART GUI, this interface is known to be implemented by the MainServlet; for the back end CHART services, this interface is known to be implemented by the Discovery Manager.

11.3.1.2.6 DMSExportHandler (Class)

The DMSExportHandler class is responsible for maintaining up to date Chart DMS information in the ObjectCache. This data is used to support the class methods which provide data in response to web service requests for exporting DMS data.

11.3.1.2.7 DMSExportModuleProperties (Class)

The DMSExportModuleProperties class provides access methods for properties used by the WSDMSExportModule. It Extends the WebServiceModuleProperties class which allows access to other properties available from the WebService Framework.

11.3.1.2.8 DMSInventoryExpView (Class)

The DMSInventoryExpView class wraps a ProxyChart2DMS object and provides a view of the proxy object specific to DMS Inventory requests. These objects are used by the Velocity Engine which is made available by the WebService Framework. Velocity will apply a defined DMS Inventory velocity template to a collection of these objects to

generate the XML response to DMS Inventory export requests.

11.3.1.2.9 DMSRequestHandler (Class)

The DMSRequestHandler extends the BasicRequestHandler abstract class and implements abstract methods used to handle web service requests for exporting DMS information.

11.3.1.2.10 DMSStatusExpView (Class)

The DMSStatusExpView class wraps a ProxyChart2DMS object and provides a view of the proxy object specific to DMS Status requests. These objects are used by the Velocity Engine which is made available by the WebService Framework. Velocity will apply a defined DMS Status velocity template to a collection of these objects to generate the XML response to DMS Status export requests.

11.3.1.2.11 DMSSubscriptionRequestHandler (Class)

The DMSSubscriptionRequestHandler extends the BasicRequestHandler and defines process required to handle DMS export Subscription requests made available by the Chart Export Web Service. Subscriptions allow clients to receive "real time" updates to DMSs as opposed to "on demand" updates which the client has to initiate.

11.3.1.2.12 ExternalSystemConnectionImpl (Class)

This class knows how to maintain the status of external connections and push them up to the GUI. Also, ExternalSystemConnectionAlerts and Notifications can be sent as configured by the admin.

11.3.1.2.13 java.lang.Runnable (Interface)

This interface allows the run method to be called from another thread using Java's threading mechanism.

11.3.1.2.14 ObjectCache (Class)

The ObjectCache is a wrapper for the DataModel. It provides access to DataModel methods to find objects in the data model, delegating those methods to the DataModel itself. It also provides additional methods of finding name filtered objects and discovering "duplicate" objects (as defined by an isDuplicateOf() method of the Duplicatable interface).

11.3.1.2.15 ProxyDMS (Class)

The ProxyChart2DMS object is a proxy for a Chart2DMS corba object which is used to by the DiscoveryManager / ObjectCache. The objects are used to maintain an up to date cache of Chart2DMS data in the object cache for application use.

11.3.1.2.16 ProxyObject (Class)

This class is a base class for many types of proxy objects store in the CHART object cache

(which have been discovered by the DiscoveryDriver tasks), used to provide a standard set of access methods for the proxy objects.

11.3.1.2.17QueueableCommand (Interface)

A QueueableCommand is an interface used to represent a command that can be placed on a CommandQueue for asynchronous execution. Derived classes implement the execute method to specify the actions taken by the command when it is executed. This interface must be implemented by any device command in order that it may be queued on a CommandQueue. The CommandQueue driver calls the execute method to execute a command in the queue and a call to the interrupted method is made when a CommandQueue is shut down.

11.3.1.2.18WebService (Class)

This class is the core of each Web Service. It extends the VelocityServlet base class and implements the Service CORBA interface so that Web Service servlets can be administered in the same manner as other CHART service applications.

11.3.1.2.19WebServiceModule (Class)

This interface defines the methods that each module must implement in order to run within the web service framework.

11.3.1.2.20WebServiceModuleProperties (Class)

This abstract base class provides a base for WebServiceModule implementation classes to extend in order to get access to their configuration properties.

11.3.1.2.21WebServiceProperties (Class)

This class provides convenient access to the java Properties object that contains configuration data for the web service and its modules.

11.3.1.2.22WSDMSExportModule (Class)

The WSDMSExportModule implements the WebServiceModule interface and provides DMS export functionality via the WebService framework.

11.3.1.2.23WSRequestHandlerSupporter (Class)

This interface defines the methods that will be available to every WSRequestHandler when it is invoked by the framework. It defines the services that the framework will make available to the handlers.

11.3.2 Sequence Diagrams

The following sequence diagrams use the DMS device module as an example for how all devices export their data. In all cases, you can merely substitute the device type for the letters “DMS” to obtain the actual class names. For example for TSS, HAR, SHAZAM, and CCTV.

11.3.2.1 DMSExportHandler:getDMSInventoryList (Sequence Diagram)

This diagram depicts the processing needed to retrieve DMS Inventory data in response to a specific DMS Inventory export request. If no ProxyDMSFactories are found in the data model this method throws a GeneralException. This in turn will trigger the WebService framework to call the handler's handleProcessingException() method. ProxyDMS objects are retrieved from the ObjectCache. Based on the optional update window parameter, the functional rights of the client specific token passed in and the owning organization of each proxy object, a collection of appropriate export data is created and returned to the caller.

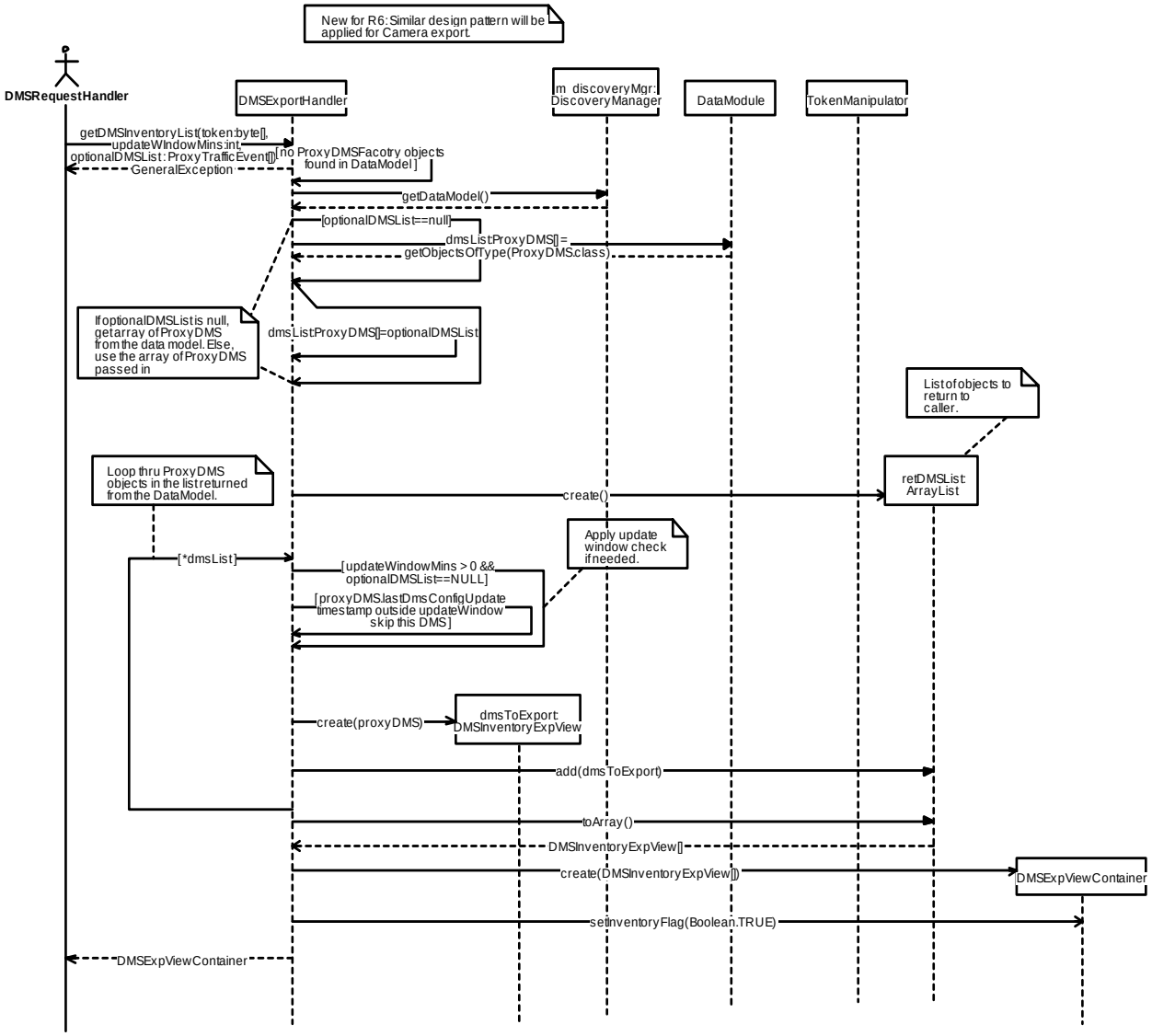


Figure 11-16 DMSExportHandler:getDMSInventoryList (Sequence Diagram)

11.3.2.2 DMSRequestHandler:processRequest (Sequence Diagram)

This diagram depicts the processing of DMS Export Requests for on demand updates and realtime updates (Subscriptions). The processRequest() method of the DMSequestHandler is called by the WebService RequestManger. An "on demand" request for dms data is handled by getting the appropriate data to export from the DMSExportHandler, adding that data to the Velocity Context passed in and finally returning the path to the correct Velocity Template for the request. A request for realtime updates of DMS Data (Subscriptions) is handled by call the appropriate method of the DMSSubscriptionManger for new/renewed subscriptions or canceled subscriptions. In both cases the appropriate information is loaded into the velocity context and the correct velocity template file path is returned to the caller.

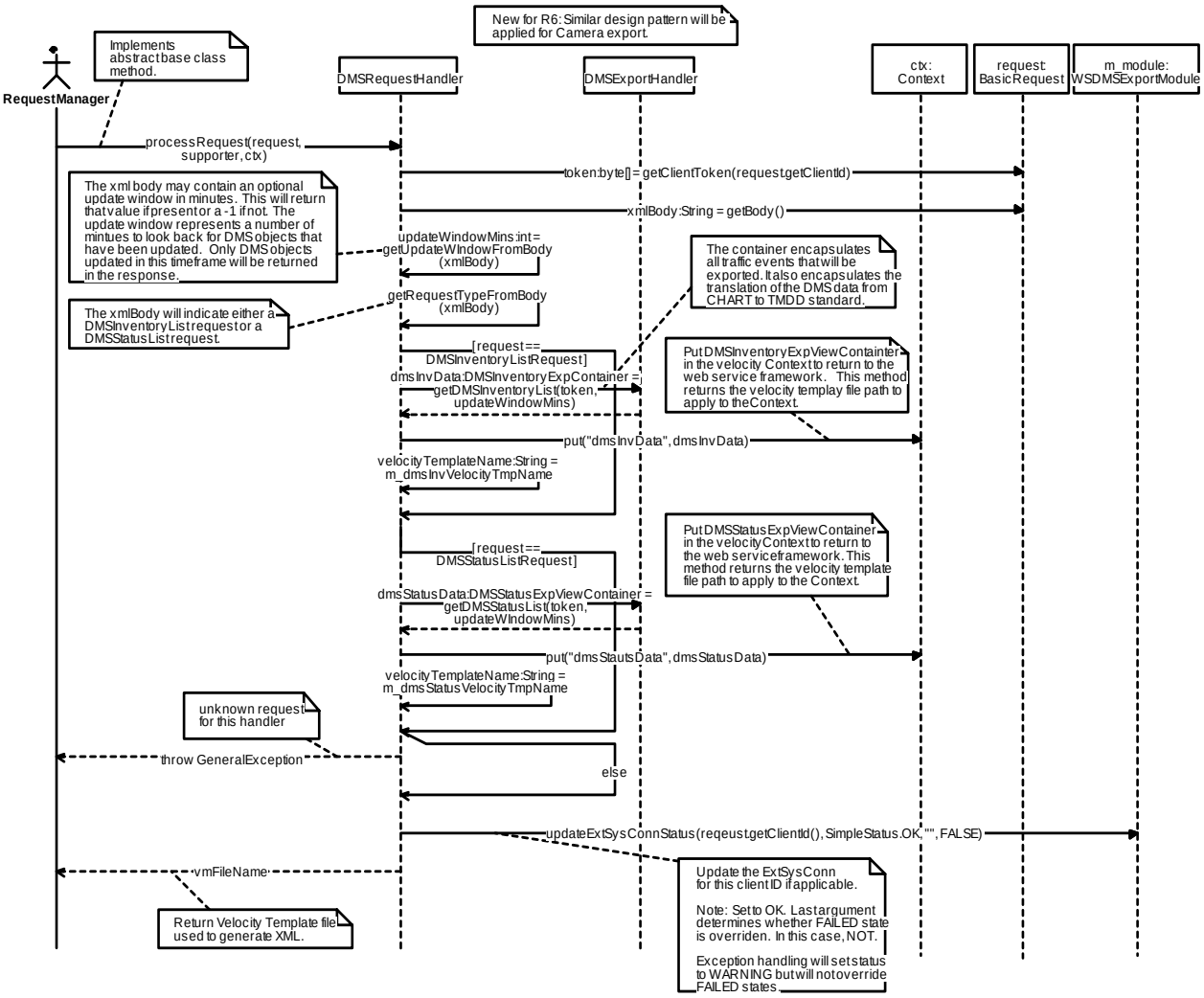


Figure 11-17 DMSRequestHandler:processRequest (Sequence Diagram)

11.3.2.3 DMSSubscriptionRequestHandler:processRequest (Sequence Diagram)

This diagram depicts the processing of DMS subscriptions requests.

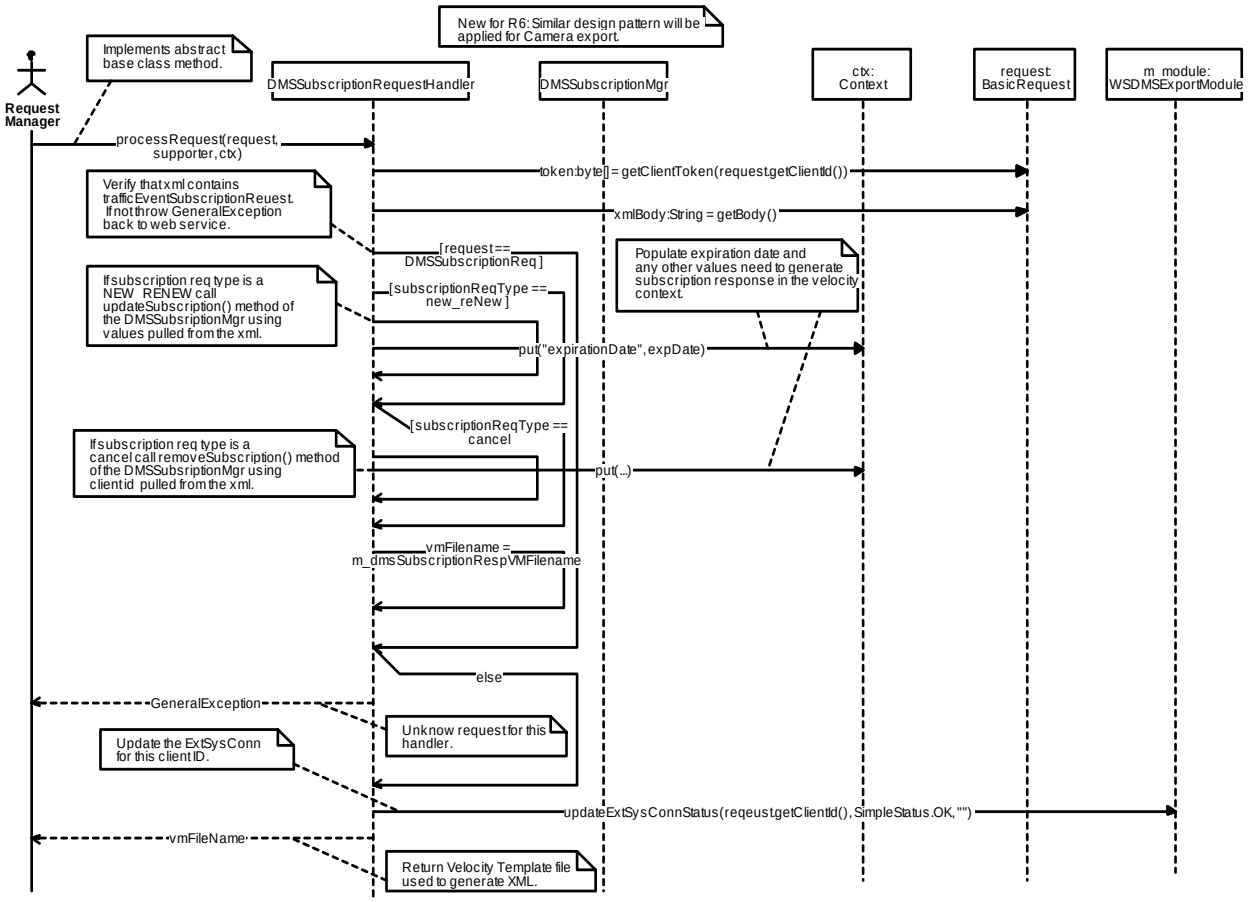


Figure 11-18 DMSSubscriptionRequestHandler:processRequest (Sequence Diagram)

11.3.2.4 DMSExportHandler:getDMSStatusList (Sequence Diagram)

This diagram depicts the processing needed to retrieve DMS Status data in response to a specific DMS Status export request. If no ProxyDMSFactories are found in the data model this method throws a GeneralException. This in turn will trigger the WebService framework to call the handler's handleProcessingException() method. ProxyDMS objects are retrieved from the ObjectCache. Based on the optional update window parameter, the functional rights of the client specific token passed in and the owning organization of each proxy object, a collection of appropriate export data is created and returned to the caller.

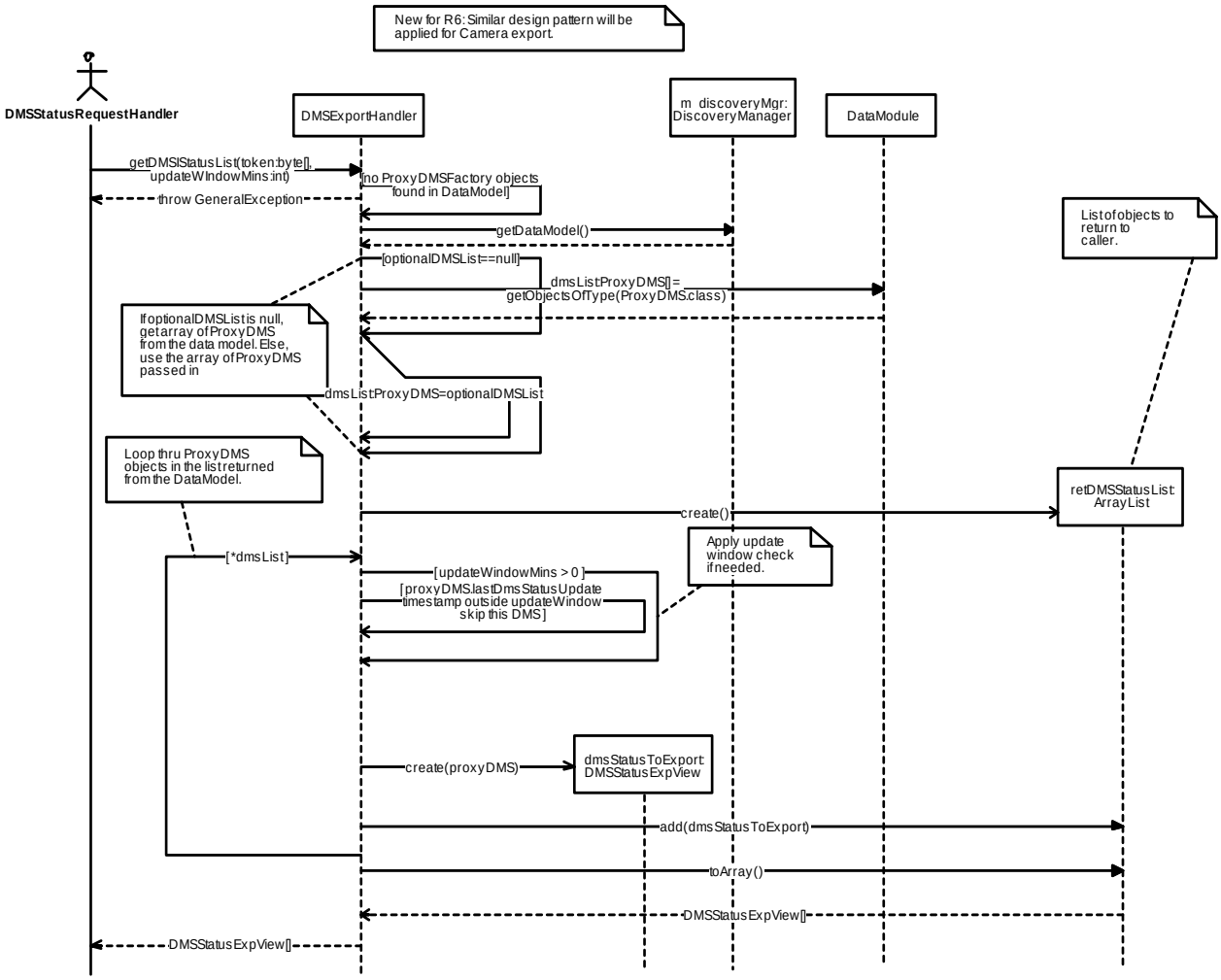


Figure 11-19 DMSExportHandler:getDMSStatusList (Sequence Diagram)

11.4 CHART User Management Web Service

11.4.1 Sequence Diagrams

11.4.1.1 UserManagerRequestHandler:processRequest (Sequence Diagram)

This diagram depicts the processing of the getRights client request. The processRequest() method of UserManagerRequestHandler is called by the WebService RequestManager.GetRights request for Functional Rights. The processing involves getting the appropriate data from Chart User Manager Wrapper, passing that data into the Velocity Context, and finally returning the path to the correct Velocity Template for the request.

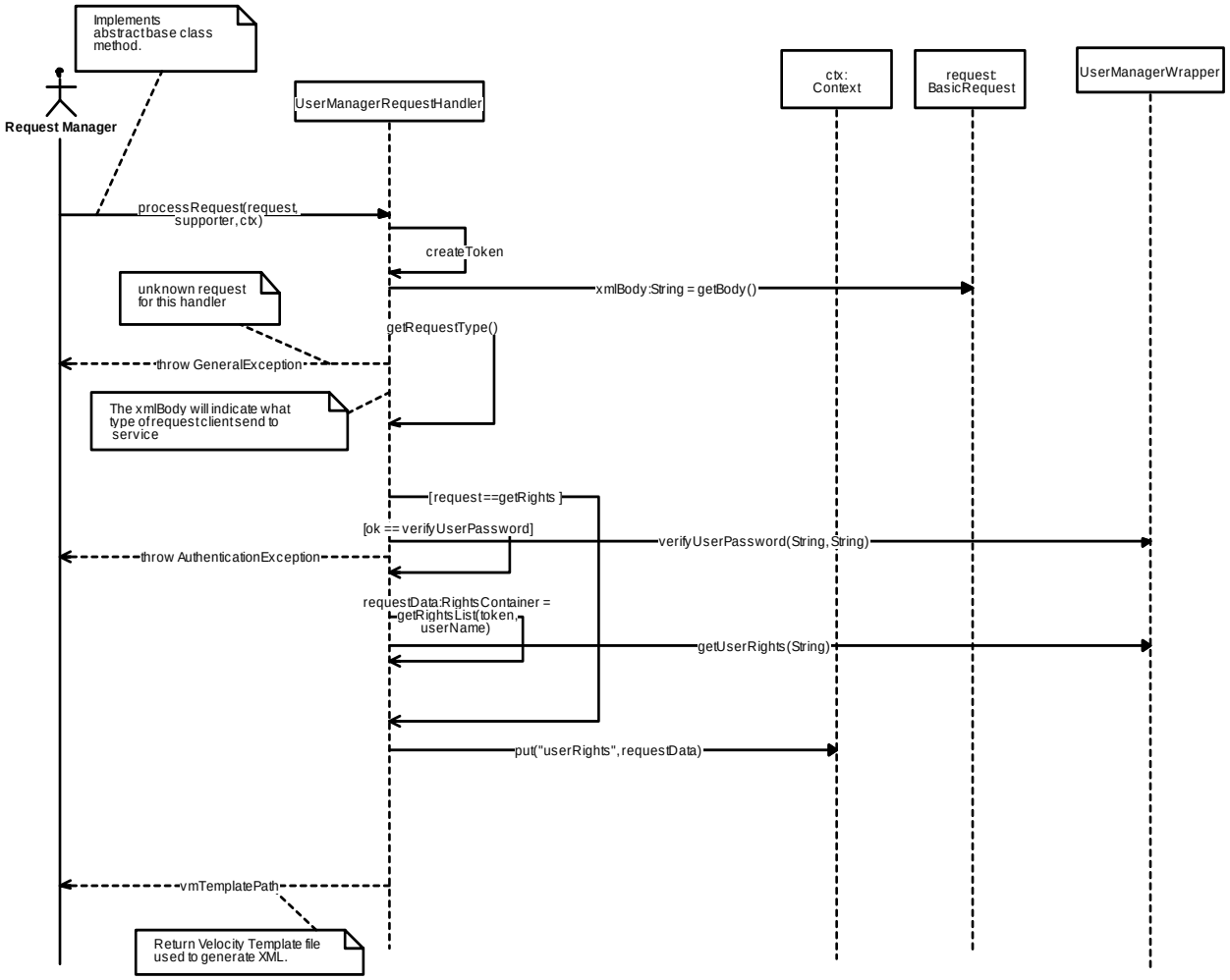


Figure 11-20 UserManagerRequestHandler:processRequest (Sequence Diagram)

11.4.1.2 UserManagerRequestHandler:handleException (Sequence Diagram)

This sequence diagram depicts Exception handling done as part of the WebService framework. The UserManagerRequestHandler derives from the BasicRequestHandler class and implements 3 abstract methods (handleValidationException(), handleAuthenticationException() and handleProcessingException()). These methods are called from the WebService frame work when exceptions are encountered. Each method retrieves information as need from the arguments passed in and creates a response using a Velocity Context object and a predefined Velocity template. Note: for handleValidationException() if inbound xml validation fails, pass the invalid XML string back in the XML response defined as CDATA so it will not be parsed.

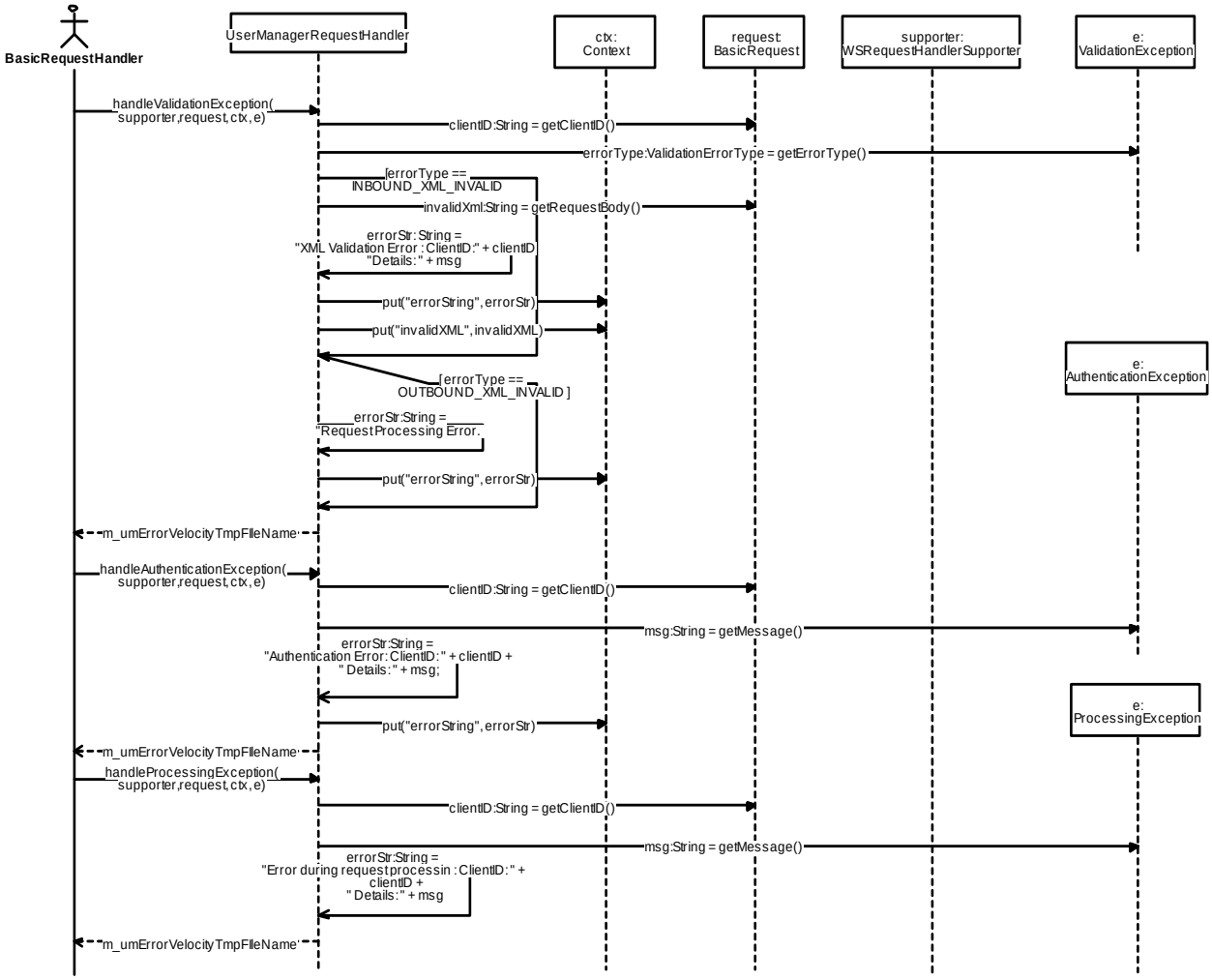


Figure 11-21 UserManagerRequestHandler:handleException (Sequence Diagram)

12 Use Cases – Camera Synchronization

The use case diagrams depict new functionality for the CHART R6 External TSS feature and also identify existing features that will be enhanced. The use case diagrams for this feature exist in the Tau design tool in the Release6 area. The sections below indicate the title of the use case diagrams that apply to this feature.

12.1 CHART Data Exporter

12.1.1 Provide Data to External Systems (Use Case Diagram)

This diagram shows uses of the system related to providing data to external system. R6 CHART provides Traffic Event, DMS, HAR, SHAZAM, TSS, and CCTV to external systems. In addition, R6 provides external client management for the authentication of external users.

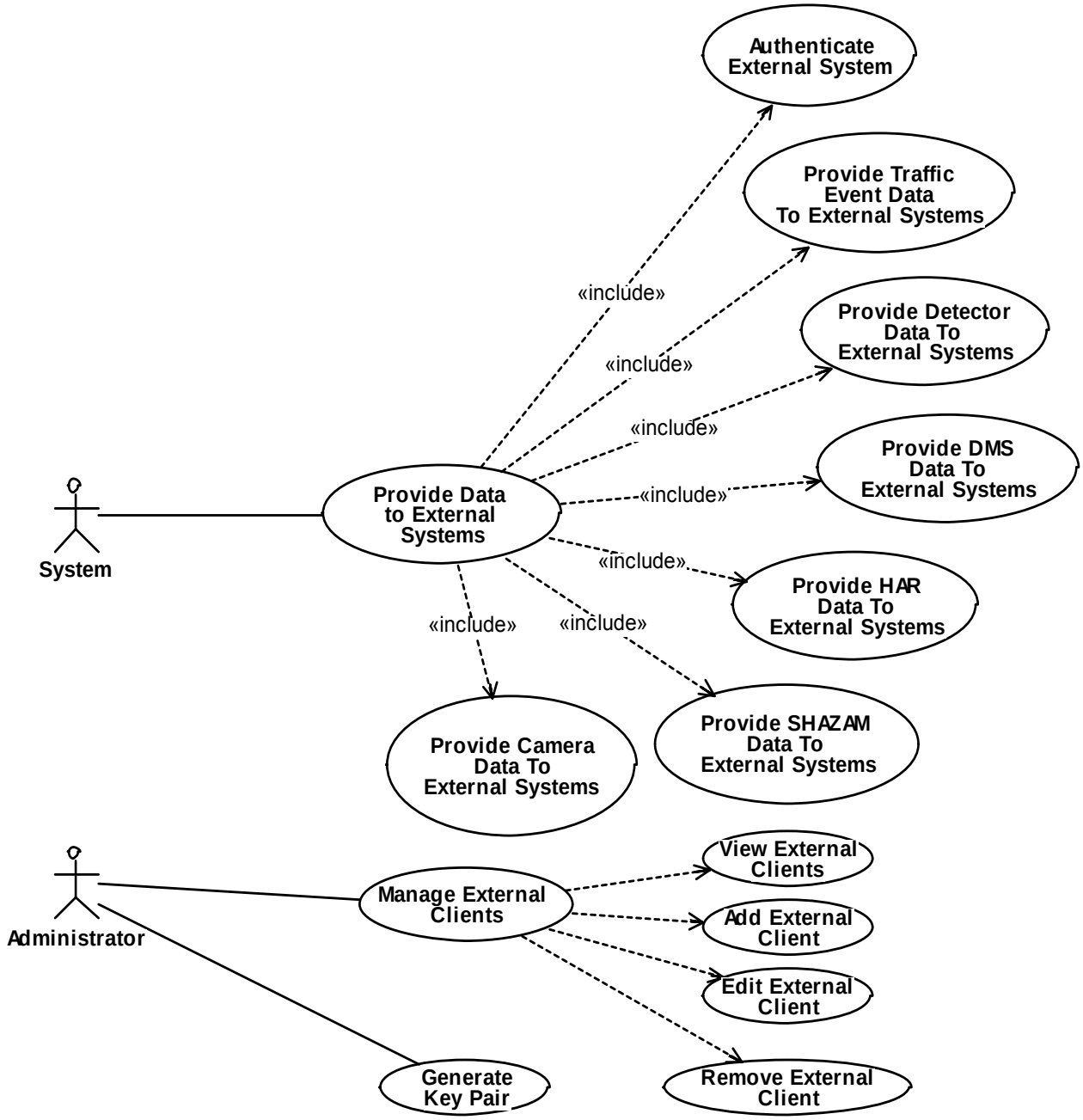


Figure 12-1 R5ProvideDataToExternalSystems (Use Case Diagram)

12.1.1.1 Add External Client (Use Case)

The system shall allow an administrator to add an external client to the CHART system. When doing so, the administrator will specify data pertaining to the client including a client ID to be used by the external system and a public key used to validate data signed by the external system. The administrator will specify whether the external system is a supplier and/or consumer of CHART data, and if it is a consumer, one or more CHART Roles whose user rights will determine the data accessible to the external system. The

administrator will also be able to specify a name and description of the external system, contact information for a person responsible for managing the external system.

12.1.1.2 Authenticate External System (Use Case)

The system shall authenticate external system connections to validate that they are authorized to connect to CHART and to enforce rights regarding the data the external system is permitted to access. Each external system owner will be provided a private key from a public/private key pair generated within the CHART system. Each request from an external system will include the system's CHART client ID (as configured within CHART) and a digital signature of the request data, created using the private key provided by the CHART administrator. The CHART system will validate each request signature using the client's public key.

12.1.1.3 Edit External Client (Use Case)

The system shall allow an administrator to edit the data associated with an external client, as described in the Add External Client use case.

12.1.1.4 Generate Key Pair (Use Case)

The system shall allow an administrator to generate a public/private key pair for use in controlling access to the CHART external interface. The private key is to be given (offline) to the owner of the external system wishing to gain access to CHART data. The public key is to be used by the administrator when adding the external client to the CHART system that corresponds to the external system that wishes to retrieve data from CHART.

12.1.1.5 Manage External Clients (Use Case)

The system shall allow an administrator to manage the external clients that are permitted to retrieve data from CHART via its external system interface.

12.1.1.6 Provide Camera Data to External Systems (Use Case)

The system shall provide Camera data to external systems using the TMDD standard format, with CHART extensions as needed. External systems can obtain an inventory and status of all CHART Camera devices, or the ones that have changed in a certain look back time period. They can obtain updates to the Camera devices (including the current status) periodically with on-demand request or by subscribing to receive updates at a specified web service URL.

12.1.1.7 Provide Data to External Systems (Use Case)

The system shall provide access to external systems via a web service to allow them to receive data that the CHART system makes available to third parties. One or more Roles assigned to each external client will be used to determine the data the client will be permitted to access. All requests made by external systems shall be validated against

published XSD. CHART will return a response XML document for each request. The XML returned will contain an error code and error text for invalid requests, and will return the requested data for valid, authorized requests. The response XML shall be formatted as specified in published XSD.

12.1.1.8 Provide Detector Data to External Systems (Use Case)

The system shall provide detector data to external systems. The system shall enforce granular, organization based user rights to allow the level of detail provided for a detector to be controlled. Two user rights will be used to determine if a detector's detailed volume, speed, and occupancy (VSO) data is exported, only a speed range, or no VSO data. When VSO data is provided for a detector, it will include the data for zone groups and for each zone within the group. The detector data will be provided using the TMDD standard, with CHART extensions as needed. External systems can obtain an inventory and status of all CHART system detectors, or the ones that have changed in a certain look back time period. They can obtain updates to the detector data (including the status) periodically with on-demand request or by subscribing to receive updates at a specified web service URL.

12.1.1.9 Provide DMS Data to External Systems (Use Case)

The system shall allow external systems to receive data pertaining to DMSs using the TMDD standard, with CHART extensions to the standard as needed. External systems can obtain an inventory and status of all CHART DMS devices, or the ones that have changed in a certain look back time period. They can obtain updates to the DMS devices (including the current status) periodically with on-demand request or by subscribing to receive updates at a specified web service URL.

12.1.1.10 Provide HAR Data to External Systems (Use Case)

The system shall allow external systems to receive data pertaining to HARs using the TMDD standard, with CHART extensions to the standard as needed. External systems can obtain an inventory and status of all CHART HAR devices, or the ones that have changed in a certain look back time period. They can obtain updates to the HAR devices (including the current status) periodically with on-demand request or by subscribing to receive updates at a specified web service URL.

12.1.1.11 Provide SHAZAM Data to External Systems (Use Case)

The system shall provide SHAZAM (beacons) data to external systems using the TMDD standard format, with CHART extensions as needed. External systems can obtain an inventory and status of all CHART SHAZAM devices, or the ones that have changed in a certain look back time period. They can obtain updates to the SHAZAM devices (including the current status) periodically with on-demand request or by subscribing to receive updates at a specified web service URL.

12.1.1.12 Provide Traffic Event Data to External Systems (Use Case)

The system shall allow external systems to receive traffic event data from CHART. The CHART system shall enforce granular, organization based user rights to determine the level of detail that may be seen by each event. User rights will control whether or not the incident type of "fatality" is exported within the event name and the incident type field. A cleansed version of the incident type and event name that substitutes personal injury for fatality will be exported to external clients that don't have the proper user right. A separate user right determines whether or not event history for an event is exported. The traffic event data is exported using the SAE ATIS J2354 standard format with CHART extensions as needed. External systems can obtain a list of traffic events (an inventory) that includes either all events in the system or the ones that have changed in a certain look back time period. They can receive updates by polling to receive a new inventory periodically or by subscribing to receive updates at a specified web service URL.

12.1.1.13 Remove External Client (Use Case)

The system shall allow an administrator to remove an external client from the system, effectively preventing them from accessing CHART's external interface to retrieve data. The system will prompt the user for confirmation before removing the client.

12.1.1.14 View External Clients (Use Case)

The system shall allow an administrator to view the external clients allowed to retrieve CHART data via its external interface.

12.2 Data Exporter Client

12.2.1 R6ExportClientImportDataFromDataExporter (Use Case Diagram)

This diagram shows uses of the system related to import data from CHART Export Service. For R5 ExportClient will import Traffic Event, HAR, DMS, and SHAZAM data from CHART Export Service. For R6 ExportClient will import Camera data from CHART ExportService.Sends the Http request to Map Application Alert Map Application for changes in device configuration

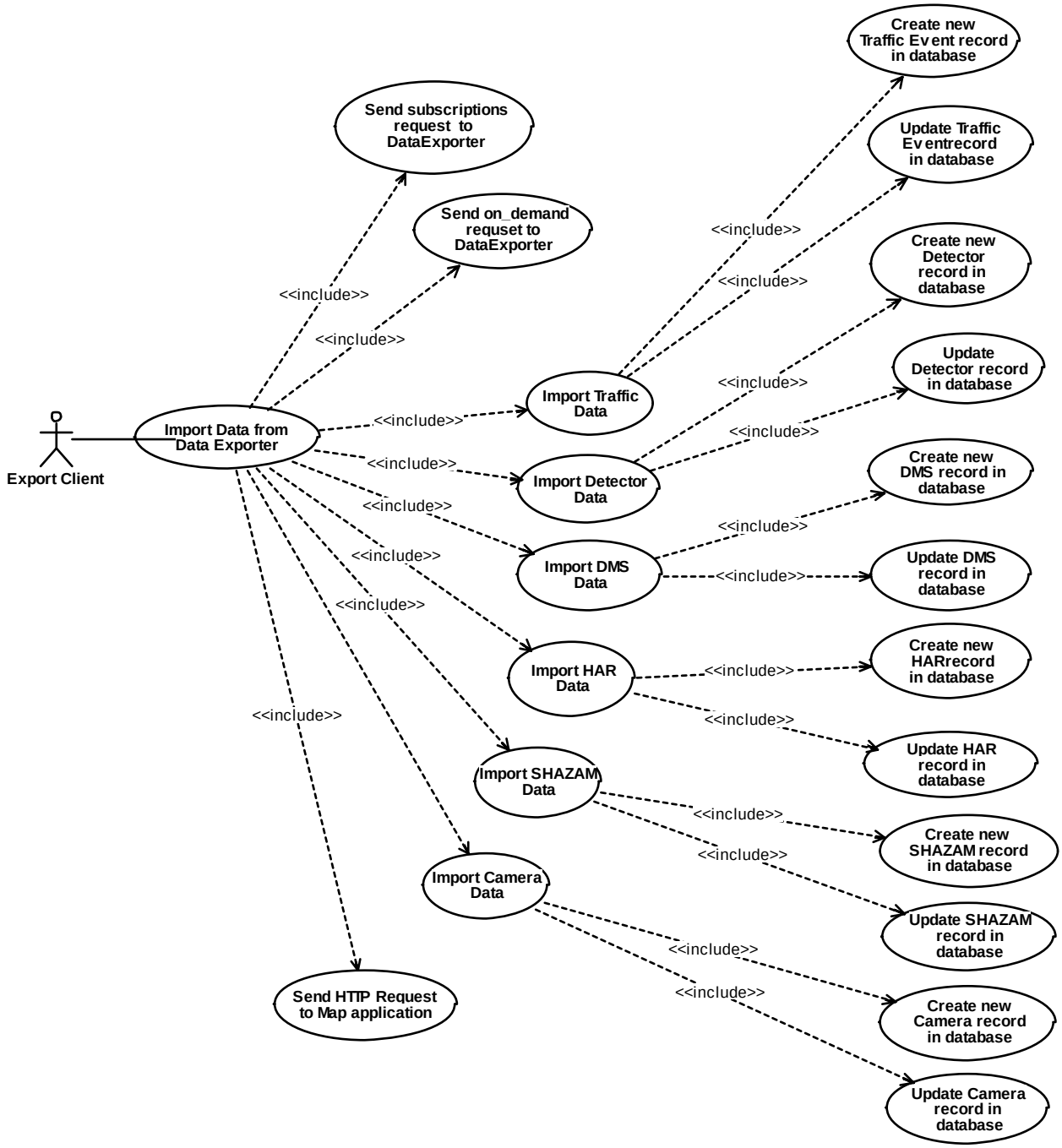


Figure 12-2 R6ExportClientImportDataFromDataExporter (Use Case Diagram)

12.2.1.1 Create new Detector record in database (Use Case)

Process data for new TSS.

12.2.1.2 Create new DMS record in database (Use Case)

Process new DMS data.

12.2.1.3 Create new HARrecord in database (Use Case)

Process new HAR data.

12.2.1.4 Create new SHAZAM record in database (Use Case)

Process new SHAZAM data.

12.2.1.5 Create new Traffic Event record in database (Use Case)

Process data for the new Traffic event.

12.2.1.6 Create new Camera record in database (Use Case)

Process new camera data.

12.2.1.7 Import Camera Data (Use Case)

Process Camera data.

12.2.1.8 Import Data from Data Exporter (Use Case)

The Export Client shall provide access to CHART Data Exporter via web service to allow import data available to third parties. All requests made by Export Client shall be validated against published XSD. CHART will return a response XML document for each request. The XML returned will contain an error code and error text for invalid requests, and will return the requested data for valid, authorized requests.

12.2.1.9 Import Detector Data (Use Case)

Process TSS data.

12.2.1.10 Import DMS Data (Use Case)

Process DMS data.

12.2.1.11 Import HAR Data (Use Case)

Process HAR data.

12.2.1.12 Import SHAZAM Data (Use Case)

Process SHAZAM data.

12.2.1.13 Import Traffic Data (Use Case)

Process Traffic Data.

12.2.1.14 Send HTTP Request to Map application (Use Case)

Alert Map Application when device configuration changed.

12.2.1.15 Send on_demand request to DataExporter (Use Case)

Request to get data on demand.

12.2.1.16 Send subscriptions request to DataExporter (Use Case)

Request to subscribe for DataExport service.

12.2.1.17 Update Camera record in database (Use Case)

Process existing camera data.

12.2.1.18 Update Detector record in database (Use Case)

Process existing TSS data.

12.2.1.19 Update DMS record in database (Use Case)

Process existing DMS data.

12.2.1.20 Update HAR record in database (Use Case)

Process existing HAR data.

12.2.1.21 Update SHAZAM record in database (Use Case)

Process existing SHAZAM data.

12.2.1.22 Update Traffic Event record in database (Use Case)

Update existing traffic event data.

12.2.2 R6ExportClientPostRequestToMapApplication (Use Case Diagram)

This diagram shows uses of the system related to send HTTP request to alert Map Application about event or device configuration is changed.

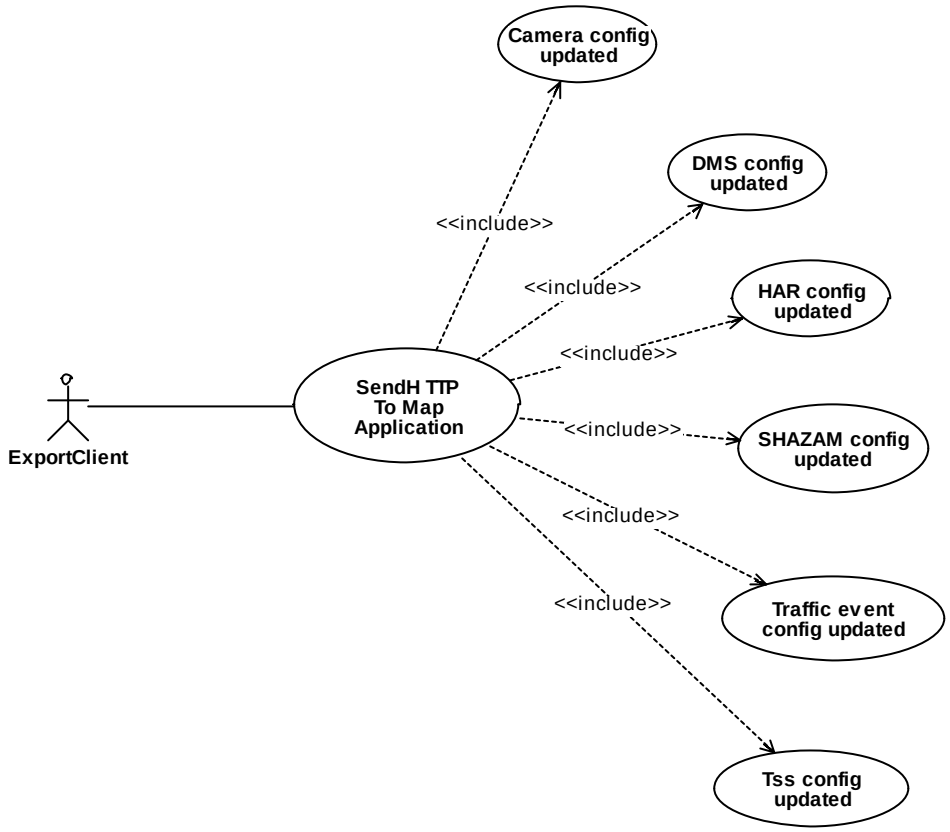


Figure 12-3 R6ExportClientPostRequestToMapApplication (Use Case Diagram)

12.2.2.1 Camera config updated (Use Case)

Send request when received CCTV inventory.

12.2.2.2 DMS config updated (Use Case)

Send request when received DMS inventory.

12.2.2.3 HAR config updated (Use Case)

Send request when received HAR inventory.

12.2.2.4 SendH TTP To Map Application (Use Case)

Send Http request to Mapping Application and also alert the Mapping Application for changes in device configuration

12.2.2.5 SHAZAM config updated (Use Case)

Send request when received SHAZAM inventory.

12.2.2.6 Traffic event config updated (Use Case)

Send request when received Traffic event inventory.

12.2.2.7 TSSS config updated (Use Case)

Send request when received TSS inventory.

12.3 Mapping Synchronization Application

12.3.1 Intranet Map Synchronization (Use Case Diagram)

This diagram shows uses of the system related to synchronizing DMS, SHAZAM, HAR, Events and CCTV updates to the CHARTWeb database.

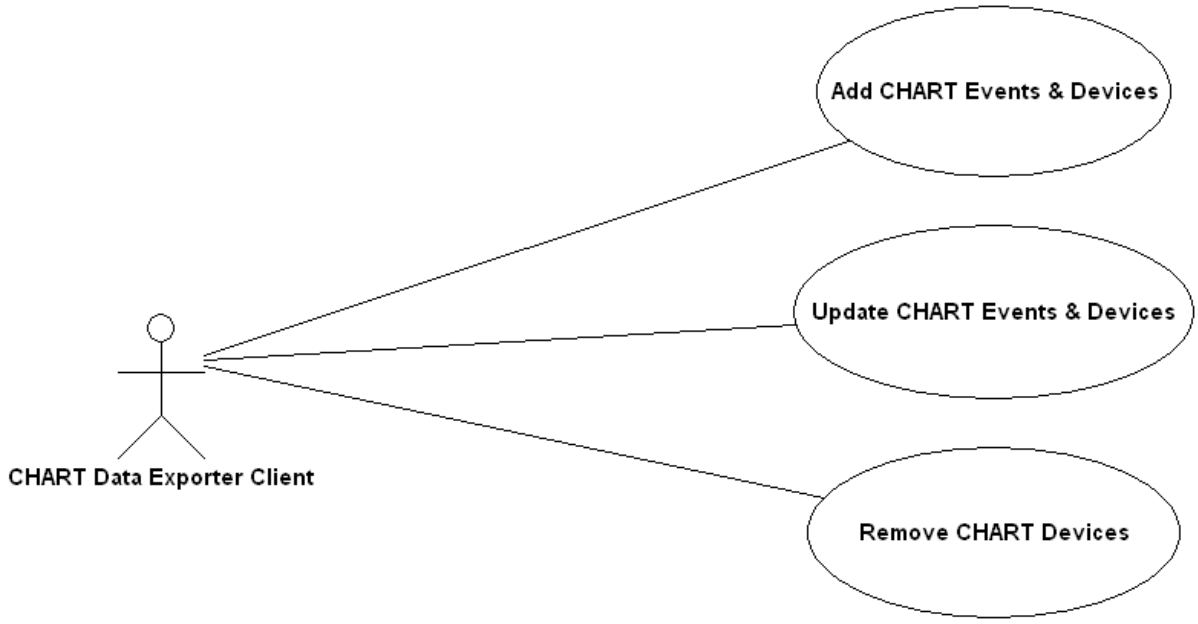


Figure 12-4 Intranet Map Synchronization (Use Case Diagram)

12.3.1.1 Add CHART Events & Devices (Use Case)

The Synchronization Application shall listen to the CHART Data Exporter Client's request when an add event occurred. In R5, the Synchronization Application shall synchronize add events when a new CHART Event, or Device (DMS, HAR, SHAZAM, CAMERA) is added in CHART. The Synchronization Application shall also synchronize any new CHART Events, or Devices (DMS, HAR, SHAZAM, CAMERA) when a full inventory update occurs in CHART Data Exporter Client.

12.3.1.2 Update CHART Events & Devices (Use Case)

The Synchronization Application shall listen to the CHART Data Exporter Client's request when an update event occurred. In R5, the Synchronization Application shall synchronize update events when an existing CHART Event or Device (DMS, HAR, SHAZAM, and CAMERA) is updated in CHART. The Synchronization Application shall also synchronize any updates of CHART Events, or Devices (DMS, HAR, SHAZAM, CAMERA) when a full inventory update occurs in CHART Data Exporter Client.

12.3.1.3 Remove CHART Devices (Use Case)

The Synchronization Application shall listen to the CHART Data Exporter Client's request when a remove event occurred. In R5, the Synchronization Application shall synchronize remove events when an existing CHART Device (DMS, HAR, SHAZAM, and CAMERA) is removed in CHART. The Synchronization Application shall also synchronize any removal of CHART Devices (DMS, HAR, SHAZAM, CAMERA) when a full inventory update occurs in CHART Data Exporter Client. The removal of CHART Events is handled by a nightly schedule job.

13 Detailed Design – Camera Synchronization

13.1 Human-Machine Interface

13.1.1 Data Exporter Server and Client

There is no GUI Interface for Data Exporter and Exporter Client application.

13.1.2 CHART Intranet Mapping GUI

There are no changes to the Intranet Mapping GUI for Camera Locations feature in R6.

13.2 CHART Data Exporter

13.2.1 Class Diagrams

13.2.1.1 WebServicesBaseClasses (Class Diagram)

This diagram shows the classes and interfaces that comprise the Web Service framework. Each WebService reads a properties file to determine which implementations of the WebServiceModule interface should be created and initialized at startup. Each implementation of the WebServiceModule interface can create implementations of the WSRequestHandler interface and install them into the RequestManager.

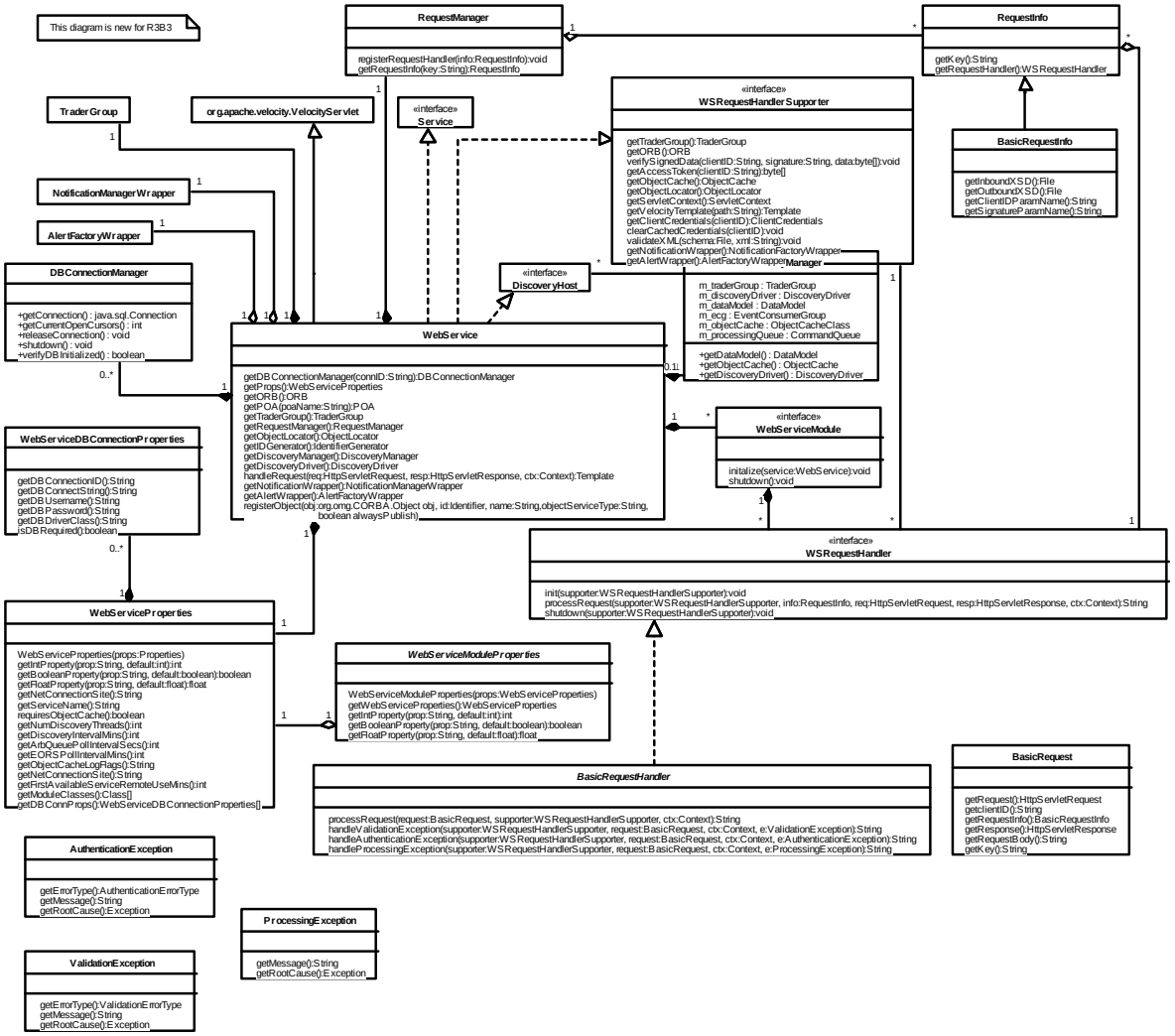


Figure 13-1 WebServicesBaseClasses (Class Diagram)

13.2.1.1.1 AlertFactoryWrapper (Class)

This singleton class provides a wrapper for the Alert Factory that provides automatic location of an Alert Factory and automatic re-discovery should the Alert Factory reference return an error. This class also allows for built-in fault tolerance by automatically failing over to a "working" Alert Factory without the user of this class being aware that this being done. In addition, this class defers the discovery of the Alert Factory until its first use, thus eliminating a start-up dependency for modules that use the Alert Factory.

This class delegates all of its method calls to the system AlertFactory using its currently known good reference to an AlertFactory. If the current reference returns a CORBA failure in the delegated call, this class automatically switches to another reference. When there are no good references (as is true the first time the object is used), this class issues a trader query to (re)discover the published Alert Factory objects in the system. During a method call, the trader will be queried at most one time and under normal circumstances, not at all.

13.2.1.1.2 AuthenticationException (Class)

An instance of this class will be provided to a BasicRequestHandler if the handler has requested digital signature verification but an incoming request does not contain a valid signature.

13.2.1.1.3 BasicRequest (Class)

This class contains data used during requst processing.

13.2.1.1.4 BasicRequestHandler (Class)

This abstract base class provides an implementation of the WSRequestHandler.processRequest() method that provides optional XML validation against specified XSD files and optional digital signature verification as well. It is intended to be used by request handlers that plan to take XML in and return XML to the calling client.

13.2.1.1.5 BasicRequestInfo (Class)

This class provides the request specific data that is required by the BasicRequestHandler implementation of the WSRequestHandler interface.

13.2.1.1.6 DBConnectionManager (Class)

This class implements a database connection manager that manages a pool of database connections. Any CHART II system thread requiring database access gets a database connection from the pool of connections maintained by this manager class. The connections are maintained in two separate lists namely, inUseList and freeList. The inUseList contains connections that have already been assigned to a thread. The freeList contains unassigned connections. This class assumes that an appropriate JDBC driver has been loaded either by using the "jdbc.drivers" system property or by loading it explicitly. The class has a monitor thread that is started by the constructor. This connection monitor thread periodically checks the inuseList to see if there are connections that are owned by dead threads and move such connections to the freeList. The connection monitor thread is started only if a non-zero value is specified for the monitoring time interval in the constructor.

13.2.1.1.7 DiscoveryHost (Class)

This interface defines the methods that the DiscoveryManager relies on. It must be implemented by any class that will create a DiscoveryManager.

13.2.1.1.8 DiscoveryManager (Class)

This SystemContextProvider interface defines some of the functionality required by a class which provides discovery services for CHART services. It is used by both the CHART GUI and the CHART backend services. A class which implements this interface must provide "get" accessor methods for the system profile properties, the data model, and the

main processing queue for a service, for instance. It also provides access to the root deployment path and dynamic image path, which is used only by the CHART GUI. For the CHART GUI, this interface is known to be implemented by the MainServlet; for the back end CHART services, this interface is known to be implemented by the Discovery Manager.

13.2.1.1.9 NotificationManagerWrapper (Class)

This singleton class presents the same interface as the NotificationManager, but uses a FirstAvailableOfferWrapper to provide fault tolerant access to the methods.

13.2.1.1.10 org.apache.velocity.VelocityServlet (Class)

The base class for the Velocity template engine. This template engine is used to provide dynamic content from the CHART GUI Servlet. The web pages are code in templates using velocity specific macros. The code in the servlet loads data that will be shown on the page into a velocity Context, and this VelocityServlet class is used to merge the content with the template to create HTML for the browser to display.

13.2.1.1.11 ProcessingException (Class)

An instance of this class will be provided to a BasicRequestHandler if an unexpected exception is encountered during processing.

13.2.1.1.12 RequestInfo (Class)

This class stores information about a request that the framework will handle. WSRequestHandler instances will register RequestInfo objects to provide the framework information about each request they handle. The framework will pass the registered request information back to the registered WSRequestHandler when the request is being processed.

13.2.1.1.13 RequestManager (Class)

This class provides a mapping from a request key to all information that a particular request handler has registered for the request.

13.2.1.1.14 Service (Class)

This interface is implemented by all services in the system that allow themselves to be shutdown externally. All implementing classes provide a means to be cleanly shutdown and can be pinged to detect if they are alive.

13.2.1.1.15 TraderGroup (Class)

This class provides a facade for trader lookups that allows application level code to be unaware of the number of CORBA trading services that the application is using or the details of the linkage between those services.

13.2.1.1.16ValidationException (Class)

An instance of this class will be provided to a BasicRequestHandler if the handler has requested XSD validation of their incoming or outgoing XML and the XML is found to be not valid. It will contain a ValidationErrorType member that will allow the request handler to determine if the invalid XML was inbound or outbound.

13.2.1.1.17WebService (Class)

This class is the core of each Web Service. It extends the VelocityServlet base class and implements the Service CORBA interface so that Web Service servlets can be administered in the same manner as other CHART service applications.

13.2.1.1.18WebServiceDBConnectionProperties (Class)

This class provides methods for accessing configuration file properties specific to each database connection required for the WebService.

13.2.1.1.19WebServiceModule (Class)

This interface defines the methods that each module must implement in order to run within the web service framework.

13.2.1.1.20WebServiceModuleProperties (Class)

This abstract base class provides a base for WebServiceModule implementation classes to extend in order to get access to their configuration properties.

13.2.1.1.21WebServiceProperties (Class)

This class provides convenient access to the java Properties object that contains configuration data for the web service and its modules.

13.2.1.1.22WSRequestHandler (Class)

This interface defines the methods that every class that wants to handle web service requests must implement.

13.2.1.1.23WSRequestHandlerSupporter (Class)

This interface defines the methods that will be available to every WSRequestHandler when it is invoked by the framework. It defines the services that the framework will make available to the handlers.

13.2.1.2 WSTrafficEventExportModuleClasses (Class Diagram)

This class diagram defines a WebServiceModule used for providing a web service interface for Exporting TrafficEvent data. It utilized the Chart WebService framework. The TrafficEventExportHandler is the main class responsible for maintaining a cache of

TrafficEvent related objects and providing methods to retrieve information in an exportable form. Note: the TrafficEventExportHandler is not WebService specific and could be used in the context of the Chart ServiceApplication framework if needed.

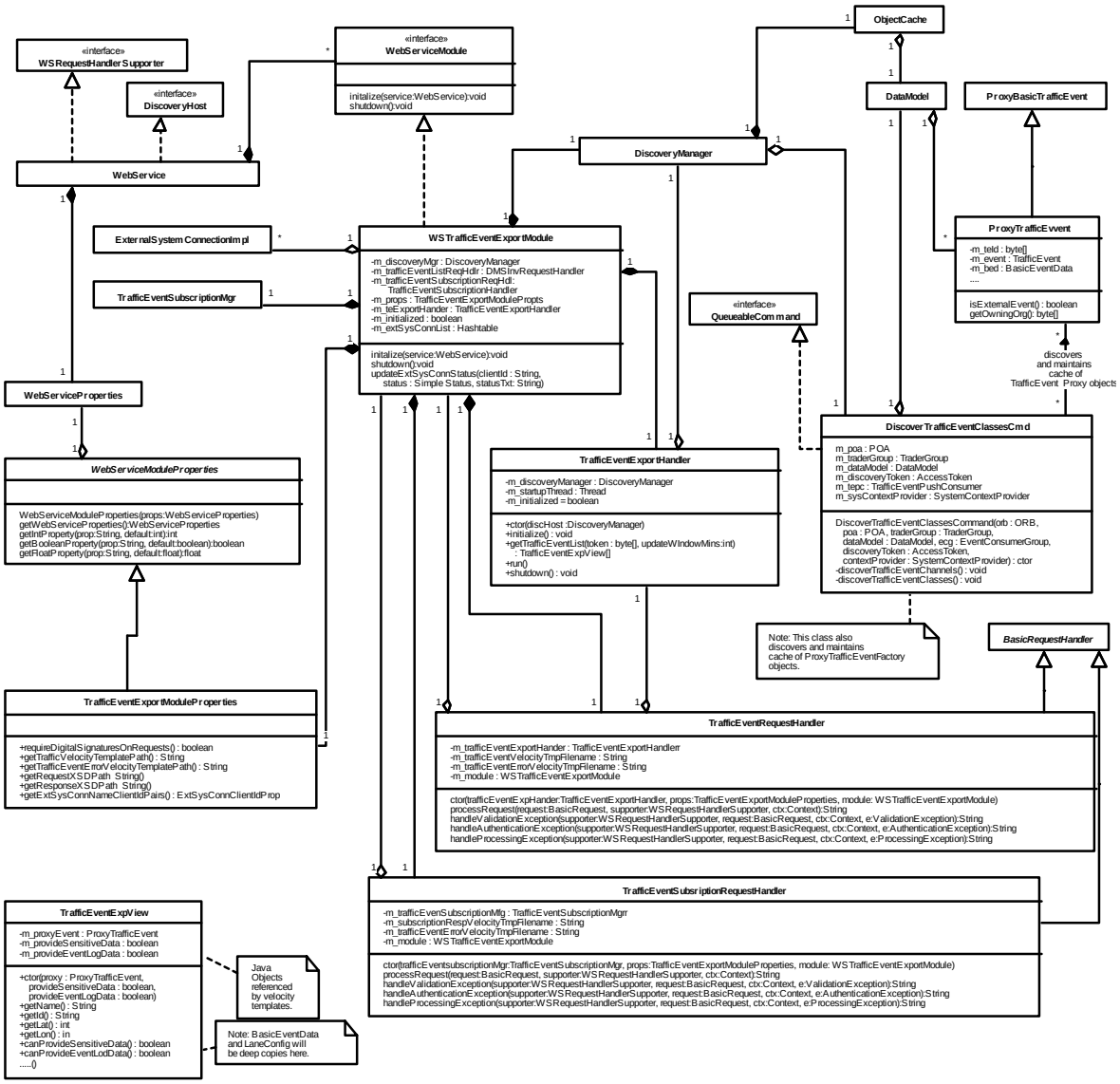


Figure 13-2 WSTrafficEventExportModuleClasses (Class Diagram)

13.2.1.2.1 BasicRequestHandler (Class)

This abstract base class provides an implementation of the WSRequestHandler.processRequest() method that provides optional XML validation against specified XSD files and optional digital signature verification as well. It is intended to be used by request handlers that plan to take XML in and return XML to the calling

client.

13.2.1.2.2 DataModel (Class)

The data model class serves as a collection of objects. It provides an efficient lookup mechanism for locating any object, and methods which allow for the retrieval of all objects of a particular type. Additionally, this class provides the ability to attach observer objects which are notified when objects are added to or removed from the model. Objects may also notify the DataModel that they have been modified. The model will periodically notify all attached observers of the changes to objects in the model.

13.2.1.2.3 DiscoverTrafficEventClassesCmd (Class)

The DiscoverTrafficEventClassesCmd class is responsible for discovering TrafficEvent and TrafficEventFactory corba objects, wrapping those objects in proxy classes and adding those classes to the DiscoveryManager's Object Cache. This class also listens to appropriate corba event channels and updates the Object cache accordingly.

13.2.1.2.4 DiscoveryHost (Class)

This interface defines the methods that the DiscoveryManager relies on. It must be implemented by any class that will create a DiscoveryManager.

13.2.1.2.5 DiscoveryManager (Class)

This SystemContextProvider interface defines some of the functionality required by a class which provides discovery services for CHART services. It is used by both the CHART GUI and the CHART backend services. A class which implements this interface must provide "get" accessor methods for the system profile properties, the data model, and the main processing queue for a service, for instance. It also provides access to the root deployment path and dynamic image path, which is used only by the CHART GUI. For the CHART GUI, this interface is known to be implemented by the MainServlet; for the back end CHART services, this interface is known to be implemented by the Discovery Manager.

13.2.1.2.6 ExternalSystemConnectionImpl (Class)

This class knows how to maintain the status of external connections and push them up to the GUI. Also, ExternalSystemConnectionAlerts and Notifications can be sent as configured by the admin.

13.2.1.2.7 ObjectCache (Class)

The ObjectCache is a wrapper for the DataModel. It provides access to DataModel methods to find objects in the data model, delegating those methods to the DataModel itself. It also provides additional methods of finding name filtered objects and discovering "duplicate" objects (as defined by an isDuplicateOf() method of the Duplicatable interface).

13.2.1.2.8 ProxyBasicTrafficEvent (Class)

This class is used as a proxy for traffic events existing in all traffic event services (including the local service). The proxy traffic events cached are not complete copies of the traffic events, because the full range of data is not needed. The ProxyBasicTrafficEvent data consists of BasicEventData and associated events only (this is why the names of these objects contain the word "Basic", e.g., DiscoverBasicTrafficEventClassesCommand. These proxy traffic events allow every traffic event service in the system to have some knowledge of every traffic event in the entire system, for the purpose of detecting duplicate traffic events.

13.2.1.2.9 ProxyTrafficEvvent (Class)

The ProxyTrafficEvent object is a proxy for a TrafficEvent corba object which is used to by the DiscoveryManager / ObjectCache. The objects are used to maintain an up to date cache of TrafficEvent data in the object cache for application use.

13.2.1.2.10 QueueableCommand (Class)

A QueueableCommand is an interface used to represent a command that can be placed on a CommandQueue for asynchronous execution. Derived classes implement the execute method to specify the actions taken by the command when it is executed. This interface must be implemented by any device command in order that it may be queued on a CommandQueue. The CommandQueue driver calls the execute method to execute a command in the queue and a call to the interrupted method is made when a CommandQueue is shut down.

13.2.1.2.11 TrafficEventExportHandler (Class)

The TrafficEventExportHandler class is responsible for maintaining up to date Chart TrafficEvent information in the ObjectCache. This data is used to support the class methods which provide data in response to web service requests for exporting Traffic Event data.

13.2.1.2.12 TrafficEventExportModuleProperties (Class)

The TrafficEventExportModuleProperties class provides access methods for properties used by the WSTrafficEventExportModule. It Extends the WebServiceModuleProperties class which allows access to other properties available from the WebService Framework.

13.2.1.2.13 TrafficEventExpView (Class)

The TrafficEventExpView class wraps a ProxyTraffic object and provides a view of the proxy object specific to Traffic Event requests. These objects are used by the Velocity Engine which is made available by the WebServcie Framework. Velocity will apply a defined Traffic Event velocity template to a collection of these objects to generate the XML response to TrafficEvent export requests.

13.2.1.2.14TrafficEventRequestHandler (Class)

The TrafficEventRequestHandler extends the BasicRequestHandler and defines process required to handle TrafficEvent export requests made available by the Chart Export Web Service.

13.2.1.2.15TrafficEventSubscriptionMgr (Class)

This class derives from the ExportSubscriptionManager abstract base class and is responsible for maintaining subscribers for Traffic Event data and delivering real time traffic event inventory updates to those subscribers. Note this class is a ModelObserver and as such, it receives updates about Traffic Events from the DataModel.

13.2.1.2.16TrafficEventSubscriptionRequestHandler (Class)

This class implements the BasicRequestHandler and is responsible for supplying the methods to be called when a subscription request is received. It is required to handle validation, authentication and exception processing calls from the Web Service request manager.

13.2.1.2.17WebService (Class)

This class is the core of each Web Service. It extends the VelocityServlet base class and implements the Service CORBA interface so that Web Service servlets can be administered in the same manner as other CHART service applications.

13.2.1.2.18WebServiceModule (Class)

This interface defines the methods that each module must implement in order to run within the web service framework.

13.2.1.2.19WebServiceModuleProperties (Class)

This abstract base class provides a base for WebServiceModule implementation classes to extend in order to get access to their configuration properties.

13.2.1.2.20WebServiceProperties (Class)

This class provides convenient access to the java Properties object that contains configuration data for the web service and its modules.

13.2.1.2.21WSRequestHandlerSupporter (Class)

This interface defines the methods that will be available to every WSRequestHandler when it is invoked by the framework. It defines the services that the framework will make available to the handlers.

13.2.1.2.22WSTrafficEventExportModule (Class)

The WSTrafficEventExportModule implements the WebServiceModule interface and provides TrafficEvent export functionality via the WebService framework.

13.2.1.3 DataExporterUtilityClasses (Class Diagram)

This Class Diagram shows subscription utility classes that are being used by the data exporter application.

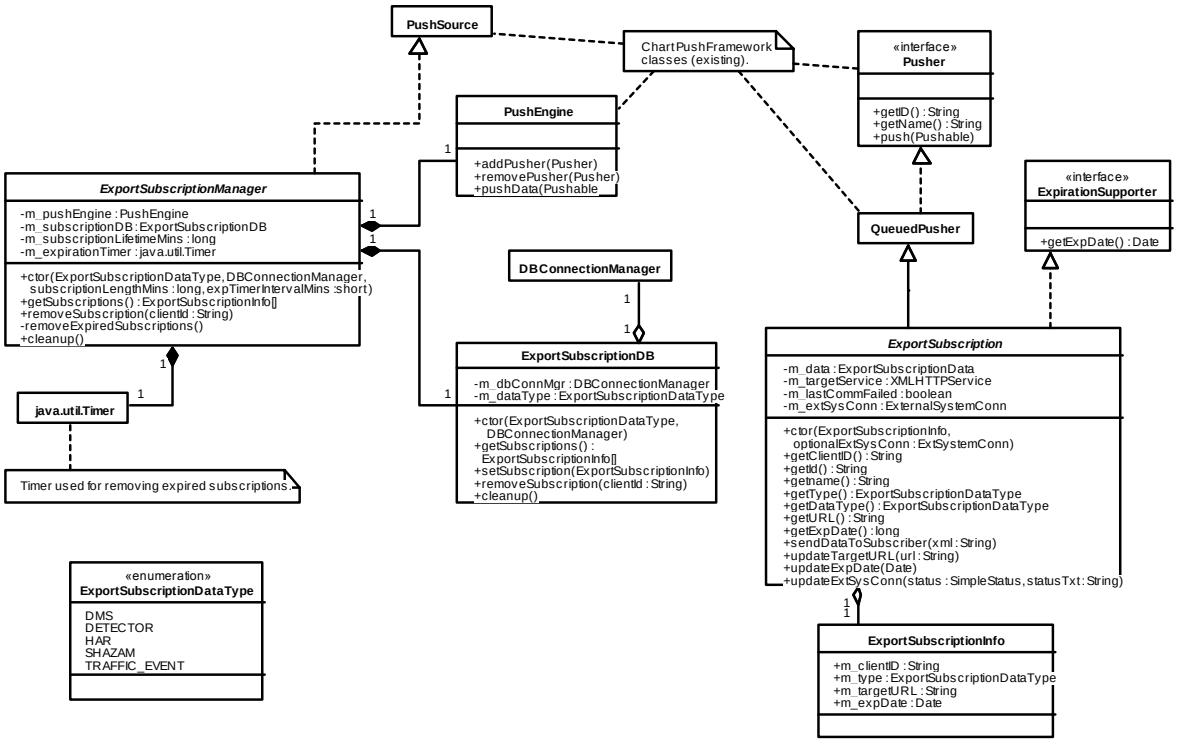


Figure 13-3 DataExporterUtilityClasses (Class Diagram)

13.2.1.3.1 DBConnectionManager (Class)

This class implements a database connection manager that manages a pool of database connections. Any CHART II system thread requiring database access gets a database connection from the pool of connections maintained by this manager class. The connections are maintained in two separate lists namely, inUseList and freeList. The inUseList contains connections that have already been assigned to a thread. The freeList contains unassigned connections. This class assumes that an appropriate JDBC driver has been loaded either by using the "jdbc.drivers" system property or by loading it explicitly. The class has a monitor thread that is started by the constructor. This connection monitor thread periodically checks the inuseList to see if there are connections that are owned by dead threads and move such connections to the freeList. The connection monitor thread is started only if a non-zero value is specified for the monitoring time interval in the constructor.

13.2.1.3.2 ExpirationSupporter (Class)

This interface is implemented by objects that support the concept of an expiration date.

13.2.1.3.3 ExportSubscription (Class)

This abstract class represents an exporter subscription for a specific client ID and data type (DMS, TSS, TrafficEvents, etc.....) and provides generic functionality that is used by derived classes. It extends the QueuePusher class which is part of the chart PushFramework package. It takes an ExportSubscriptionInfo object at creation and provides methods for the subsequent update of the target URL and expiration DateTime. An XMLHTTPService object is create at construction using the provided URL and maintained thru subsequent updateTargetURL() calls. The sendDataToSubscriber() method uses this object to post data to the target system.

13.2.1.3.4 ExportSubscriptionDataType (Class)

This public enumerations represents the types of data the is supported by the Char Data Exporter.

13.2.1.3.5 ExportSubscriptionDB (Class)

This class provides support for persistence and de-persistence for ExportSubriptions of a specific data type which is defined during constructions.

13.2.1.3.6 ExportSubscriptionInfo (Class)

This simple class contains public data members. It is used as a utility class that wraps subscription data in one object.

13.2.1.3.7 ExportSubscriptionManager (Class)

This abstract class provides functionality used to manage ExportSubsriptions. Internally this class uses the CHART PushFramework classes to push data to a list of subscribers (I.E. Pushers). A private Timer/Timer task is used to remove expired subscriptions at configurable intervals. Expired subscriptions are also removed from the DB at startup.

13.2.1.3.8 java.util.Timer (Class)

This class provides asynchronous execution of tasks that are scheduled for one-time or recurring execution.

13.2.1.3.9 PushEngine (Class)

Engine that manages the pushing of Pushable data to consumers via all registered Pushers. All data pushed into the engine will be pushed to all registered Pushers.

13.2.1.3.10 Pusher (Class)

This interface must be implemented by any class that intends to push data to an end consumer. The PushEngine will call the checkPush() and push() methods at the appropriate times.

13.2.1.3.11 PushSource (Class)

This interface must be implemented by any class that wants to use the PushEngine to push data.

13.2.1.3.12 QueuedPusher (Class)

An abstract base class that provides a thread per client implementation of the Pusher interface. This implementation queues up to a specified number of events max, then refuses to queue additional. Events are pushed FIFO and not retried.

13.2.1.4 DMSSubscriptionSupportClasses (Class Diagram)

This diagram shows the classes and interfaces that comprise the DMS Subscription framework. This is used as a generic model for all device subscription classes including the camera classes.

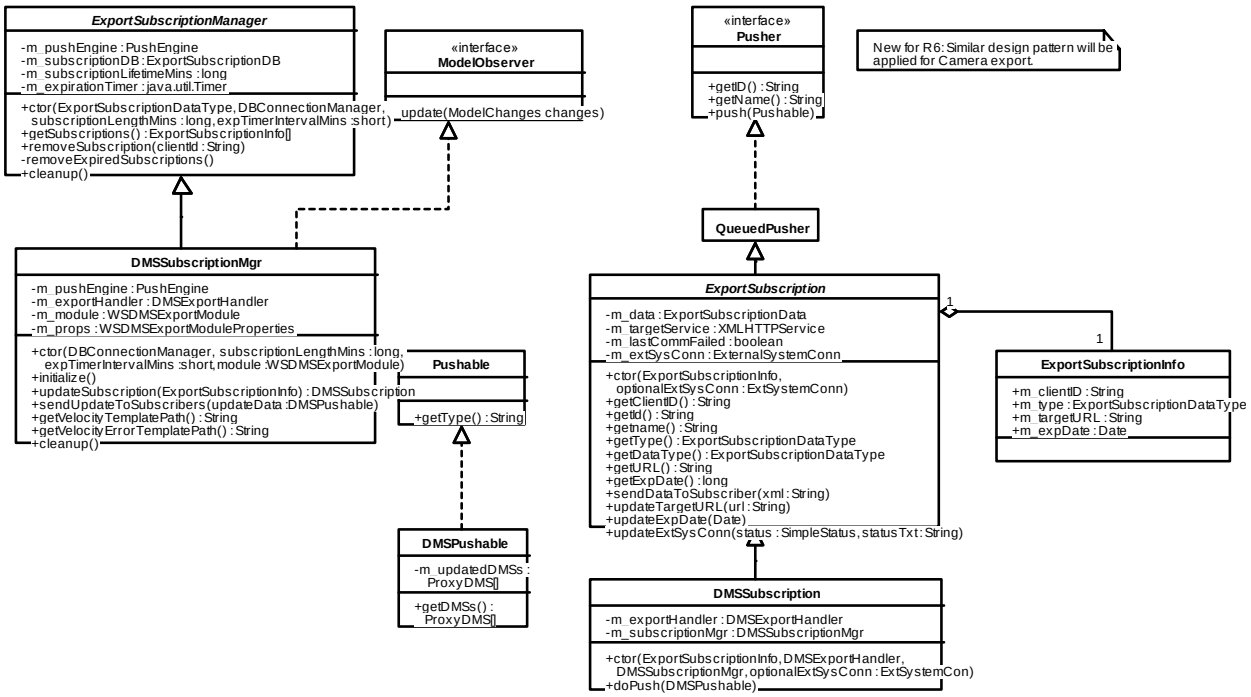


Figure 13-4 DMSSubscriptionSupportClasses (Class Diagram)

13.2.1.4.1 DMSPushable (Class)

This class implements the Pushable interface and represents updates for a list of DMSs that

will be delivered to all DMS export subscribers.

13.2.1.4.2 DMSSubscription (Class)

This class derives from the ExportSubscription abstract base class and implements methods used specifically for processing data for DMS subscribers. The doPush() method implements the QueuedPusher.doPush() abstract method and does the DMS specific processing needed to send realtime DMS inventory or status updates to a subscriber.

13.2.1.4.3 DMSSubscriptionMgr (Class)

This Class derives from the ExportSubscriptionManger abstract base class and is responsible for maintaining subscribers for DMS data and delivering real time DMS inventory and status updates to those subscribers. Note : this class is a ModelObserver and as such, it will receive updates about DMSs from the DataModel.

13.2.1.4.4 ExportSubscription (Class)

This abstract class represents an exporter subscription for a specific client ID and data type (DMS, TSS, TrafficEvents, etc.....) and provides generic functionality that is used by derived classes. It extends the QueuePusher class which is part of the chart PushFramework package. It takes an ExportSubscriptionInfo object at creation and provides methods for the subsequent update of the target URL and expiration DateTime. An XMLHTTPService object is create at construction using the provided URL and maintained thru subsequent updateTargetURL() calls. The sendDataToSubscriber() method uses this object to post data to the target system.

13.2.1.4.5 ExportSubscriptionInfo (Class)

This simple class contains public data members. It is used as a utility class that wraps subscription data in one object.

13.2.1.4.6 ExportSubscriptionManager (Class)

This abstract class provides functionality used to manage ExportSubsriptions. Internally this class uses the CHART PushFramework classes to push data to a list of subscribers (I.E. Pushers). A private Timer/Timer task is used to remove expired subscriptions at configurable intervals. Expired subscriptions are also removed from the DB at startup.

13.2.1.4.7 ModelObserver (Class)

This interface must be implemented by any object which would like to attach to the DataModel as an observer and get updated as system objects are added, deleted or changed.

13.2.1.4.8 Pushable (Class)

The Pushable class is an base class for an object that can be put on a queue by a class that implements the Pusher interface.

13.2.1.4.9 Pusher (Class)

This interface must be implemented by any class that intends to push data to an end consumer. The PushEngine will call the checkPush() and push() methods at the appropriate times.

13.2.1.4.10 QueuedPusher (Class)

An abstract base class that provides a thread per client implementation of the Pusher interface. This implementation queues up to a specified number of events max, then refuses to queue additional. Events are pushed FIFO and not retried.

13.2.1.5 WSDMSExportModuleClasses (Class Diagram)

This class diagram defines a WebServiceModule used for providing a web service interface for Exporting DMS data. It utilized the Chart WebService framework. The DMSExportHandler is the main class responsible for maintaining a cache of DMS related objects and providing methods to retrieve information in an exportable form. Note: the DMSExportHandler is not WebService specific and could be used in the context of the Chart ServiceApplication framework if needed.

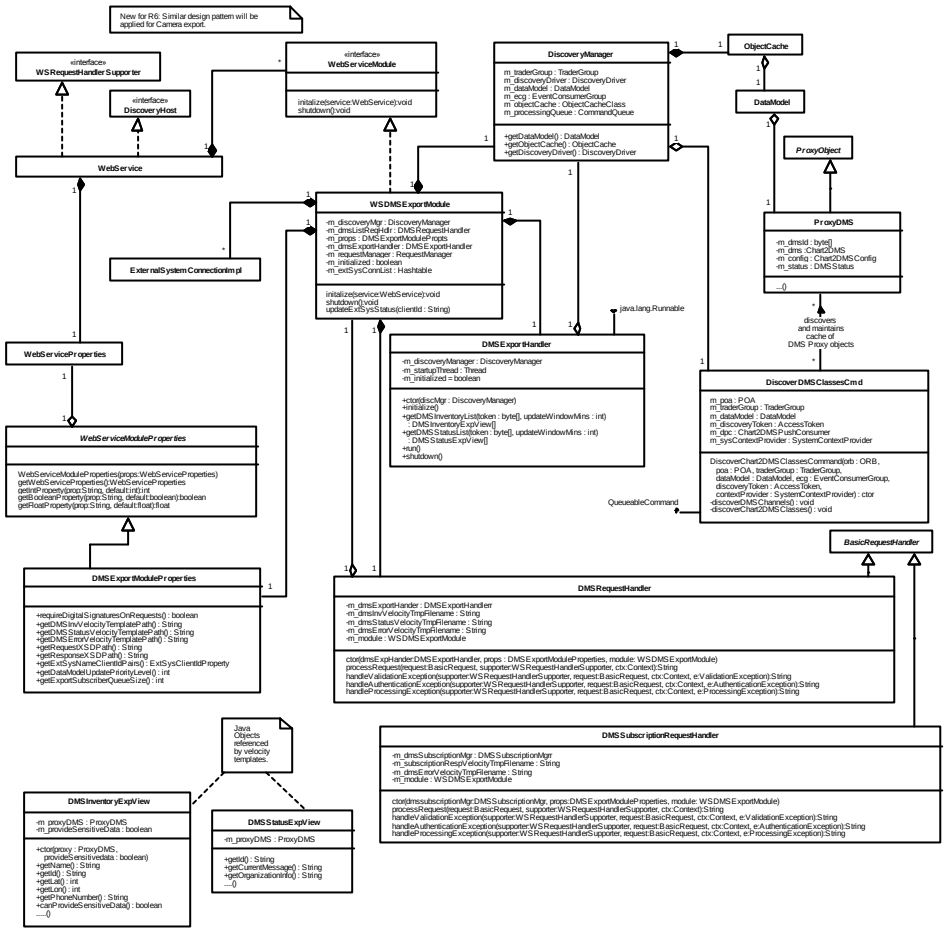


Figure 13-13 WSDMSExportModuleClasses (Class Diagram)

13.2.1.5.1 BasicRequestHandler (Class)

This abstract base class provides an implementation of the WSRequestHandler.processRequest() method that provides optional XML validation against specified XSD files and optional digital signature verification as well. It is intended to be used by request handlers that plan to take XML in and return XML to the calling client.

13.2.1.5.2 DataModel (Class)

The data model class serves as a collection of objects. It provides an efficient lookup mechanism for locating any object, and methods which allow for the retrieval of all objects of a particular type. Additionally, this class provides the ability to attach observer objects which are notified when objects are added to or removed from the model. Objects may also notify the DataModel that they have been modified. The model will periodically notify all attached observers of the changes to objects in the model.

13.2.1.5.3 DiscoverDMSClassesCmd (Class)

The DiscoverChart2DMSClassesCmd class is responsible for discovering Chart2DMS and Chart2DMSFactory corba objects, wrapping those objects in proxy classes and adding those objects to the DiscoveryManager's Object Cache. This class also listens to appropriate corba events and updates the Object cache accordingly.

13.2.1.5.4 DiscoveryHost (Class)

This interface defines the methods that the DiscoveryManager relies on. It must be implemented by any class that will create a DiscoveryManager.

13.2.1.5.5 DiscoveryManager (Class)

This SystemContextProvider interface defines some of the functionality required by a class which provides discovery services for CHART services. It is used by both the CHART GUI and the CHART backend services. A class which implements this interface must provide "get" accessor methods for the system profile properties, the data model, and the main processing queue for a service, for instance. It also provides access to the root deployment path and dynamic image path, which is used only by the CHART GUI. For the CHART GUI, this interface is known to be implemented by the MainServlet; for the back end CHART services, this interface is known to be implemented by the Discovery Manager.

13.2.1.5.6 DMSExportHandler (Class)

The DMSExportHandler class is responsible for maintaining up to date Chart DMS information in the ObjectCache. This data is used to support the class methods which provide data in response to web service requests for exporting DMS data.

13.2.1.5.7 DMSExportModuleProperties (Class)

The DMSExportModuleProperties class provides access methods for properties used by the WSDMSExportModule. It Extends the WebServiceModuleProperties class which allows access to other properties available from the WebService Framework.

13.2.1.5.8 DMSInventoryExpView (Class)

The DMSInventoryExpView class wraps a ProxyChart2DMS object and provides a view of the proxy object specific to DMS Inventory requests. These objects are used by the Velocity Engine which is made available by the WebService Framework. Velocity will apply a defined DMS Inventory velocity template to a collection of these objects to generate the XML response to DMS Inventory export requests.

13.2.1.5.9 DMSRequestHandler (Class)

The DMSRequestHandler extends the BasicRequestHandler abstract class and implements abstract methods used to handle web service requests for exporting DMS information.

13.2.1.5.10 DMSStatusExpView (Class)

The DMSStatusExpView class wraps a ProxyChart2DMS object and provides a view of the proxy object specific to DMS Status requests. These objects are used by the Velocity Engine which is made available by the WebService Framework. Velocity will apply a defined DMS Status velocity template to a collection of these objects to generate the XML response to DMS Status export requests.

13.2.1.5.11 DMSSubscriptionRequestHandler (Class)

The DMSSubscriptionRequestHandler extends the BasicRequestHandler and defines process required to handle DMS export Subscription requests made available by the Chart Export Web Service. Subscriptions allow clients to receive "real time" updates to DMSs as opposed to "on demand" updates which the client has to initiate.

13.2.1.5.12 ExternalSystemConnectionImpl (Class)

This class knows how to maintain the status of external connections and push them up to the GUI. Also, ExternalSystemConnectionAlerts and Notifications can be sent as configured by the admin.

13.2.1.5.13 ObjectCache (Class)

The ObjectCache is a wrapper for the DataModel. It provides access to DataModel methods to find objects in the data model, delegating those methods to the DataModel itself. It also provides additional methods of finding name filtered objects and discovering "duplicate" objects (as defined by an isDuplicateOf() method of the Duplicatable interface).

13.2.1.5.14 ProxyDMS (Class)

The ProxyChart2DMS object is a proxy for a Chart2DMS corba object which is used to by the DiscoveryManager / ObjectCache. The objects are used to maintain an up to date cache of Chart2DMS data in the object cache for application use.

13.2.1.5.15 ProxyObject (Class)

This class is a base class for many types of proxy objects store in the CHART object cache (which have been discovered by the DiscoveryDriver tasks), used to provide a standard set of access methods for the proxy objects.

13.2.1.5.16 WebService (Class)

This class is the core of each Web Service. It extends the VelocityServlet base class and implements the Service CORBA interface so that Web Service servlets can be administered in the same manner as other CHART service applications.

13.2.1.5.17WebServiceModule (Class)

This interface defines the methods that each module must implement in order to run within the web service framework.

13.2.1.5.18WebServiceModuleProperties (Class)

This abstract base class provides a base for WebServiceModule implementation classes to extend in order to get access to their configuration properties.

13.2.1.5.19WebServiceProperties (Class)

This class provides convenient access to the java Properties object that contains configuration data for the web service and its modules.

13.2.1.5.20WSDMSExportModule (Class)

The WSDMSExportModule implements the WebServiceModule interface and provides DMS export functionality via the WebService framework.

13.2.1.5.21WSRequestHandlerSupporter (Class)

This interface defines the methods that will be available to every WSRequestHandler when it is invoked by the framework. It defines the services that the framework will make available to the handlers.

13.2.2 Sequence Diagrams

13.2.2.1 DMSExportHandler:getDMSInventoryList (Sequence Diagram)

This diagram depicts the processing needed to retrieve DMS Inventory data in response to a specific DMS Inventory export request. If no ProxyDMSFactories are found in the data model this method throws a GeneralException. This in turn will trigger the WebService framework to call the handler's handleProcessingException() method. ProxyDMS objects are retrieved from the ObjectCache. Based on the optional update window parameter, the functional rights of the client specific token passed in and the owning organization of each proxy object, a collection of appropriate export data is created and returned to the caller.

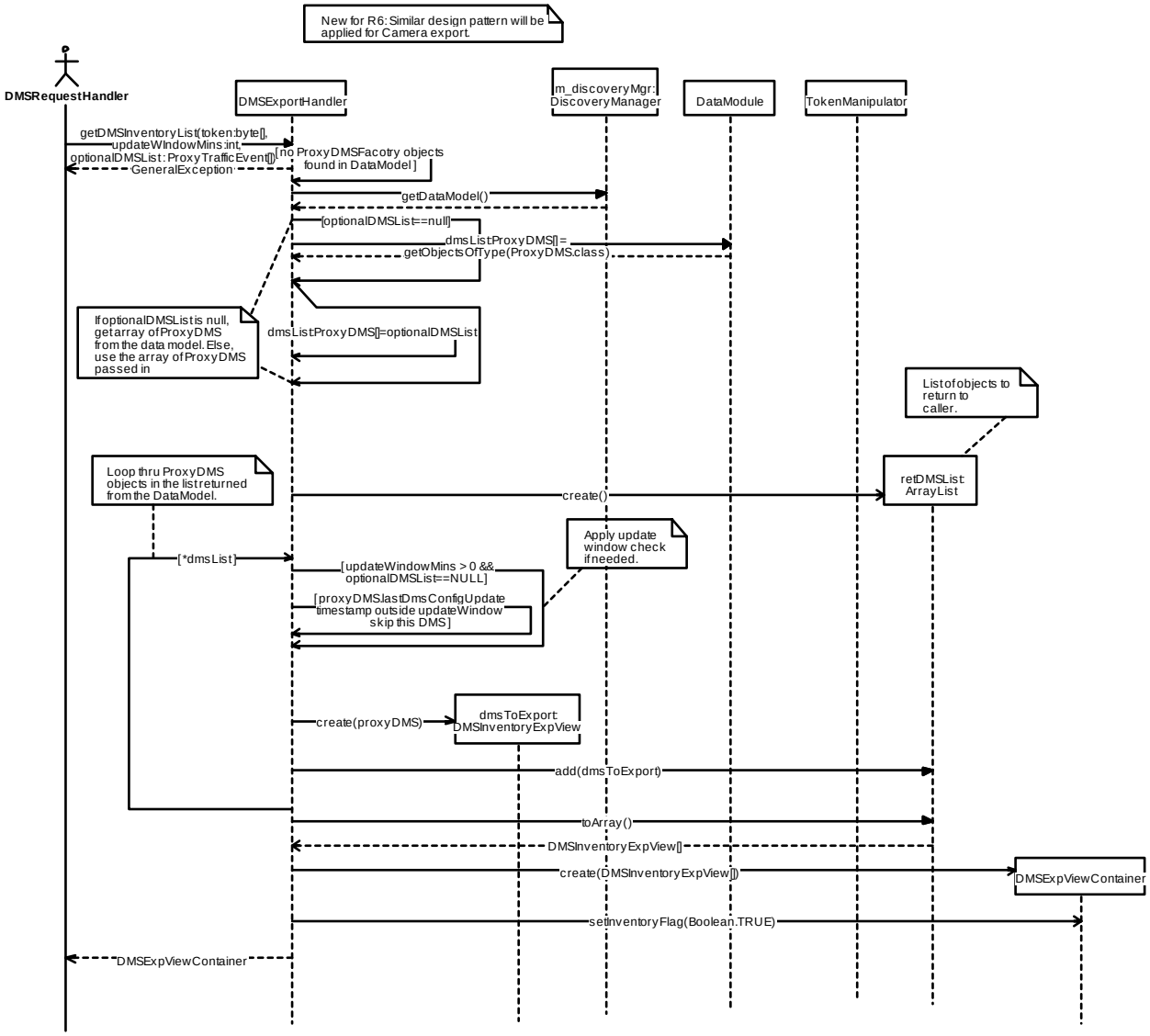


Figure 13-6 DMSExportHandler:getDMSInventoryList (Sequence Diagram)

13.2.2.2 DMSExportHandler:getDMSStatusList (Sequence Diagram)

This diagram depicts the processing needed to retrieve DMS Status data in response to a specific DMS Status export request. If no ProxyDMSFactories are found in the data model this method throws a GeneralException. This in turn will trigger the WebService framework to call the handler's handleProcessingException() method. ProxyDMS objects are retrieved from the ObjectCache. Based on the optional update window parameter, the functional rights of the client specific token passed in and the owning organization of each proxy object, a collection of appropriate export data is created and returned to the caller.

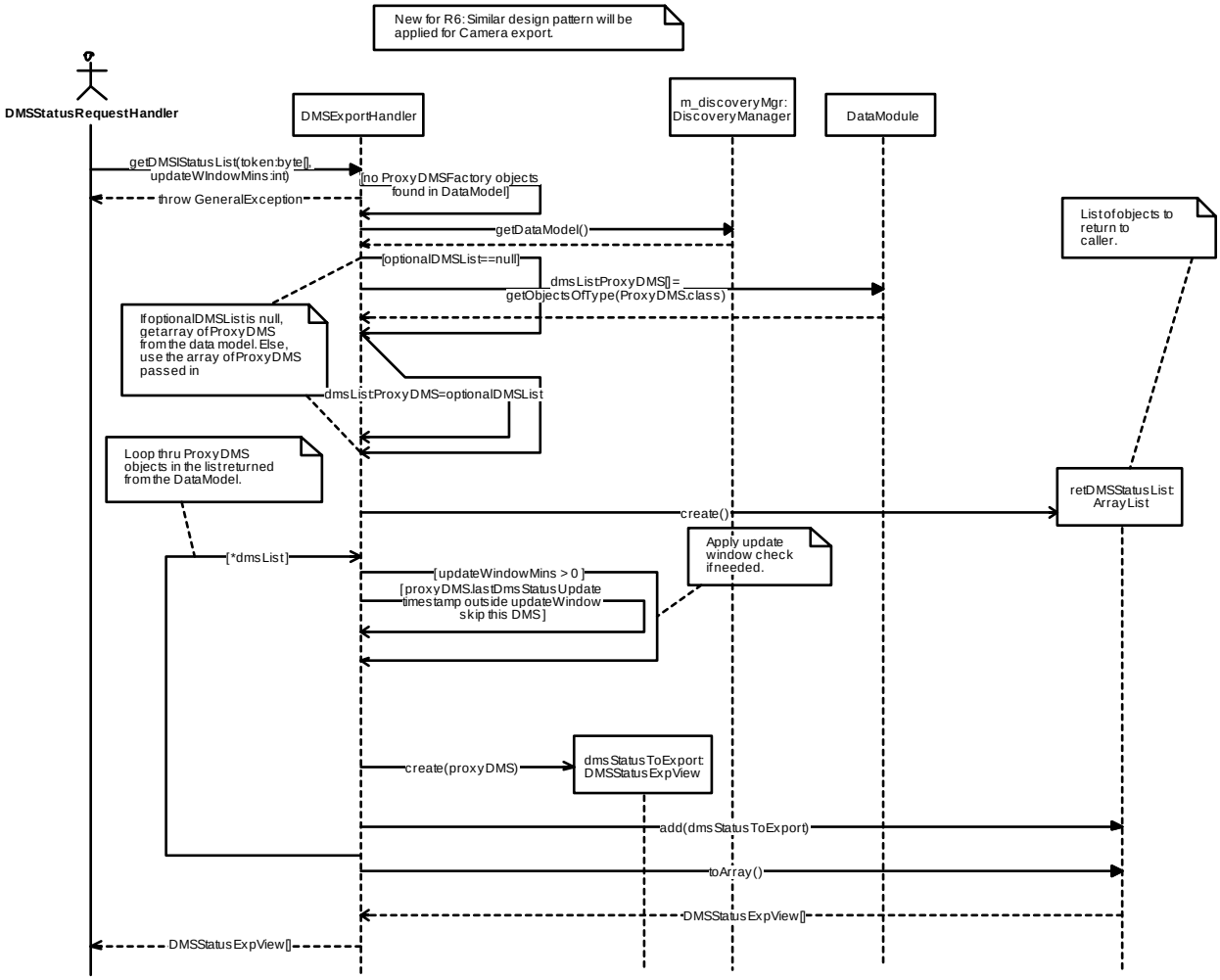


Figure 13-7 DMSExportHandler:getDMSStatusList (Sequence Diagram)

13.2.2.3 DMSExportHandler:initialize (Sequence Diagram)

This diagram depicts the initialization of the DMSExportHandler class. A DiscoveryChart2DMSClassesCmd is created and added to the DiscoveryManager to populate and maintain DMS data in the ObjectCache. As a best attempt to make sure the ObjectCache is populated before responding to dms web service export requests a thread is started to make sure Chart2DMSFactories exist in the ObjectCache before setting initialization flag to true.

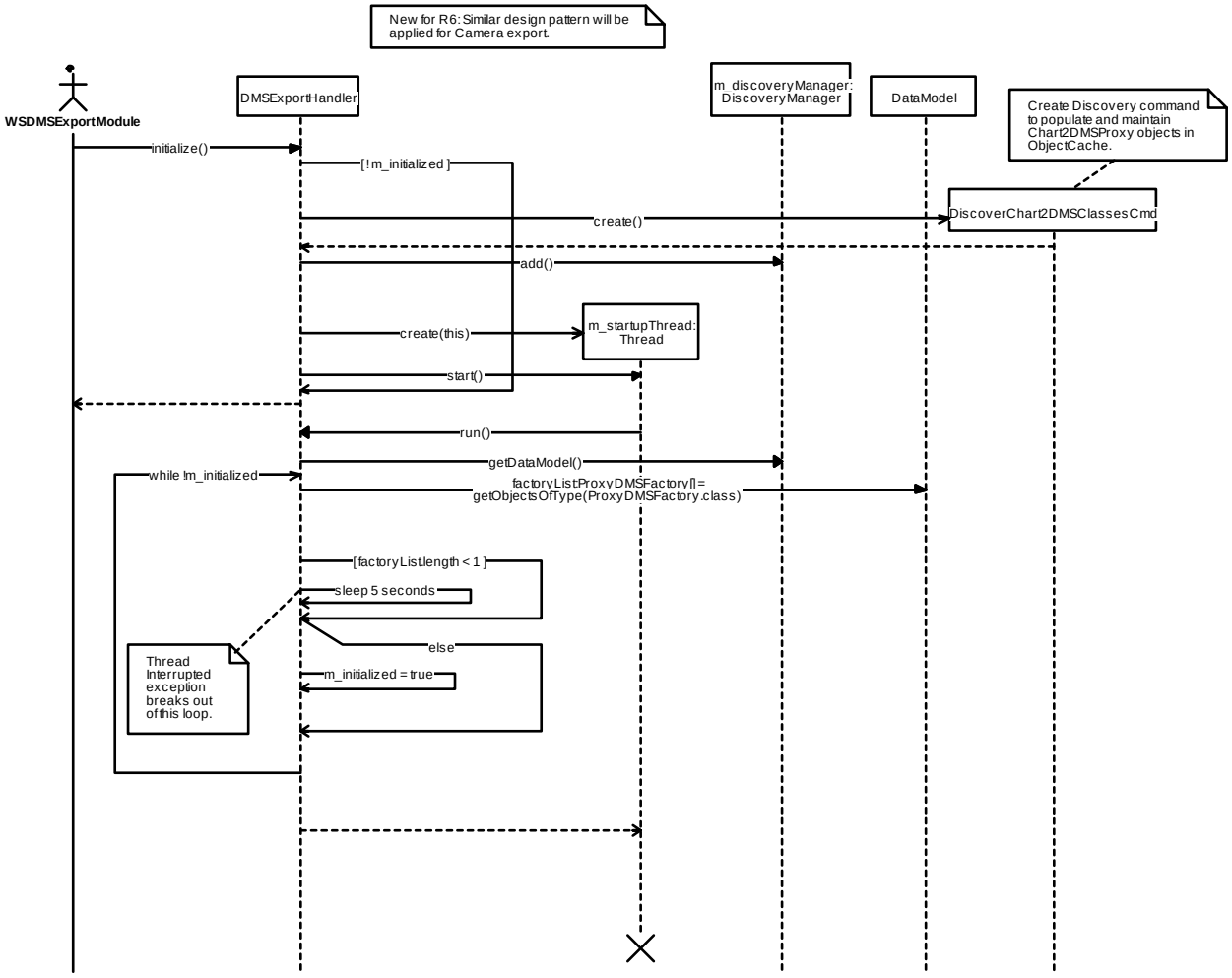


Figure 13-8 DMSExportHandler:initialize (Sequence Diagram)

13.2.2.4 DMSRequestHandler:handleExceptions (Sequence Diagram)

This sequence diagram depicts Exception handling done as part of the WebService framework. The DMSRequestHandler derives from the BasicRequestHandler class and implements 3 abstract methods (handleValidationException(), handleAuthenticationException() and handleProcessingException()). These methods are called from the WebService frame work when exceptions are encountered. Each method retrieves information as need from the arguments passed in and creates a response using a Velocity Context object and a predefine Velocity template. Note: for handleValidationException() if inbound xml validation fails, pass the invalid XML string back in the XML response defined as CDATA so it will not be parsed.

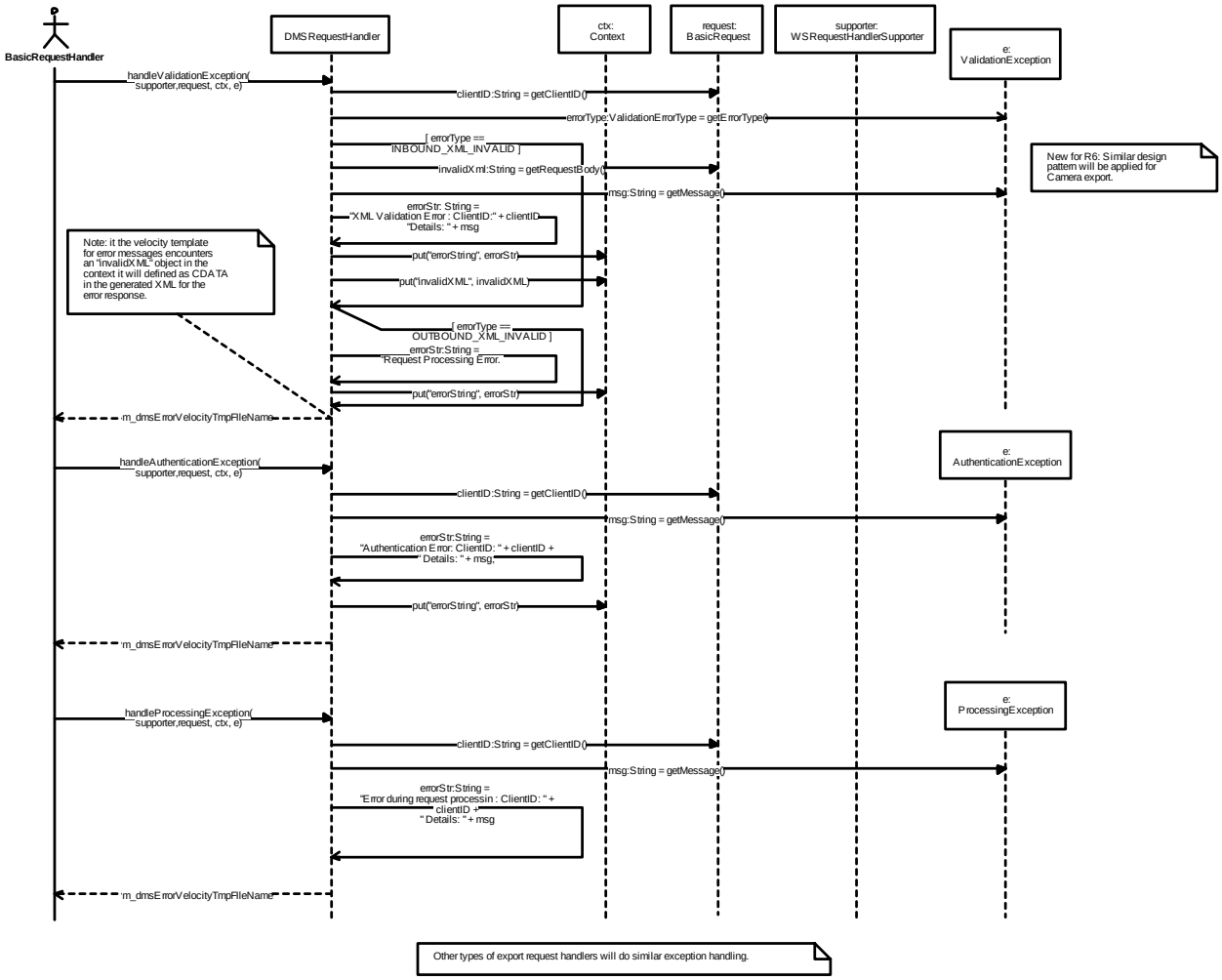


Figure 13-9 DMSRequestHandler:handleExceptions (Sequence Diagram)

13.2.2.5 DMSRequestHandler:processRequest (Sequence Diagram)

This diagram depicts the processing of DMS Export Requests for on demand updates and realtime updates (Subscriptions). The processRequest() method of the DMSequestHandler is called by the WebService RequestManger. An "on demand" request for dms data is handled by getting the appropriate data to export from the DMSExportHandler, adding that data to the Velocity Context passed in and finally returning the path to the correct Velocity Template for the request. A request for realtime updates of DMS Data (Subscriptions) is handled by call the appropriate method of the DMSSubscriptionManger for new/renewed subscriptions or canceled subscriptions. In both cases the appropriate information is loaded into the velocity context and the correct velocity template file path is returned to the caller.

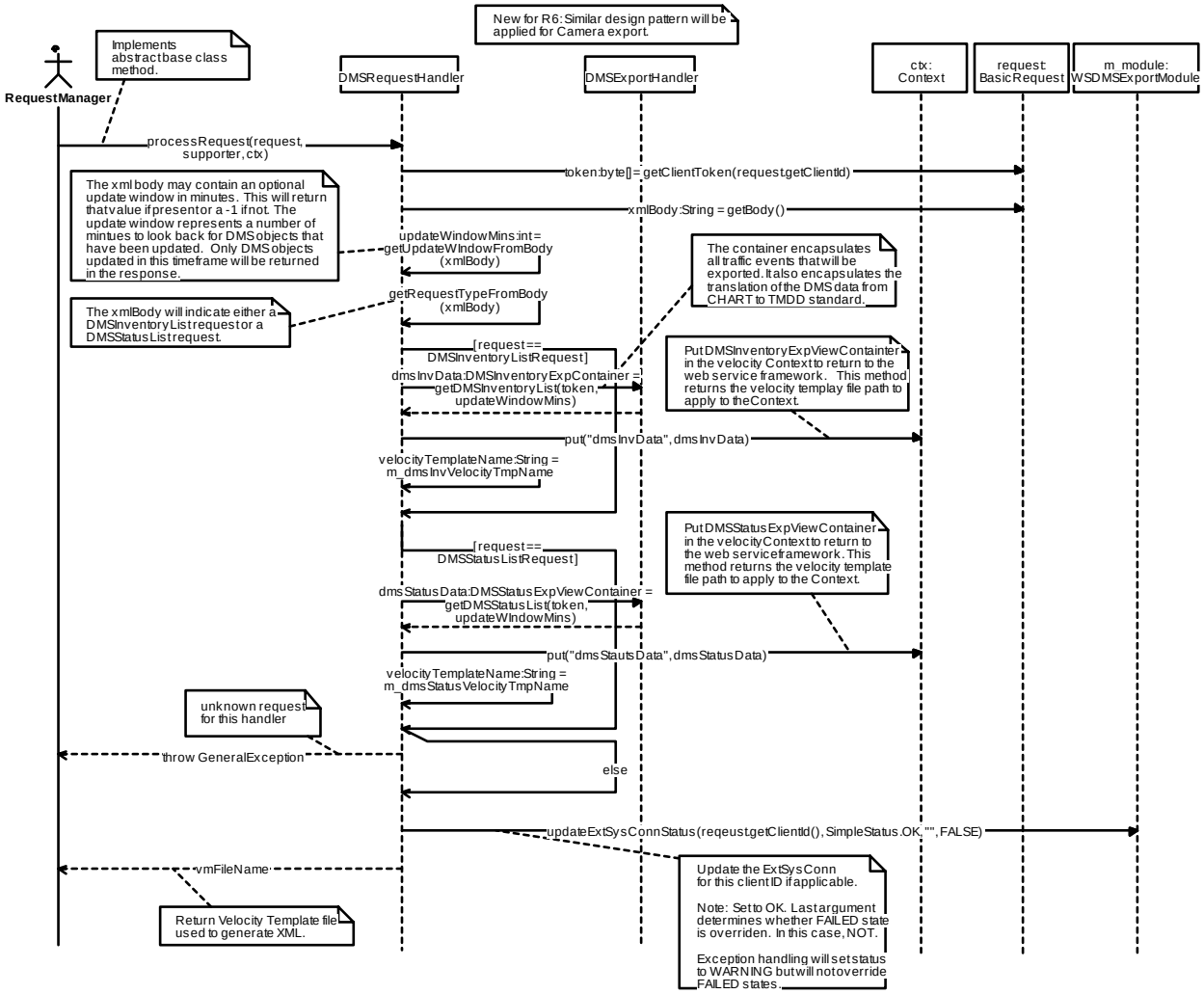


Figure 13-10 DMSRequestHandler:processRequest (Sequence Diagram)

13.2.2.6 DMSSubscription:doPush (Sequence Diagram)

This method is used by each subscription to push dms updates to the subscribers URL. Data is retrieved from the DMSExportHandler for the updated DMSs. Data is loaded into a VelocityContext and XML is generated for the DMS by merging the data with the appropriate velocity template. The XML data is then passed to the sendDataToSubscriber method to actually send the message to the remote service.

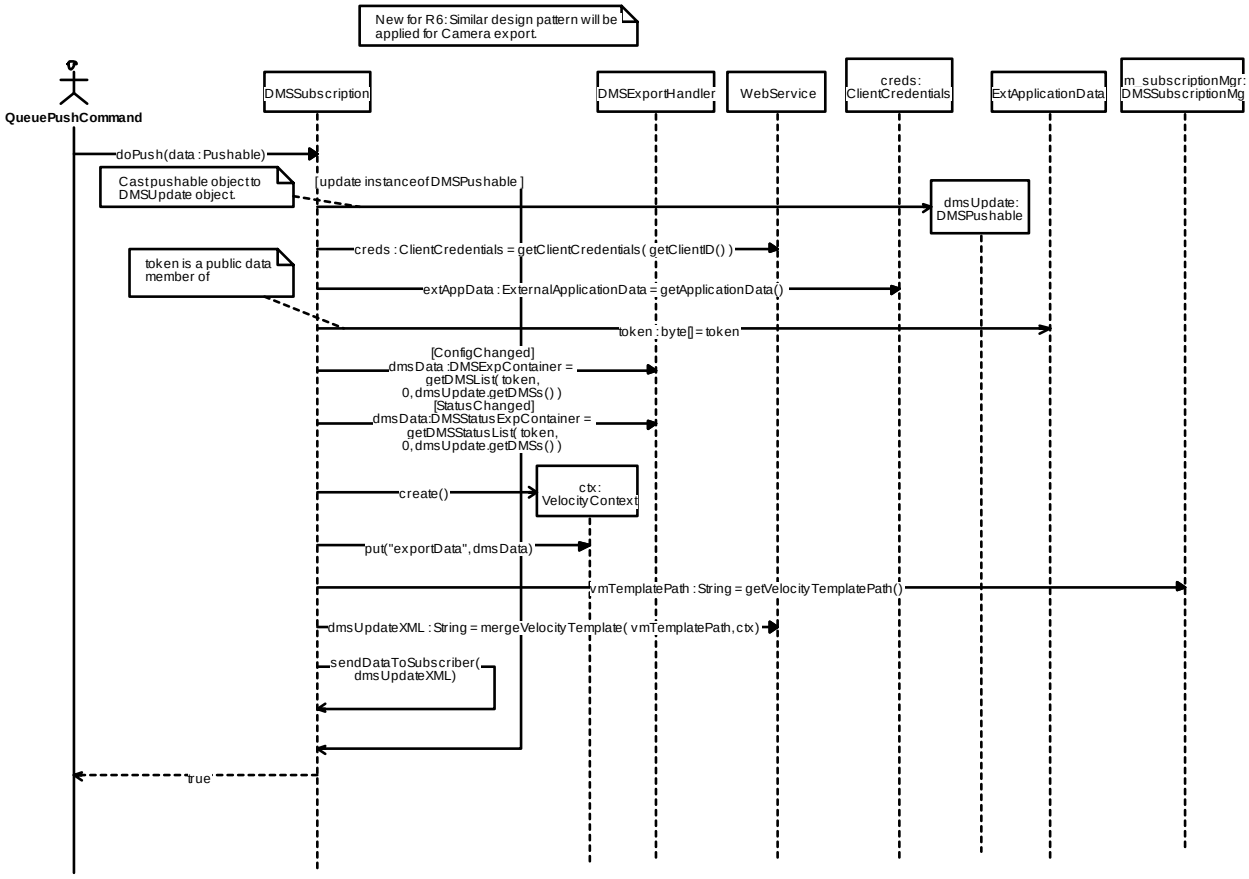


Figure 13-11 DMSSubscription:doPush (Sequence Diagram)

13.2.2.7 DMSSubscriptionMgr:creation (Sequence Diagram)

This diagram depicts the class constructor of the ExportSubscriptionManager abstract class. The PushFramework is used by the class to provide the mechanism used push data to external clients of the exporter. An ExportSubscriptionDB class is created to handle export subscription of specific data type (DMS, TSS, TrafficEvents, etc...). A timer is then created to handle automatic expiration of subscriptions.

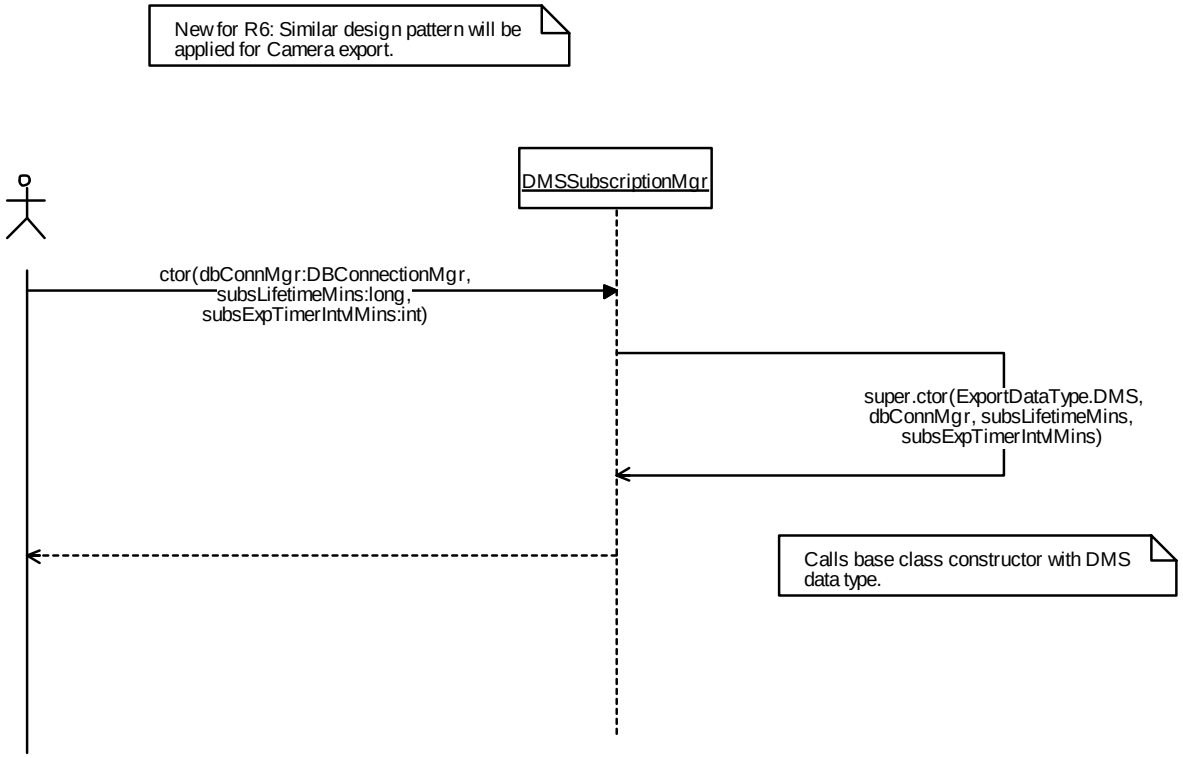


Figure 13-12 DMSSubscriptionMgr:creation (Sequence Diagram)

13.2.2.8 DMSSubscriptionMgr:initialize (Sequence Diagram)

This diagram depicts the initialization of the DMSSubscriptionMgr. Previously created subscriptions are read from the DB and used to create DMSSubscription objects. These objects implement the Pusher interface and are added as Pushers to the PushEngine.

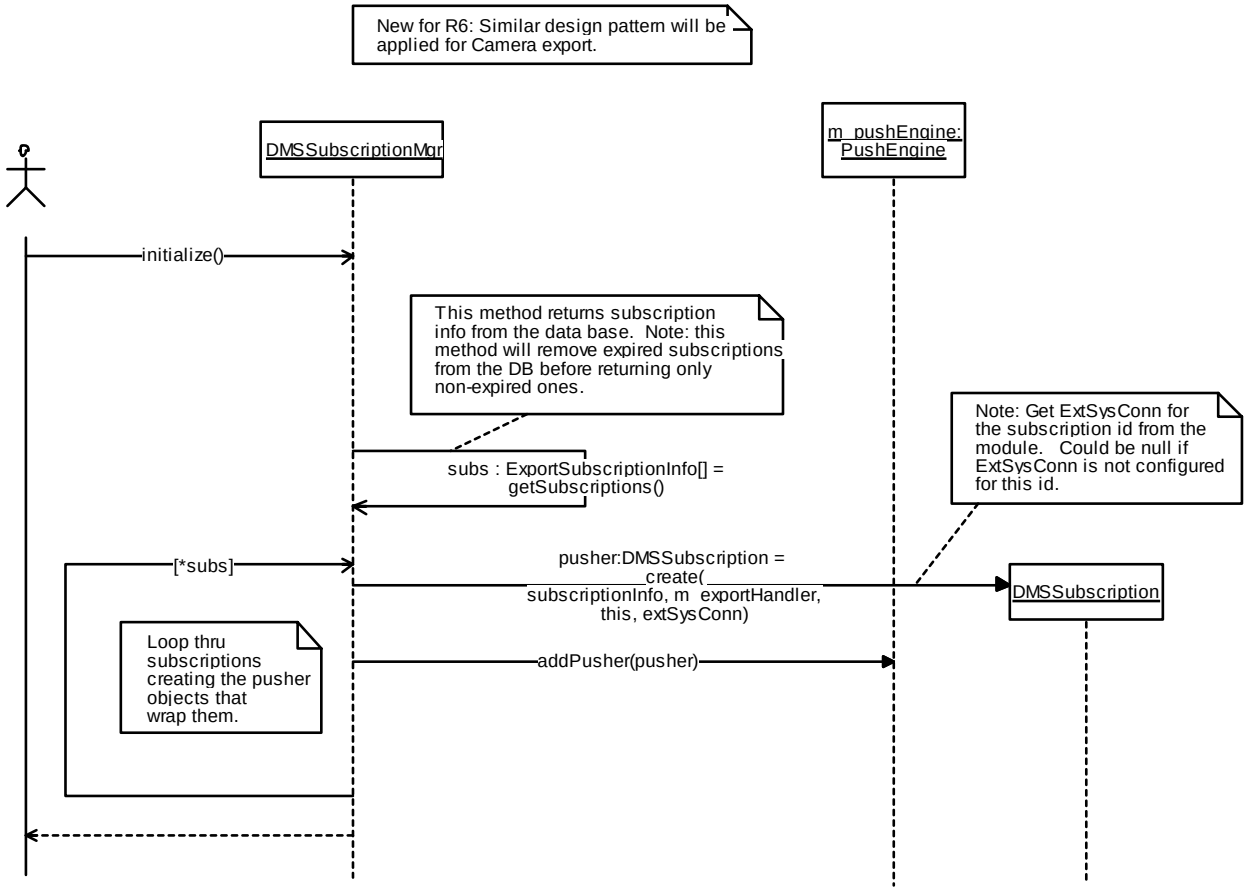


Figure 13-13 DMSSubscriptionMgr:initialize (Sequence Diagram)

13.2.2.9 DMSSubscriptionMgr:modelObserverUpdate (Sequence Diagram)

This method, part of the ModelObserver interface, is called when the DataModel needs to make observers aware of DMS updates, given the observers update interval. A DMSUpdate object is created. Objects of this type implement the Pushable interface. The update object is passed to the sendDataToSubscriber() method which used the PushFramework to push the update to all current Pushers (i.e. Subscribers).

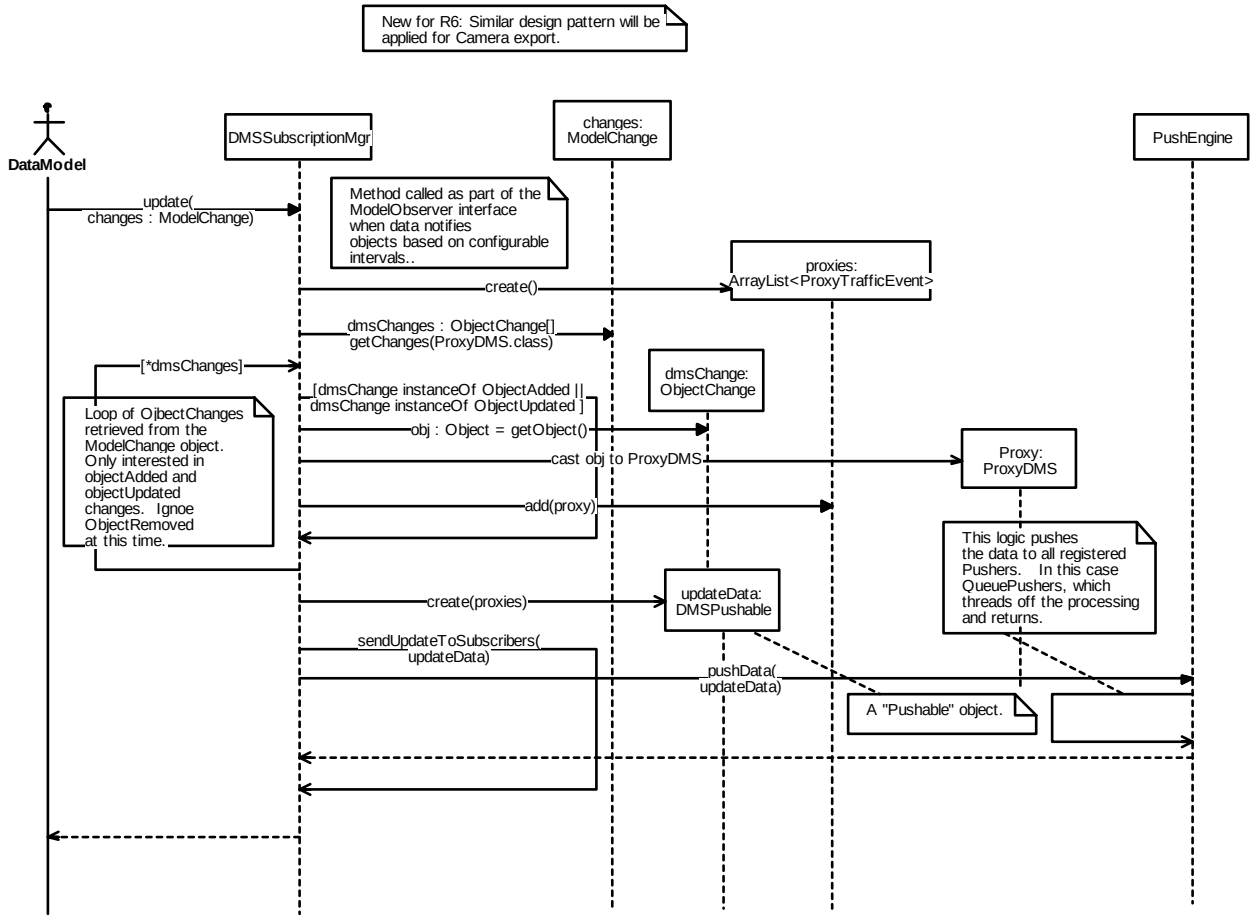


Figure 13-14 DMSSubscriptionMgr:modelObserverUpdate (Sequence Diagram)

13.2.2.10DMSSubscriptionMgr:updateSubscription (Sequence Diagram)

This method is used to update or add subscriptions to the DMSSubscriptionManager.

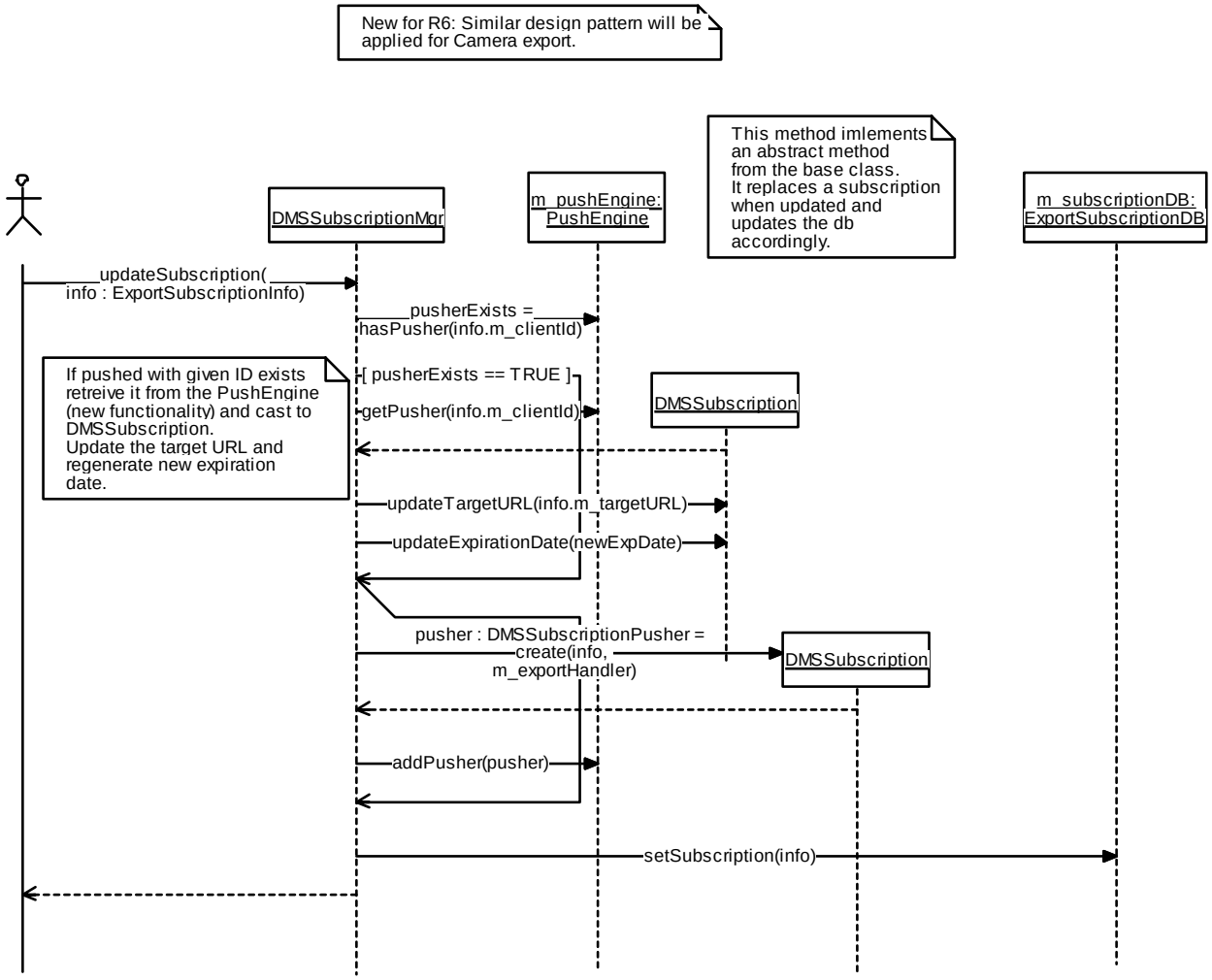


Figure 13-15 DMSSubscriptionMgr:updateSubscription (Sequence Diagram)

13.2.2.11DMSSubscriptionRequestHandler:processRequest (Sequence Diagram)

This diagram depicts the processing of DMS subscriptions requests.

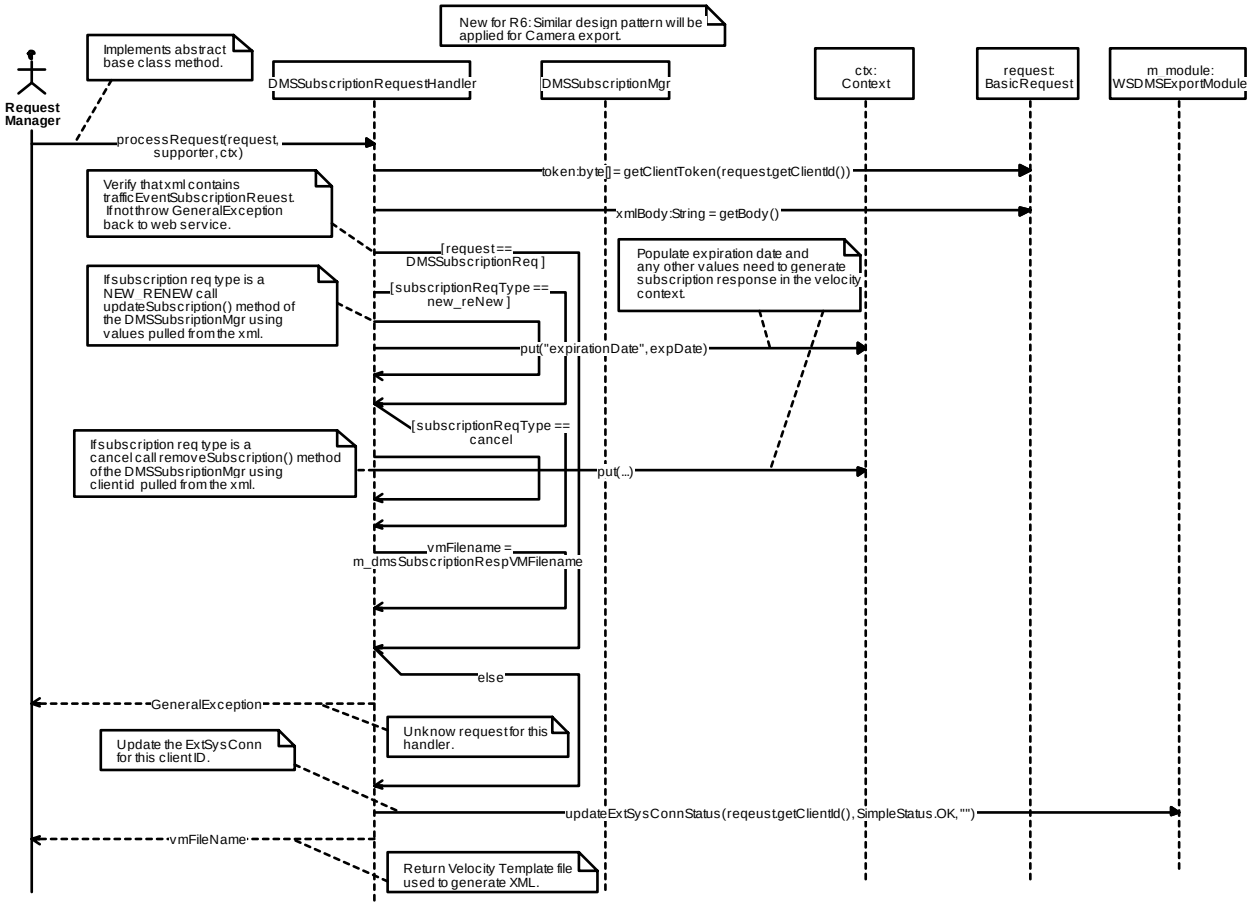


Figure 13-16 DMSSubscriptionRequestHandler:processRequest (Sequence Diagram)

13.2.2.12WSDMSExportModule:initialize (Sequence Diagram)

This diagram depicts the initialization of the WSDMSExportModule class. A module properties class is created followed by the creation of the DMSExportHander class. This class is responsible for maintaining DMS data in the ObjectCache and providing methods to retrieve dms export data in response to dms export web service requests as well as supporting subscription functionality. Next the DMSReqeustHandler and TrafficEventReqeustSubscriptionHandlers are created and registered with the WebService framework to allow the web service URL to start responding to requests.

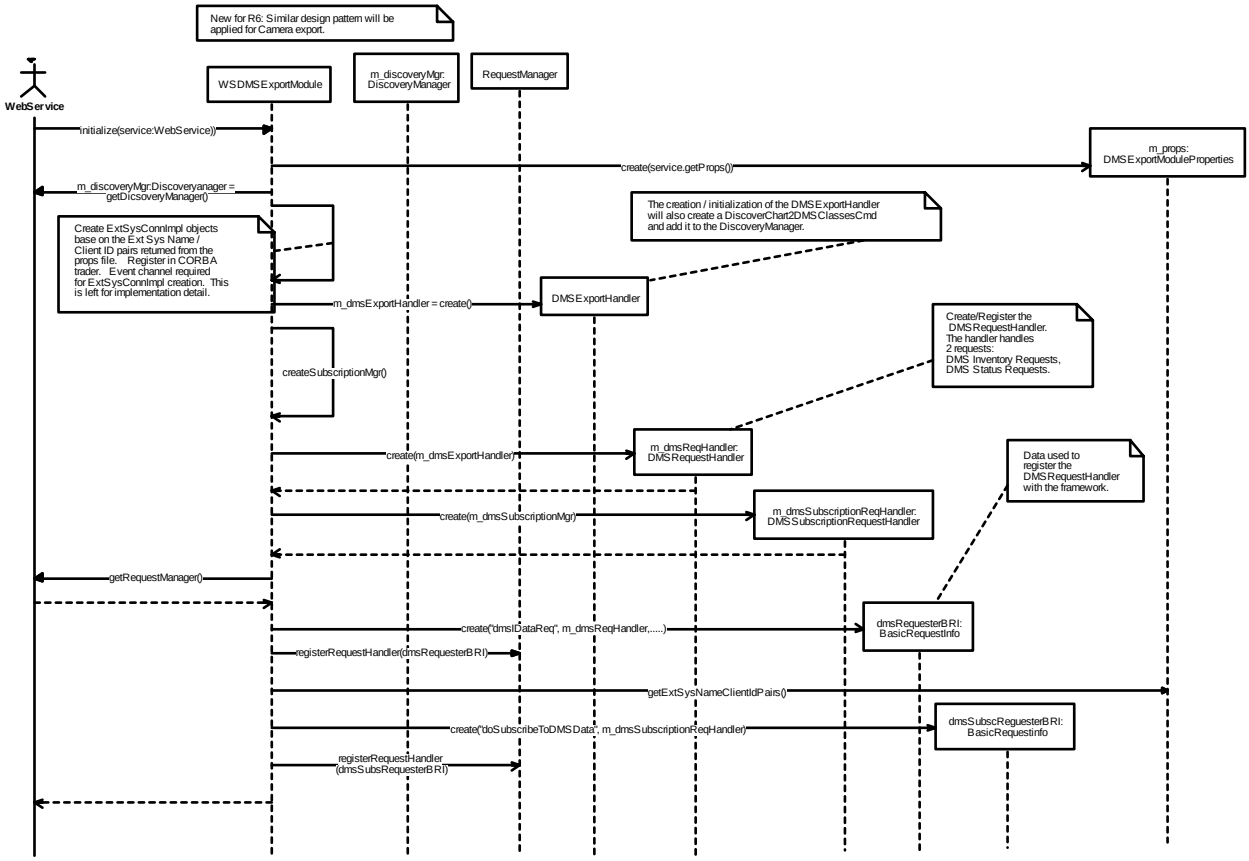


Figure 13-17 WSDMSExportModule:initialize (Sequence Diagram)

13.2.2.13WSDMSExportModule:shutdown (Sequence Diagram)

The diagram depicts shutdown processing for the WSDMSExportModule. Currently shutdown of the module initiates the DMSExportHandler.shutdown() which cleans up the startupThread if still running. Other cleanup may be determined during implementation.

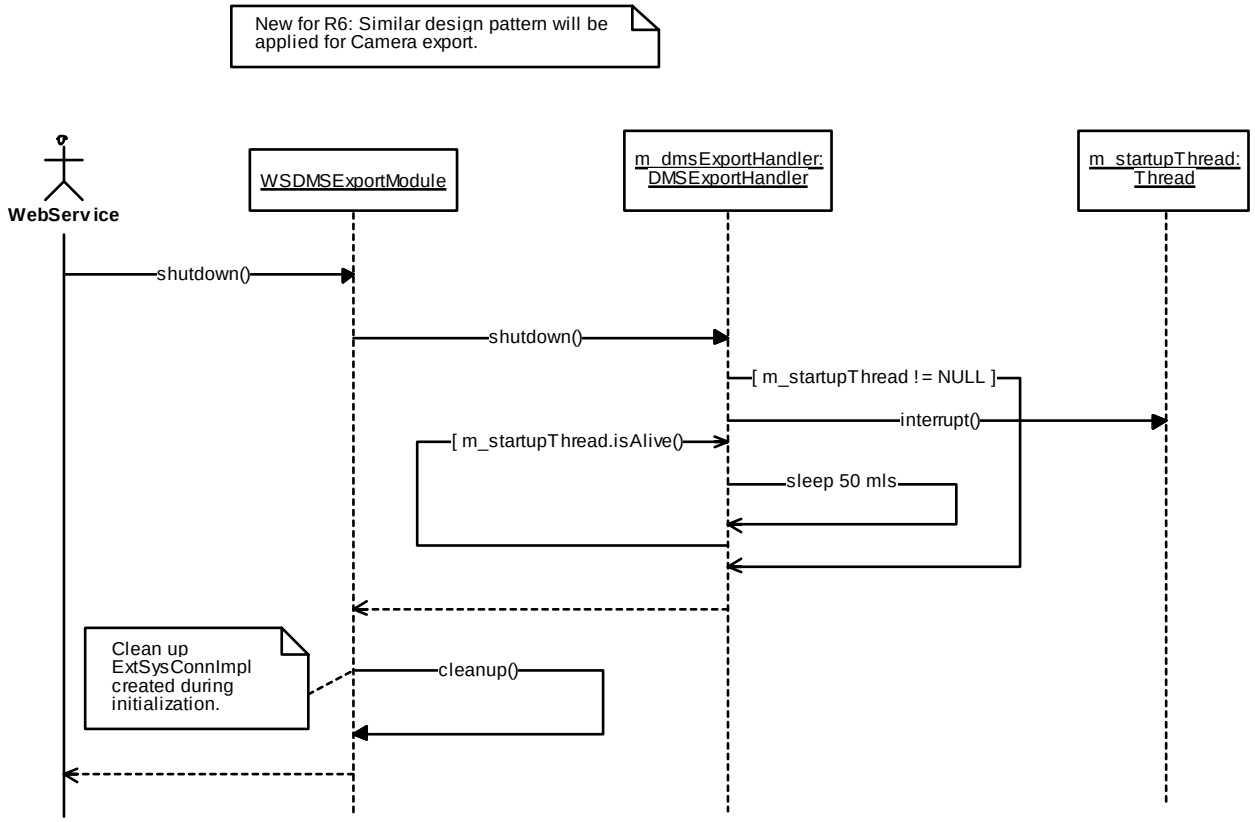


Figure 13-18 WSDMSExportModule:shutdown (Sequence Diagram)

13.3 Data Exporter Client

13.3.1 Class Diagrams

13.3.1.1 ExportListenerModuleClasses (Class Diagram)

This class diagram shows classes that comprise the Data Exporter Client Service.

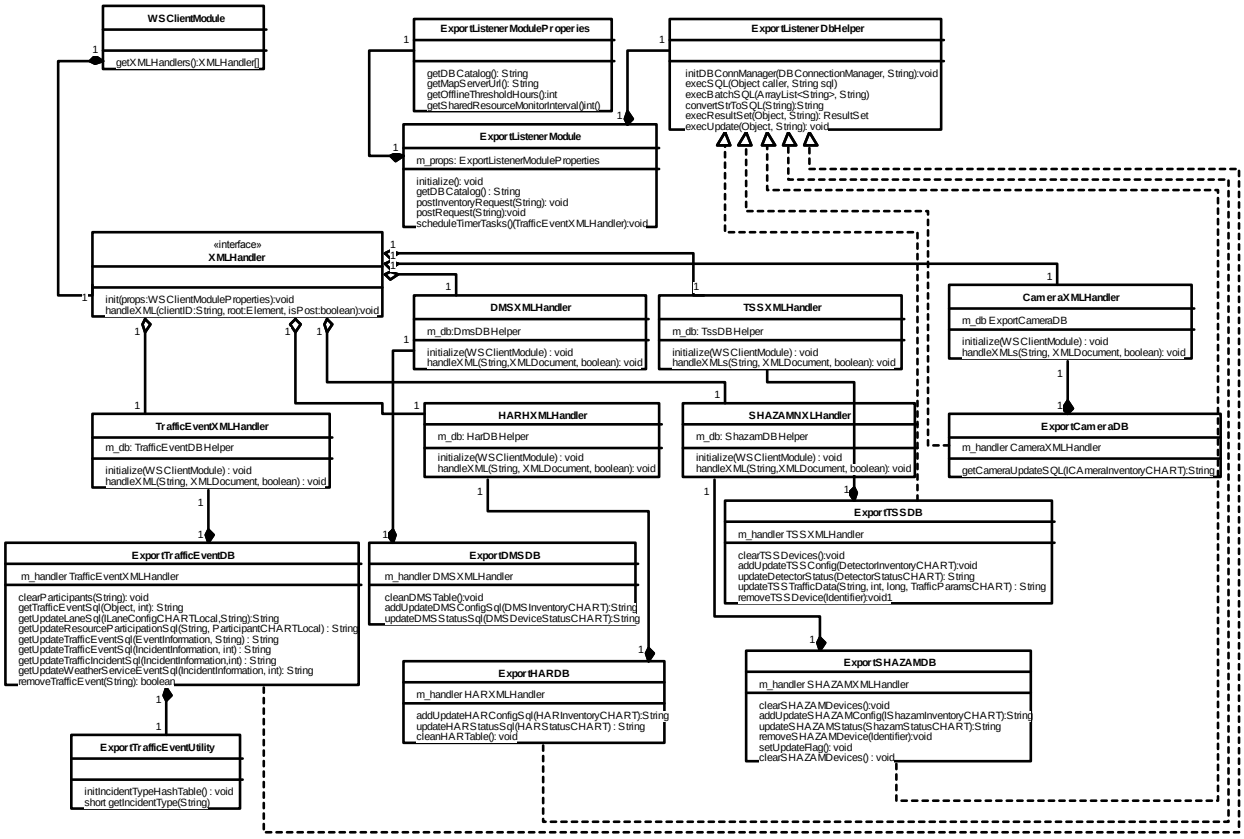


Figure 13-19 ExportListenerModuleClasses (Class Diagram)

13.3.1.1.1 CameraXMLHandler (Class)

This class is responsible for parsing CCTV XML data received from the CHART data exporter and calling ExportCameraDB to store it in the CHART Web database.

13.3.1.1.2 DMSXMLHandler (Class)

This class is responsible for handling DMS XML data received from the CHART data exporter.

13.3.1.1.3 ExportCameraDB (Class)

This class stores CCTV data received from the CHART data exporter into the CHART Web database.

13.3.1.1.4 ExportDMSDB (Class)

This class provides access to DMS related data that is stored in the database.

13.3.1.1.5 ExportHARDB (Class)

This class provides access to HAR related data that is stored in the database.

13.3.1.1.6 ExportListenerDbHelper (Class)

An abstract class that provides updates to the CHARTWeb database.

13.3.1.1.7 ExportListenerModule (Class)

Module that is responsible for setting up the classes that will handle XML pulled from the CHART data exporter and handling XML data updates posted by the exporter.

13.3.1.1.8 ExportListenerModuleProperies (Class)

This Class provides access to properties for the Export Listener web module.

13.3.1.1.9 ExportSHAZAMDB (Class)

This class provides access to SHAZAM related data that is stored in the database.

13.3.1.1.10 ExportTrafficEventDB (Class)

This class provides access to traffic event related data that is stored in the database.

13.3.1.1.11 ExportTrafficEventUtility (Class)

A utility class used to translate Incident types.

13.3.1.1.12 ExportTSSDB (Class)

This class provides access to TSS related data that is stored in the database.

13.3.1.1.13 HARHXMLHandler (Class)

This class is responsible for handling HAR XML data received from the CHART data exporter.

13.3.1.1.14 SHAZAMNXLHandler (Class)

This class is responsible for handling Shazam XML data received from the CHART data exporter.

13.3.1.1.15 TrafficEventXMLHandler (Class)

This class is responsible for handling traffic event XML data received from the CHART data exporter.

13.3.1.1.16 TSSXMLHandler (Class)

This class is responsible for handling TSS XML data received from the CHART data exporter.

13.3.1.1.17WSClientModule (Class)

A WebServiceModule that can be configured to handle XML data posted to it and post XML/HTTP requests to remote services.

13.3.1.1.18XMLHandler (Class)

The XMLHandler interface should be implemented by classes that will handle XML data posted to this service or returned from a remote service in response to a request posted to it from this service.

13.3.2 Sequence Diagrams

13.3.2.1 ExportListener:ProcessCameraInventory (Sequence Diagram)

This diagram shows the processing performed when the RequestLooper is ready to process posted XML with camera configuration. The posted XML is passed to CamearXMLHandler. HandleXMl method create a CameraProcessXMLCmd and put in Command Queue. Then for each Camera config system will call isMessageLatestDate().This method compare two date(from database and from xml message) and return false if update time from database latest then messageUpdateTime, true otherwise. If method return true call getCameraUpdateSql(). Method return array of camera update sql. After all Camera XML message processed call execBatchSQL().This method executes the specified batch of statements on the specified connection.

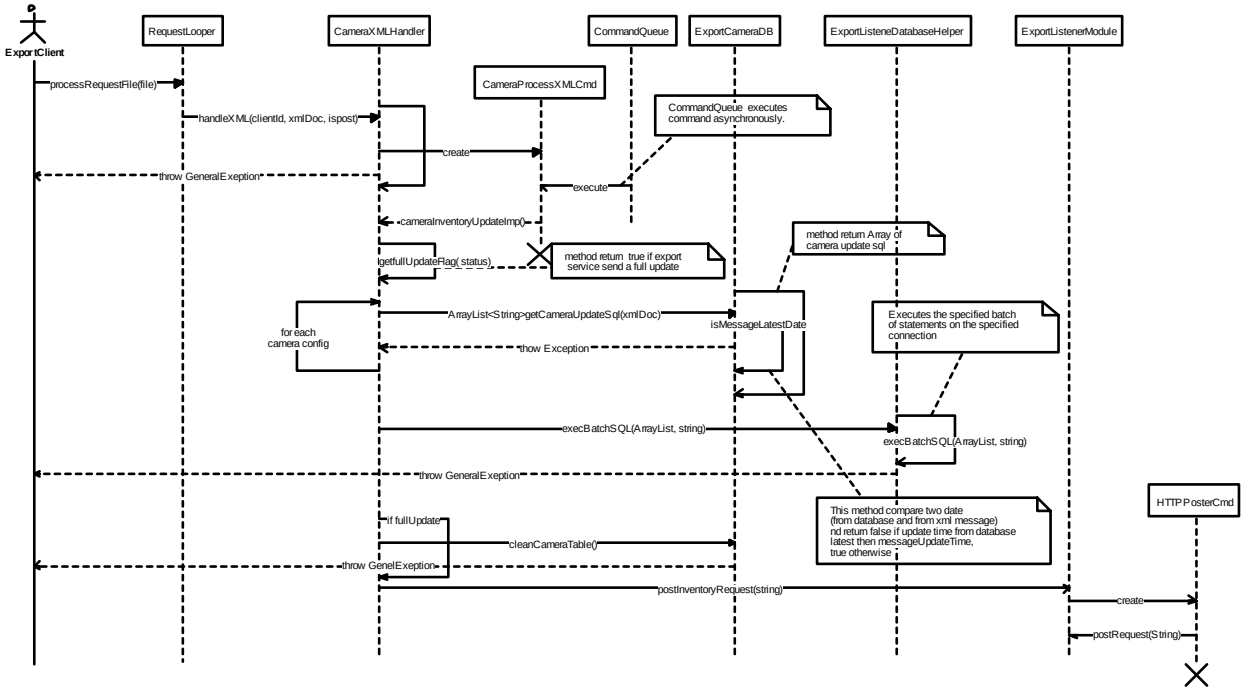


Figure 13-21 ExportListener:ProcessCameraInventory (Sequence Diagram)

13.3.2.2 ExportListener:ProcessCameraStatus (Sequence Diagram)

This diagram shows the processing performed when the RequestLooper is ready to process posted XML with camera status. The posted XML is passed to CamearXMLHandler. HandleXMl method create a CameraProcessXMLCmd and put in Command Queue.Then for each Camera status system will call isMessageLatestDate().This method compare two date(from database and from xml message) and return false if update time from database latest then messageUpdateTime, true otherwise. If method return true call getCameraStatusSql(). Method return array of camera update sql. After all Camera XML message processed call execBatchSQL().This method executes the specified batch of statements on the specified connection.

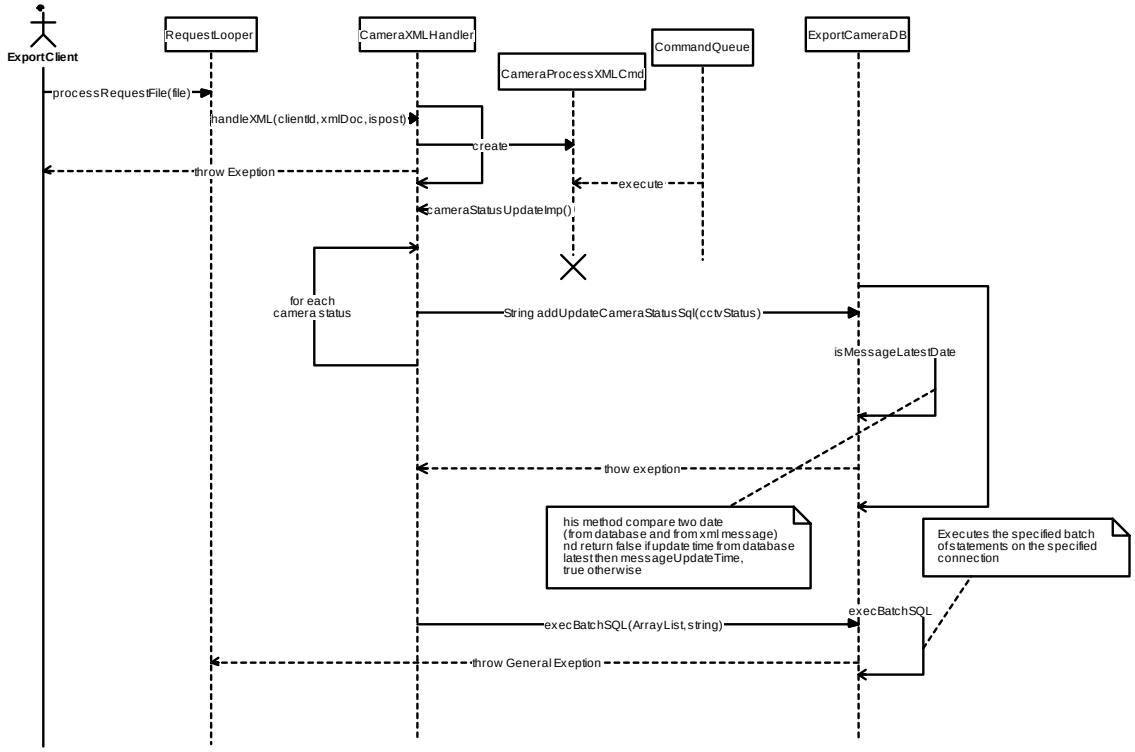


Figure 13-22 ExportListener:ProcessCameraStatus (Sequence Diagram)

13.4 Mapping Synchronization Application

13.4.1 Class Diagrams

13.4.1.1 Data Exporter Synchronization Intranet Map (Class Diagram)

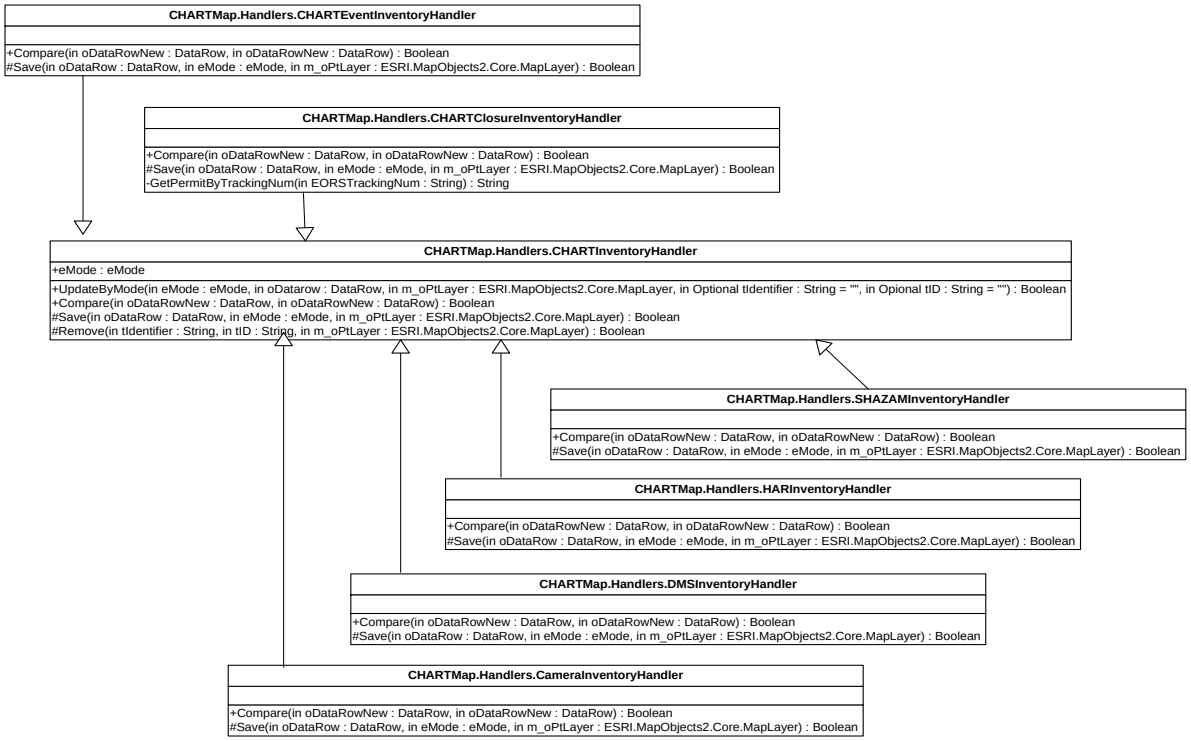


Figure 13-23 Intranet Map Exporter Synchronization (Class Diagram)

13.4.1.1.1 CHARTMap.Handlers.CHARTInventoryHandler (Class)

This is a base class for inventory updates. This class contains methods to determine the update status (add, update, or remove). In addition, this class contains a method to remove an object from the spatial table. Derived classes must implement Compare() and Save().

13.4.1.1.2 CHARTMap.Handlers.DMSInventoryHandler (Class)

This class extends the CHARTInventoryHandler class and implements the Compare() and Save() methods. This class is used to compare DMS records between the spatial and non-spatial table. This class is also used to update or add new record(s) to the DMS spatial table.

13.4.1.1.3 CHARTMap.Handlers.HARInventoryHandler (Class)

This class extends the CHARTInventoryHandler class and implements the Compare() and Save() methods. This class is used to compare HAR records between the spatial and non-spatial table. This class is also used to update or add new record(s) to the HAR spatial table.

13.4.1.1.4 CHARTMap.Handlers.SHAZAMInventoryHandler (Class)

This class extends the CHARTInventoryHandler class and implements the Compare() and Save() methods. This class is used to compare SHAZAM records between the spatial and non-spatial table. This class is also used to update or add new record(s) to the SHAZAM spatial table.

13.4.1.1.5 CHARTMap.Handlers.CHARTEventInventoryHandler (Class)

This class extends the CHARTInventoryHandler class and implements the Compare() and Save() methods. This class is used to compare CHART Event records between the spatial and non-spatial table. This class is also used to update or add new record(s) to the CHART Event spatial table.

13.4.1.1.6 CHARTMap.Handlers.CHARTClosureInventoryHandler (Class)

This class extends the CHARTInventoryHandler class and implements the Compare() and Save() methods. This class is used to compare CHART Closure records between the spatial and non-spatial table. This class is also used to update or add new record(s) to the CHART Closure spatial table.

13.4.1.1.7 CHARTMap.Handlers.CameraInventoryHandler (Class)

This class extends the CHARTInventoryHandler class and implements the Compare() and Save() methods. This class is used to compare Camera records between the spatial and non-spatial table. This class is also used to update or add new record(s) to the Camera spatial table.

13.4.2 Sequence Diagrams

13.4.2.1 CHART Data Exporter Synchronization – (Sequence Diagram)

This diagram shows the processing that occurs when a request to synchronize CHART Events or Devices. The process starts by parsing the incoming query string and determines which object to be synchronized. The process also writes each process events into the log file in the local machine. An associated object is created once the process determines which object to be synchronized. Then it calls the UpdateInventory method to start comparing each column of the spatial and the non-spatial tables based on the distinct identifier. The existing spatial record is updated by calling the UpdateByMode method if the process finds a difference between the two tables. As an exception, a record will be removed by calling the UpdateByMode method if the non-spatial table contains 0 or Null value for both Latitude and Longitude columns which indicates the Events or Device has been un-mapped from the integrated map. If a record existed in the non-spatial table but it does not exist in the spatial table, the process will then add a new record by calling the UpdateByMode method to the spatial table unless the value of the Latitude and Longitude columns are either 0 or Null. The process will remove the record from the spatial table if the record by calling the UpdateByMode method only existed in the spatial table

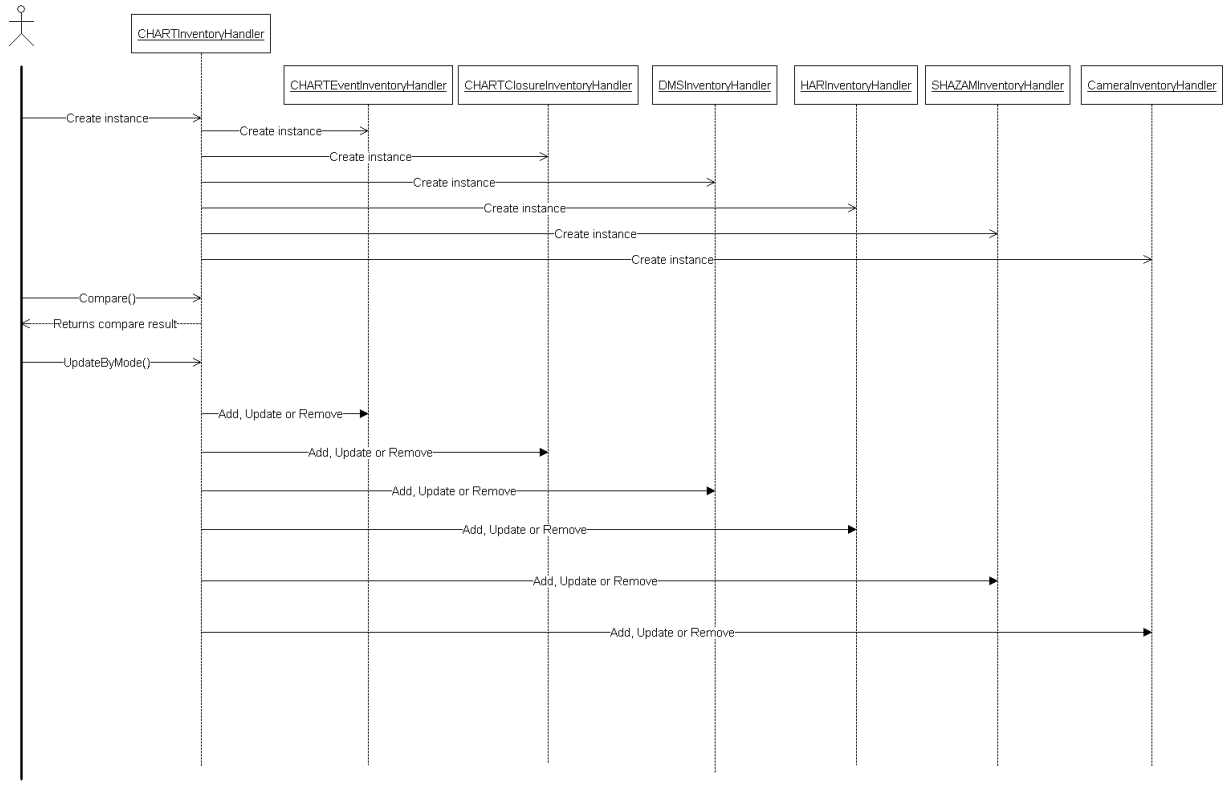


Figure 13-24 Intranet Map Exporter Synchronization (Sequence Diagram)

14 Deprecated Functionalities

The following functions have been deprecated due to CHART R6 / Mapping R5 changes.

14.1.1 EORS V2

14.1.1.1 Camera Hacking Page

In R6, Camera location information is being imported by the Exporter Client to replace the camera creation by the camera hacking page.

14.1.2 CHART Device Editor

14.1.2.1 View, Add, Update, Remove CHART Devices

In R6, the Integrated Map in the CHART application will start handling the mapping of CHART Devices (DMS, HAR, SHAZAM and CAMERA); Device viewing, adding, updating and removing will no longer be available in the CHART Device Editor.

15 Mapping To Requirements

The following table shows how the requirements in the CHART R6 Requirements document map to design elements contained in this design.

Tag	Text	Feature	Use Cases	Other Design Elements
SR1	ADMINISTER SYSTEMS AND EQUIPMENT		N/A	
SR1.1	ADMINISTER CHART ORGANIZATIONS, LOCATIONS, AND USERS		N/A	
SR1.1.8	Manage Object Locations		N/A	
SR1.1.8.1	Specify Object Location		N/A	
SR1.1.8.1.1	Specify Object Location Using Form	Map	N/A	
SR1.1.8.1.1.8	The system shall allow the user to specify that the object is located between two features on the route.	Map	Specify Intersecting Feature Pair	FlexLocationClasses CD GUIFlexComponentClasses CD flex.shared.components:SpecifyLocation.updateIntFeatures SD UtilityPkg.ObjectCache.LocationRelatedProxyObjects CD UtilityPkg.UtilityClasses2 CD
SR1.1.8.1.1.8.1	The system shall allow the user to specify a feature on the route marking the beginning of the interval containing the object, as described in the Specify Feature On Route requirements.	Map	Select First Intersecting Feature	FlexLocationClasses CD GUIFlexComponentClasses CD flex.shared.components:SpecifyLocation.updateIntFeatures SD Screenshot: HMI Figure 4-4
SR1.1.8.1.1.8.1.1	The system shall automatically populate the geographic location of the object using the beginning of the interval if an intersection with another route is specified and the geographic location of the intersection is available from the Spatial Web Service.	Map	Select First Intersecting Feature	chartlite.data.location-data.GUILocationDataClasses CD FlexLocationClasses CD

Tag	Text	Feature	Use Cases	Other Design Elements
SR1.1.8.1.1.8.1.2	The system shall automatically populate the geographic location of the object using the beginning of the interval if an exit is specified and the geographic location of the exit is available from the GIS database.	Map	Select First Intersecting Feature	chartlite.data.location-data.GUILocationDataClasses CD FlexLocationClasses CD
SR1.1.8.1.1.8.1.3	The system shall automatically populate the geographic location of the object using the beginning of the interval if a milepost is specified and the geographic location of the milepost is available from the GIS database.	Map	Select First Intersecting Feature	chartlite.data.location-data.GUILocationDataClasses CD FlexLocationClasses CD
SR1.1.8.1.1.8.2	The system shall allow the user to specify a feature on the route marking the end of the interval containing the object, as described in the Specify Feature On Route requirements.	Map	Select Second Intersecting Feature	FlexLocationClasses CD GUIFlexComponentClasses CD flex.shared.components:SpecifyLocation.updateIntFeatures SD Screenshot: HMI Figure 4-4
SR1.1.8.1.1.8.2.1	The system shall attempt to populate the geographic location of the object using the ending of the interval only if the geographic location is not available for the beginning of the interval.	Map	Select Second Intersecting Feature	
SR1.1.8.1.1.8.2.1.1	The system shall automatically populate the geographic location of the object using the ending of the interval if an intersection with another route is specified and the geographic location of the intersection is available from the GIS database.	Map	Select Second Intersecting Feature	chartlite.data.location-data.GUILocationDataClasses CD FlexLocationClasses CD
SR1.1.8.1.1.8.2.1.2	The system shall automatically populate the geographic location of the object using the ending of the interval if an exit is specified and the geographic location of the exit is available from the GIS database.	Map	Select Second Intersecting Feature	chartlite.data.location-data.GUILocationDataClasses CD FlexLocationClasses CD

Tag	Text	Feature	Use Cases	Other Design Elements
SR1.1.8.1.1.8.2.1.3	The system shall automatically populate the geographic location of the object using the ending of the interval if a milepost is specified and the geographic location of the milepost is available from the GIS database.	Map	Select Second Intersecting Feature	chartlite.data.location-data.GUILocationDataClasses CD FlexLocationClasses CD
SR1.1.8.1.1.8.3	The system shall prevent the user from specifying a beginning and ending feature of an interval that are the same feature.	Map	Specify Intersecting Feature Pair	flex.shared.components:SpecifyLocation.updateIntFeatures SD
SR1.1.8.1.1.8.4	The system shall allow the user to specify the object's proximity to the specified features on the route.	Map	Specify Intersecting Feature Pair	FlexLocationClasses CD GUIFlexComponentClasses CD flex.shared.components:SpecifyLocation.updateIntFeatures SD
SR1.1.8.1.1.8.4.1	The proximity values for describing an object that is located between two features on the route shall include: 'FROM-TO' and 'BETWEEN'.	Map	Specify Intersecting Feature Pair	Common2 CD Screenshot: HMI Figure 4-2
SR1.1.8.1.1.10	The system shall allow the user to specify a textual location description.	Map	MapAndGISUses.Specify ObjectLocation	
SR1.1.8.1.1.10.1	The system shall generate the object location description based on the values specified in the location fields, unless the location description has been overridden by the user.	Map	MapAndGISUses.Specify ObjectLocation	
SR1.1.8.1.1.10.1.2	If a main route is specified, the textual location shall be the main route, with further identifying information as available.	Map	MapAndGISUses.Specify ObjectLocation	
SR1.1.8.1.1.10.1.2.4	If the object location is relative to a feature (or features) on the route, the textual description shall include the object's location relative to the feature(s).	Map	Specify Object Location	
SR1.1.8.1.1.10.1.2.4.1	The textual description shall include the proximity of the object to the specified feature(s).	Map	MapAndGISUses.Specify ObjectLocation MapAndGISUses.SelectIntersectingFeature	

Tag	Text	Feature	Use Cases	Other Design Elements
SR1.1.8.1.1.10.1.2.4.1.1	If the object location's proximity is 'FROM-TO', the textual description shall be formatted as: FROM first feature description TO second feature description.	Map	Specify Object Location	
SR1.1.8.1.1.10.1.2.4.1.2	If the object location's proximity is 'BETWEEN', the textual description shall be formatted as: BETWEEN first feature description AND second feature description.	Map	Specify Object Location	
SR1.1.8.1.2	Specify Object Location Using Map and Form	Map	NA	
SR1.1.8.1.2.3	The system shall pan and/or zoom the map used for specifying an object's location when the user makes selections on the associated location form.	Map	MapAndGISUses.SpecifyObjectLocation MapAndGISUses.SelectState MapAndGISUses.SelectCounty MapAndGISUses.SelectIntersectingFeature	MapViewSpecificClasses CD SpecifyLocation[Map]:jsUpdateMapGeoLocInfo SD EditLocation:zoomMapOnCountyOrStateSelectionChanged
SR1.1.8.1.2.3.4	The system shall pan and/or zoom the map when the Intersecting Feature selection is changed on the Object Location form to include an area around the intersecting feature, if the location of the intersecting feature is known.	Map	MapAndGISUses.SpecifyObjectLocation MapAndGISUses.SelectIntersectingFeature	MapViewSpecificClasses CD EditLocation:latitudeChanged SpecifyLocation[Map]:jsUpdateMapGeoLocInfo SD
SR1.1.8.1.2.3.4.1	The system shall pan and/or zoom the map when the second Intersecting Feature selection is changed on the Object Location form (for a proximity of BETWEEN or FROM-TO) to include an area including both intersecting features, if the location of the intersecting features are known.	Map	Specify Intersecting Feature Pair	SpecifyLocation[Map]:jsUpdateMapGeoLocInfo SD
SR4	MANAGE EVENTS		N/A	N/A
SR4.2	OPEN EVENT		N/A	N/A
SR4.2.2	RECORD EVENT DETAILS		N/A	N/A
SR4.2.2.2	SPECIFY LOCATION(S) AND IMPACT		N/A	N/A

Tag	Text	Feature	Use Cases	Other Design Elements
SR4.2.2.2.9	The system shall allow the user to select or specify the number of roadway lanes affected.	Lane Config	Edit Traffic Event Lane Configuration and Status	chartlite/servlet/trafficevents/initLaneConfigEditor SD
SR4.2.2.2.9.4	The system shall allow a user with the Manage Traffic Events right to specify the state of the traffic lanes for an open Planned Roadway Closure Event.	Lane Config	Edit Traffic Event Lane Configuration and Status	
SR4.2.2.2.9.5	The system shall allow a user with the Manage Traffic Events right to specify the state of the traffic lanes for an open Incident Event.	Lane Config	Edit Traffic Event Lane Configuration and Status	
SR4.2.2.2.9.6	The system shall allow a user with the Manage Traffic Events right to specify the state of the traffic lanes for an open Special Event.	Lane Config	Edit Traffic Event Lane Configuration and Status	
SR4.2.2.2.9.11	The system shall support the lane configuration editing features as specified in the sub-requirements of 10.15.2.	Lane Config	Edit Traffic Event Lane Configuration and Status	chartlite/servlet/trafficevents/initLaneConfigEditor SD
SR4.2.2.2.9.11.1	The system shall allow the radius used by the lane editor when finding nearby lane configurations to be set in the system profile.	Lane Config	Edit Traffic Event Lane Configuration and Status	chartlite/servlet/trafficevents/initLaneConfigEditor SD
SR4.2.2.6	CAPTURE RELATED EVENTS		N/A	
SR4.2.2.6.1	The system shall automatically check on a regular basis to see if any two open events are similar geographically.			
SR4.2.2.6.1.1	Two events shall be flagged as duplicates when two events have the same location.			
SR4.2.2.6.1.1.1	Two event locations will be considered the same when both are geographically located within a configurable number of miles from one another, both are located on the same primary route, and both have the same direction.	Map	Create Duplicate Events CD	UtilityPkg.ObjectCache.ProxyTrafficEvent::areLocationsTheSame SD

Tag	Text	Feature	Use Cases	Other Design Elements
SR4.2.2.6.1.1.1.1	The system shall allow an user with Configure System functional right to configure the number of miles used (to the thousandth of a mile) when comparing two event geographic locations for the purpose of duplicate detection.	Map	Configure Duplicate Event Comparison Rules CD	
SR10	SYSTEM INTEGRATION		N/A	N/A
SR10.1	The system shall interface with other regional ATMS's in the area. Suggestion/example to be validated: RITIS, Regional 911, IEN, CAPWIN, EMMA/MEGIN, WEBEOC, 511, etc.			
SR10.1.4	The system shall support the TMDD standard for TSS data exchange with external systems.			
SR10.1.4.2	The system shall support the export of TSS data to external systems using the TMDD standard.	Exporter	Provide Detector Data to External Systems	webservices.base.BasicRequestHandler.processRequest
SR10.1.4.2.2	The system shall support a method for external systems to obtain an inventory of known TSS's. 'Known TSSs' is defined as CHART TSSs plus imported external TSSs.	Exporter	Provide Detector Data to External Systems	WSDMSExportModuleClasses, DMSExportHandler:getDMSInventoryList (generic for all devices)
SR10.1.4.2.3	The system shall support a method for external systems to receive updates to the inventory of known TSSs.	Exporter	Provide Detector Data to External Systems	WSDMSExportModuleClasses, DMSExportHandler:getDMSInventoryList (generic for all devices)
SR10.1.4.2.3.1	The system shall support an on-demand method for external systems to receive updates to the known TSS inventory.	Exporter	Provide Detector Data to External Systems	Similar to DMSRequestHandler:processRequest, DMSExportHandler:getDMSInventoryList
SR10.1.4.2.3.2	The system shall support a real-time method for external systems to receive updates to the known TSS inventory.	Exporter	Provide Detector Data to External Systems	Similar to DMSSubscriptionSupportClasses, DMSSubscriptionRequestHandler:processRequest, ExportSubscription:sendDataToSubscriber
SR10.1.4.2.4	The system shall support a method for external systems to obtain the status of known TSSs.	Exporter	Provide Detector Data to External Systems	Similar to DMSRequestHandler:processRequest, DMSExportHandler:getDMSStatusList

Tag	Text	Feature	Use Cases	Other Design Elements
SR10.1.4.2.5	The system shall support a method for external systems to receive updates to the status of known TSSs.	Exporter	Provide Detector Data to External Systems	
SR10.1.4.2.5.1	The system shall support an on-demand method for external systems to receive updates to the status of known TSSs.	Exporter	Provide Detector Data to External Systems	Similar to DMSRequestHandler:processRequest, DMSExportHandler:getDMSStatusList
SR10.1.4.2.5.2	The system shall support a real-time method for external systems to receive updates to the status of known TSSs.	Exporter	Provide Detector Data to External Systems	Similar to DMSSubscriptionSupportClasses, DMSSubscriptionRequestHandler:processRequest, ExportSubscription:sendDataToSubscriber
SR10.1.7	The system shall support the TMDD standard for CCTV (Closed Circuit Television) data exchange with external systems.	Exporter	Provide Camera Data to External Systems	
SR10.1.7.1	The system shall support the export of CCTV (Closed Circuit Television) data to external systems using the TMDD standard.	Exporter	Provide Camera Data to External Systems	Webservices.base.BasicRequestHandler.processRequest
SR10.1.7.1.1	The system shall translate CHART CCTV formatted data into TMDD standard formatted CCTV data with CHART extensions.	Exporter	Provide Camera Data to External Systems	<use case only>
SR10.1.7.1.2	The system shall support a method for external systems to obtain an inventory of CHART CCTVs.	Exporter	Provide Camera Data to External Systems	similar to DMSExportHandler:getDMSInventoryList
SR10.1.7.1.3	The system shall support a method for external systems to receive updates to the CHART CCTV inventory.	Exporter	Provide Camera Data to External Systems	similar to DMSRequestHandler:getDMSStatusList
SR10.1.7.1.3.1	The system shall support an on-demand method for external systems to receive updates to the CHART CCTV inventory.	Exporter	Provide Camera Data to External Systems	Similar to DMSRequestHandler:processRequest, DMSExportHandler:getDMSInventoryList
SR10.1.7.1.3.1.1	The on-demand method for receiving CCTV inventory updates shall allow the external system to optionally specify a look-back time period. CCTV inventory updates made that far in the past until 'now' are returned to the external system.	Exporter	Provide Camera Data to External Systems	Similar to DMSRequestHandler:processRequest, DMSExportHandler:getDMSInventoryList

Tag	Text	Feature	Use Cases	Other Design Elements
SR10.1.7.1.3.2	The system shall support a real-time method for external systems to receive updates to the CHART CCTV inventory.	Exporter	Provide Camera Data to External Systems	Similar to DMSSubscriptionSupportClasses, DMSSubscriptionRequestHandler:processRequest, ExportSubscription:sendDataToSubscriber
SR10.1.7.1.4	The system shall support a method for external systems to obtain the status of CHART CCTVs.	Exporter	Provide Camera Data to External Systems	Similar to DMSRequestHandler:getDMSStatusList
SR10.1.7.1.5	The system shall support a method for external systems to receive updates to the status of CHART CCTV's.	Exporter	Provide Camera Data to External Systems	
SR10.1.7.1.5.1	The system shall support an on-demand method for external systems to receive updates to the status of CHART CCTVs.	Exporter	Provide Camera Data to External Systems	Similar to DMSRequestHandler:processRequest, DMSExportHandler:getDMSStatusList
SR10.1.7.1.5.1.1	The on-demand method for receiving CCTV status updates shall allow the external system to optionally specify a look-back time period. CCTV status updates made that far in the past until 'now' are returned to the external system.	Exporter	Provide Camera Data to External Systems	Similar to DMSRequestHandler:processRequest, DMSExportHandler:getDMSStatusList
SR10.1.7.1.5.2	The system shall support a real-time method for external systems to receive updates to the status of CHART CCTVs.	Exporter	Provide Camera Data to External Systems	Similar to DMSSubscriptionSupportClasses, DMSSubscriptionRequesthandler:processRequest, ExportSubscription:sendDataToSubscriber
SR10.6	The system shall integrate with EORS.			
SR10.6.4	The system shall provide user interface tools that assist a CHART operator with the Manage Traffic Events functional right in assigning an EORS permit to a CHART planned closure event.	EORS Integ	Set EORS Permit for Planned Roadway Closure Event	Human Machine Interface: all screens ManageTrafficEvents Use Case Diagram
SR10.6.4.1	The system shall suggest potential matching EORS permits as the user types a search string based on the EORS permit tracking number. Tracking numbers are matched according to Requirement 10.6.4.4 and its sub-requirements.	EORS Integ	View Suggested EORS Permits	Use Case: Set EORS Permit for Planned Roadway Closure Event Human Machine Interface: Figure 4-2 through 4-5 CD: EORSDataClasses

Tag	Text	Feature	Use Cases	Other Design Elements
SR10.6.4.1.1	The system shall allow a user with the Manage Traffic Events functional right to click on a suggested EORS permit to assign it to the CHART Planned Closure Event the operator is currently working with.	EORS Integ	View Suggested EORS Permits	Use Case: View Suggested EORS Permits Human Machine Interface: Figure 4-4: Permit Selection Suggestion.
SR10.6.4.2	The system shall allow a CHART operator to search for EORS permits based on the following fields: permit tracking number, start county name, end county name, permit type, route location, route type, route number, work order description, permittee name, contract number and days of week. Tracking numbers are matched according to Requirement 10.6.4.4 and its sub-requirements.	EORS Integ	Search EORS Permits	Class Diagram: SystemInterfaces.EORS Use Case: Search EORS Permits Human Machine Interface: Figure 4-5, through Figure 4-9
SR10.6.4.2.1	The system shall allow a user to view the results of their EORS permit search.	EORS Integ	Search EORS Permits	Human Machine Interface: Figure 4-6 through 4-9. Use case: Search EORS Permits CD: EORSDataClasses, EORSServletClasses SD: EORSReqHdlr: handleSearchEORSPermitsRequest
SR10.6.4.2.1.1	The system shall sort the search results by relevance to provided search criteria where relevance is calculated as a weighted percentage of words from the search criteria that matched at least one of the search-able attributes of the permit.	EORS Integ	Search EORS Permits	Human Machine Interface: Figure 4-6, 4-7. Use case: Search EORS Permits SD: EORSReqHdlr.handleNewEORSPermitSearch CD: EORSDataClasses
SR10.6.4.2.1.2	The system shall allow the user to view a summary of each EORS permit that matched the search criteria.	EORS Integ	Search EORS Permits	Human Machine Interface: Figure 4-6, 4-7. Use case: Search EORS Permits
SR10.6.4.2.1.3	The system shall allow the user to view more detailed information about any permit shown in the search results.	EORS Integ	Search EORS Permits	Human Machine Interface: Figure 4-8. Use case: Search EORS Permits

Tag	Text	Feature	Use Cases	Other Design Elements
SR10.6.4.2.1.4	The system shall allow a user with the Manage Traffic Events functional right to assign a permit shown on the search results page to the CHART Planned Closure Event the operator is currently working with.	EORS Integ	Search EORS Permits	Use Case: Set EORS Permit for Planned Roadway Closure Event Human Machine Interface: Figure 4-9, Figure 4-10
SR10.6.4.3	The system shall allow users to limit the scope of EORS permit searches.	EORS Integ	Set EORS Permit for Planned Roadway Closure Event	Use Case: Set EORS Permit for Planned Roadway Closure Event Human Machine Interface: Figure 4-2. CD: EORSDataClasses SD: hadnleNewEORSPermitSearch
SR10.6.4.3.1	The system shall allow a user to specify that only EORS permits in the active or queued state should be included in a search.	EORS Integ	Set EORS Permit for Planned Roadway Closure Event	Use Case: Set EORS Permit for Planned Roadway Closure Event Human Machine Interface: Figure 4-2. CD: EORSDataClasses SD: handleNewEORSPermitSearch
SR10.6.4.3.2	The system shall allow a user to specify that all EORS permits should be included in a search.	EORS Integ	Set EORS Permit for Planned Roadway Closure Event	Use Case: Set EORS Permit for Planned Roadway Closure Event Human Machine Interface: Figure 4-2. CD: SystemInterfaces:EORS CD: EORSDataClasses SD: handleNewEORSPermitSearch
SR10.6.4.4	The system shall allow an operator to control how tracking numbers are checked for matches during permit searches.	EORS Integ	Set EORS Permit for Planned Roadway Closure Event	Use Case: Set EORS Permit for Planned Roadway Closure Event Human Machine Interface: Figure 4-2. CD: EORSDataClasses SD: handleNewEORSPermitSearch
SR10.6.4.4.1	The system shall allow a user to indicate that a permit tracking number should be considered a match if it either starts with the user entered text, or if the last three (3) or last four (4) digits of the tracking number start with the user entered text.	EORS Integ	Set EORS Permit for Planned Roadway Closure Event	Use Case: Set EORS Permit for Planned Roadway Closure Event Human Machine Interface: Figure 4-2. CD: EORSDataClasses SD: handleNewEORSPermitSearch
SR10.6.4.4.2	The system shall allow a user to indicate that a permit tracking number should be considered a match only if its last three (3) or last four (4) digits start with the user entered text.	EORS Integ	Set EORS Permit for Planned Roadway Closure Event	Use Case: Set EORS Permit for Planned Roadway Closure Event Human Machine Interface: Figure 4-2. CD: EORSDataClasses SD: handleNewEORSPermitSearch
SR10.13	CHART Export Service. The system shall provide a service for the purpose of providing CHART information to authorized third-party clients.	Exporter	Provide Data to External Systems	WebServicesBaseClasses, WSTrafficEventExportModuleClasses, DMSExportModuleClasses

Tag	Text	Feature	Use Cases	Other Design Elements
SR10.13.1	The external export service shall export data for CHART devices and Traffic Events.	Exporter	Provide Data to External Systems	
SR10.13.1.2	The external export service shall export data on known TSSs to authorized third-party clients who request it based on the client's functional rights (see CHART TSS Export Interface Control Document).	Exporter	Authenticate External System	webservices.base.BasicRequestHandler.processRequest
SR10.13.1.5	The external export service shall export data on CHART Cameras to authorized third-party clients who request it based on the client's functional rights (see CHART CCTV Export Interface Control Document).	Exporter	Authenticate External Systems	webservices.base.BasicRequestHandler.processRequest
SR10.15	The system shall provide an interface to allow external applications to create and edit lane configurations for use in their application.	Lane Config	Edit Lane Configuration and Status	webservices/laneeditormodule/LaneEditorClassDiagram CD webservices/laneeditormodule/LaneEditorRequestHandler:initializeEditingSession SD
SR10.15.1	The system shall allow a client application to initiate a lane configuration editing session.	Lane Config	Edit Lane Configuration and Status	webservices/laneeditormodule/LaneEditorRequestHandler:initializeEditingSession SD
SR10.15.1.1	The system shall require the client application to specify the title for the lane editor, the primary direction for the lanes in the editor, and a callback URL to be called when the editing session has ended.	Lane Config	Edit Lane Configuration and Status	webservices/laneeditormodule/LaneEditorRequestHandler:initializeEditingSession SD LaneEditorWebService ICD
SR10.15.1.2	The system shall allow the client application to specify search criteria including lat/long, a search radius, and optionally a route. This search criteria will be used by the lane configuration editor to find nearby lane configurations to include in the list of available lane configurations.	Lane Config	Edit Lane Configuration and Status	webservices/laneeditormodule/LaneEditorRequestHandler:initializeEditingSession SD LaneEditorWebService ICD

Tag	Text	Feature	Use Cases	Other Design Elements
SR10.15.1.3	The system shall allow the client application to specify name/value pairs to be included in the data passed back to the client application when the user ends the lane editing session.	Lane Config	Edit Lane Configuration and Status	webservices/laneeditormodule/LaneEditorRequestHandler:initializeEditingSession SD LaneEditorWebService ICD
SR10.15.1.4	The system shall allow the client application to specify an initial lane configuration, used to initialize the state of the lane editor when first displayed.	Lane Config	Edit Lane Configuration and Status	webservices/laneeditormodule/LaneEditorRequestHandler:initializeEditingSession SD LaneEditorWebService ICD
SR10.15.1.5	The system shall allow the client application to specify that certain features of the lane editor are to be disabled (or not shown).	Lane Config	Edit Lane Configuration and Status	webservices/laneeditormodule/LaneEditorRequestHandler:initializeEditingSession SD LaneEditorWebService ICD
SR10.15.1.5.1	The system shall allow the lane editor feature that sets a lane direction to bidirectional to be disabled.	Lane Config	Edit Lane Configuration and Status	LaneEditorWebService ICD
SR10.15.1.5.2	The system shall allow the lane editor feature that initializes lane states in the primary direction of the lane configuration to unknown when a lane configuration is first selected to be disabled. (See 10.15.2.5.8 and its subrequirements.)	Lane Config	Edit Lane Configuration and Status	LaneEditorWebService ICD
SR10.15.2	Lane configuration editing functionality shall be made available to the client application as a web page for use within the client application.	Lane Config	Edit Lane Configuration and Status	webservices/laneeditormodule/LaneEditorRequestHandler:viewLaneEditor LaneEditorWebService ICD
SR10.15.2.2	The lane configuration editor shall be initialized to the lane configuration and status specified when the lane configuration editing session was initialized, if any.	Lane Config	Edit Lane Configuration and Status	webservices/laneeditormodule/LaneEditorClassDiagram (LaneEditorData constructor)
SR10.15.2.5	The lane configuration editor shall allow the user to choose the lane configuration from a list of available lane configurations.	Lane Config	Select Lane Configuration	webservices/laneeditormodule/LaneEditorRequestHandler:useStandardLaneConfig SD webservices/laneeditormodule/LaneEditorRequestHandler:useNearbyLaneConfig SD

Tag	Text	Feature	Use Cases	Other Design Elements
SR10.15.2.5.1	The available lane configurations shall include those nearby the location search criteria specified during initialization of the lane configuration editing session (if any).	Lane Config	Select Lane Configuration	webservices/laneeditormodule/LaneEditorRequestHandler:getNearbyLaneConfigs SD
SR10.15.2.5.1.1	If the initialization of the lane configuration editing session included a lat/long, the system shall consider a lane configuration nearby if the lane configuration location is within the specified radius of the lat/long.	Lane Config	View Nearby Lane Configurations	webservices/laneeditormodule/LaneEditorRequestHandler:getNearbyLaneConfigs SD
SR10.15.2.5.1.2	If the initialization of the lane configuration editing session included a main route, the nearby lane configurations shall be filtered to include only those lane configurations that are on the specified route.	Lane Config	View Nearby Lane Configurations	webservices/laneeditormodule/LaneEditorRequestHandler:getNearbyLaneConfigs SD
SR10.15.2.5.1.5	The lane configuration editor shall include for selection nearby lane configurations that are obtained from the mapping database.	Lane Config	Select Lane Configuration	webservices/laneeditormodule/LaneEditorRequestHandler:getNearbyLaneConfigs SD
SR10.15.2.5.1.6	The lane configuration editor shall include for selection nearby lane configurations that were specified by users.	Lane Config	Select Lane Configuration	webservices/laneeditormodule/LaneEditorRequestHandler:getNearbyLaneConfigs SD
SR10.15.2.5.2	The available lane configurations shall include standard lane configurations available in the system.	Lane Config	Select Lane Configuration	webservices/laneeditormodule/LaneEditorCalassDiagram CD
SR10.15.2.5.3	The lane configuration editor shall list the available lane configurations in the following order: user specified lane configurations (ordered by distance from the specified location), lane configurations from the mapping database (ordered by distance from the specified location), followed by standard lane configurations available in the system.	Lane Config	Select Lane Configuration	

Tag	Text	Feature	Use Cases	Other Design Elements
SR10.15.2.5.4	The lane configuration editor shall indicate the source of a lane configuration as being from the mapping database, specified by a user, or a standard lane configuration.	Lane Config	Select Lane Configuration	
SR10.15.2.5.5	The lane configuration editor shall show only unique lane configurations for selection.	Lane Config	Select Lane Configuration	
SR10.15.2.5.7	The first lane configuration in the list of available lane configurations shall be selected by default unless a lane configuration was specified during initialization of the lane configuration editing session. (Note, the list is sorted as specified in 10.15.2.5.3).	Lane Config	Specify Lane Configuration	
SR10.15.2.5.8	The lane configuration editor shall set all lanes in the primary direction to “unknown” and all lanes in the other direction to “open” when a lane configuration is first selected. (This applies to the user choosing a new lane configuration or the system selecting an initial lane configuration when an initial lane configuration is not specified as part of the initialization of the lane configuration editing session).	Lane Config	Select Lane Configration	webservices/laneeditormodule/LaneEditorRequestHandler:UseNearbyLaneConfig SD, webservices/laneeditormodule/LaneEditorRequestHandler:UseStandardLaneConfig SD,
SR10.15.2.5.8.1	The lane configuration editor shall initialize all lanes to "open" if the initialization of the lane configuration editing session indicates the feature to set lanes to Unknown is specified to be disabled.	Lane Config	Select Lane Configuration	webservices/laneeditormodule/LaneEditorRequestHandler:UseNearbyLaneConfig SD, webservices/laneeditormodule/LaneEditorRequestHandler:UseStandardLaneConfig SD,

Tag	Text	Feature	Use Cases	Other Design Elements
SR10.15.2.5.8.2	The lane configuration editor shall set all unknown lanes in the main direction to open the first time an operator changes any lane to open or closed after the lane configuration has been set and the state of the lanes in the main direction were initialized to unknown.	Lane Config	Record Lane Closures	webservices/laneeditormodule/LaneEditorRequestHandler:SetLanesState SD
SR10.15.2.6	The lane configuration editor shall allow the user to customize the currently selected lane configuration.	Lane Config	Customize Lane Configuration	webservices/laneeditormodule/LaneEditorRequestHandler:addLane SD, webservices/laneeditormodule/LaneEditorRequestHandler:removeLanes SD, webservices/laneeditormodule/LaneEditorRequestHandler:removeAllLanes SD
SR10.15.2.6.1	The lane configuration editor shall allow the user to add a lane to the configuration.	Lane Config	Customize Lane Configuration	webservices/laneeditormodule/LaneEditorRequestHandler:addLane SD
SR10.15.2.6.2	The lane configuration editor shall allow the user to remove a lane from the configuration.	Lane Config	Customize Lane Configuration	webservices/laneeditormodule/LaneEditorRequestHandler:removeLanes SD
SR10.15.2.6.4	The lane configuration editor shall allow the user to remove all lanes from the configuration so they may create a custom configuration "from scratch".	Lane Config	Customize Lane Configuration	webservices/laneeditormodule/LaneEditorRequestHandler:removeAllLanes SD
SR10.15.2.7	The lane configuration editor shall allow the user to specify the status of each lane (does not apply to medians and other dividers such as double yellow line).	Lane Config	Record Lane Closure	webservices/laneeditormodule/LaneEditorRequestHandler:setLanesState SD, webservices/laneeditormodule/LaneEditorRequestHandler:setLanesStateChangedTime SD, webservices/laneeditormodule/LaneEditorRequestHandler:setAllLanesOpen SD
SR10.15.2.7.3	The lane configuration editor shall allow the user to specify if a lane is "open", "closed", or "unknown".	Lane Config	Record Lane Closure	webservices/laneeditormodule/LaneEditorRequestHandler:setLanesState SD

Tag	Text	Feature	Use Cases	Other Design Elements
SR10.15.2.7.3.1	The lane configuration editor shall record the date/time a lane is closed. (The time closed is the time when the user marks the lane as closed).	Lane Config	Record Lane Closure	webservices/laneeditormodule/LaneEditorRequestHandler:setLanesState SD
SR10.15.2.7.3.2	The lane configuration editor shall record the date/time a lane is opened. (The time opened is the time the user marks the lane as open).	Lane Config	Record Lane Closure	webservices/laneeditormodule/LaneEditorRequestHandler:setLanesState SD
SR10.15.2.7.3.3	The lane configuration editor shall allow the user to override the lane closure time recorded by the system.	Lane Config	Override Lane Status Change Time	webservices/laneeditormodule/LaneEditorRequestHandler:setLanesStateChangedTime SD
SR10.15.2.7.3.4	The lane configuration editor shall allow the user to override the lane opening time recorded by the system.	Lane Config	Override Lane Status Change Time	webservices/laneeditormodule/LaneEditorRequestHandler:setLanesStateChangedTime SD
SR10.15.2.7.3.5	The system shall allow the user to change the state of all lanes to "open" without having to select lanes individually.	Lane Config	Record Lane Closure	webservices/laneeditormodule/LaneEditorRequestHandler:setAllLanesOpen SD
SR10.15.2.7.4	The lane configuration editor shall allow the user to change the traffic flow direction for a lane or set the traffic flow for the lane to be multi-directional (when traffic is given alternating use of a single lane during flagging operations, etc.)	Lane Config	Change Lane Direction	webservices/laneeditormodule/LaneEditorRequestHandler:setLanesTrafficFlowDir SD
SR10.15.2.7.4.1	The lane configuration editor shall disable the ability to set lane traffic flow directions to multi-directional if the request to initialize the lane configuration editing session specified that this feature is to be disabled.	Lane Config	Change Lane Direction	webservices/laneeditormodule/LaneEditorClassDiagram CD
SR10.15.2.8	The lane configuration editor shall display a roadway graphic depicting the currently specified lane configuration and status.	Lane Config	View Lane Configuration and Status Graphically	webservices/laneeditormodule/LaneEditorRequestHandler:sendLaneConfigGraphicJSON SD
SR10.15.2.8.1	The roadway graphic shall support displaying the status of each lane.	Lane Config	View Lane Configuration and Status Graphically	webservices/laneeditormodule/LaneEditorRequestHandler:sendLaneConfigGraphicJSON SD

Tag	Text	Feature	Use Cases	Other Design Elements
SR10.15.2.8.1.1	The roadway graphic shall support lane state indicators that do not rely solely on color to accommodate colorblind users.	Lane Config	View Lane Configuration and Status Graphically	
SR10.15.2.8.2	The roadway graphic shall indicate the roadway / lane direction(s).	Lane Config	View Lane Configuration and Status Graphically	
SR10.15.2.8.3	The roadway graphic shall display the current traffic flow direction for each lane.	Lane Config	View Lane Configuration and Status Graphically	
SR10.15.2.8.4	The roadway graphic shall be capable of displaying travel lanes, shoulders, multi-lane ramps, tunnel bores, toll plazas, left exit ramps, right entrance ramps, left entrance ramps, right entrance ramps, center turn lanes, medians, and double yellow lines.	Lane Config	View Lane Configuration and Status Graphically	
SR10.15.2.9	The lane configuration editor shall display a textual description of the currently specified lane configuration and status.	Lane Config	View Lane Configuration and Status Textually	
SR10.15.2.10	The system shall store the lane configuration specified when the lane configuration editor is submitted (if a lat/long and main route were specified when the lane configuration editor was initialized) as the user specified lane configuration at the specified location (User specified lane configurations are stored for the purpose of their use in the features specified in 10.15.2.5.1.6)	Lane Config	Store User Specified Lane Config	webservices/laneeditormodule/LaneEditorRequestHandler:endLaneEditorSession SD, webservices/laneeditormodule/LaneEditorRequestHandler:sendStoreLaneConfigRequest SD
SR10.15.2.10.1	The system shall store a timestamp indicating when a user specified lane configuration was stored.	Lane Config	Store User Specified Lane Config	

Tag	Text	Feature	Use Cases	Other Design Elements
SR10.15.2.10.2	If a user specified lane configuration already exists for a location where a new user specified lane configuration is being stored, the system shall replace the old configuration with the newer configuration. (This has the effect of updating the timestamp if the new configuration is identical to the existing configuration)	Lane Config	Store User Specified Lane Config	
SR10.15.2.11	After the user makes any changes in the lane configuration editor, the editor shall display a warning to the user to indicate that they must submit the form to commit their changes, and shall warn them if they cancel the form.	Lane Config	Edit Lane Configuration and Status	
SR10.15.3	The system shall notify the client application when the user has ended the lane configuration session by issuing an HTTP POST request to the URL that was provided when the lane configuration editing session was initialized.	Lane Config	Edit Lane Configuration and Status	webservices/laneeditormodule/LaneEditorRequestHandler:sendLaneEditorCallbackRequest SD
SR10.15.3.1	The notification of the end of a lane editing session shall include the lane configuration and status as specified by the user in the lane configuration editor.	Lane Config	Edit Lane Configuration and Status	webservices/laneeditormodule/LaneEditorRequestHandler:sendLaneEditorCallbackRequest SD
SR10.15.3.2	The notification of the end of a lane configuration editing session shall include name/value pairs that were specified when the lane configuration editing session was initialized.	Lane Config	Edit Lane Configuration and Status	webservices/laneeditormodule/LaneEditorRequestHandler:sendLaneEditorCallbackRequest SD
SR10.15.4	The lane configuration editing interface shall be provided as an HTTP web service.	Lane Config	Edit Lane Configuration and Status	webservices/laneeditormodule/LaneEditorClassDiagram CD
SR10.15.6	The system shall allow a client application to request an image that corresponds to a specified lane configuration and status.	Lane Config	Render Lane Configuration and Status	webservices/laneeditormodule/LaneEditorRequestHandler:renderLaneConfig SD

Tag	Text	Feature	Use Cases	Other Design Elements
SR14	OPERATIONAL ENVIRONMENT	Map		
SR14.1	The system shall support Internet Explorer Versions 7.x and 8.x.	Map		

The following table shows how the requirements in the Mapping R5 Requirements document map to design elements contained in this design.

Tag	Text	Feature	Use Cases	Other Design Elements
SR4	Non-Functional Requirements.	MapGran		
SR4.2	Security	ExternalTSS		
SR4.2.1	The system shall integrate with CHART user authentication mechanism.	ExternalTSS		
SR4.2.1.1	The system shall support the definition of a default user whose rights are used for unauthenticated users. An unauthenticated user is one who has not yet logged in.	ExternalTSS, Map		
SR4.2.1.2	The system shall support the use of login procedure to authenticate users with the CHART user authentication mechanism.	ExternalTSS, Map		
SR4.2.1.3	Once a user has been authenticated, the system shall retain the user's authentication rights until they close the application.	ExternalTSS, Map		
SR6	Detailed Map Layer Requirements			
SR6.4	Traffic Speed Sensor (TSS/RTMS)			

Tag	Text	Feature	Use Cases	Other Design Elements
SR6.4.1	The system shall display traffic speed detectors on the map as Arrow icons.			
SR6.4.1.1	The system shall display SHA traffic speed detectors on the map to all users by default.	ExternalTSS, Map		
SR6.4.1.2	The system shall display external traffic speed detectors on the map.	ExternalTSS, Map		
SR6.4.1.3	The system shall allow user to include or exclude the display of speed sensors on the map by the owning organization.	ExternalTSS, Map		
SR6.4.1.4	The system shall display traffic speed detector icons on the map based on the current map extent and at or below the configurable level.	ExternalTSS, Map		
SR6.4.6	When user hovers over the detector icon, tool tip shall display following information: Detector location, Last data report date and time, Road direction(s), Actual speed value or speed range, Owning organization	ExternalTSS		
SR6.4.6.1	The system shall display an actual speed value for each detector direction if the current user has CHART's ViewVSODetailedData right for the detector's owning organization.	ExternalTSS, Map		

Tag	Text	Feature	Use Cases	Other Design Elements
SR6.4.6.2	The system shall display the speed range for each detector direction if the current user does not have CHART's ViewVSODetailedData right for the detector's owning organization.	ExternalTSS, Map		
SR6.4.8	Legend Display	ExternalTSS, Map		
SR6.4.8.1	System shall display both Speed Ranges and Organizations as the sub category of Road Speed Sensor	ExternalTSS, Map		
SR6.4.8.2	The System shall provide a list of speed ranges in the Speed Ranges sub-category of the Road Speed Sensor element of the legend.	ExternalTSS, Map		
SR6.4.8.3	The System shall provide a list of organizations in the Organization sub-category of the Road Speed Sensor element of the legend.	ExternalTSS, Map		
SR11	Data Exporter Synchronization Requirements	DataSync	N/A	N/A
SR11.1	General	DataSync	N/A	N/A
SR11.1.9	The system shall import camera Inventory data from CHART.	DataSync, DataSyncCamera	Synchronize Add Events , Synchronize Update Events	CHARTMap.Handlers.CameraInventoryHandler CD
SR11.1.9.1	The system shall use ID, Name, County, location, latitude, longitude and blocked to public information provided by CHART for CAMERA devices.	DataSync, DataSyncCamera	Synchronize Add Events , Synchronize Update Events	CHARTMap.Handlers.CameraInventoryHandler CD

Tag	Text	Feature	Use Cases	Other Design Elements
SR11.2	Synchronize Add Event	DataSync	Synchronize Add Events	CHART Data Exporter Synchronization SD
SR11.2.2	The synchronization application shall add an entry to the spatial table if a new CHART Device (DMS, HAR, SHAZAM, CAMERA) is found.	DataSync, DataSyncCamera	Synchronize Add Events	CHART Data Exporter Synchronization SD
SR11.3	Synchronize Update Event	DataSync	Synchronize Update Events	CHART Data Exporter Synchronization:UpdateInventory() SD
SR11.3.2	The synchronization application shall update an entry based on the unique ID in the relevant spatial table if a CHART Device (DMS, HAR, SHAZAM, CAMERA) changes location.	DataSync, DataSyncCamera	Synchronize Update Events	CHART Data Exporter Synchronization:UpdateInventory() SD
SR11.5	Functionalities removed from Device Editor	DataSync	Synchronize Add Events , Synchronize Update Events , Synchronize Remove Events	N/A
SR11.5.1	The Device Editor shall prohibit the display of CHART Devices (DMS, HAR, SHAZAM and CAMERA).	DataSync, DataSyncCamera	Synchronize Add Events , Synchronize Update Events , Synchronize Remove Events	N/A
SR11.5.2	The Device Editor shall not allow users to add CHART Devices (DMS, HAR, SHAZAM and CAMERA) to the map. (This will be done via CHART only.)	DataSync, DataSyncCamera	Synchronize Add Events , Synchronize Update Events , Synchronize Remove Events	N/A

Tag	Text	Feature	Use Cases	Other Design Elements
SR11.5.3	The Device Editor shall not allow users to update CHART Devices (DMS, HAR, SHAZAM and CAMERA). (This will be done via CHART only.)	DataSync, DataSyncCamera	Synchronize Add Events , Synchronize Update Events , Synchronize Remove Events	N/A
SR11.5.4	The Device Editor shall not allow users to remove CHART Devices (DMS, HAR, SHAZAM and CAMERA) from the map. (This will be done via CHART only.)	DataSync, DataSyncCamera	Synchronize Add Events , Synchronize Update Events , Synchronize Remove Events	N/A
SR12	MD iMap REST Service Requirements		Synchronize Add Events , Synchronize Update Events , Synchronize Remove Events	CHART Data Exporter Synchronization SD CHARTMap.Handlers.CameraInventoryHandler CD
SR12.2	Layer availability		Synchronize Add Events , Synchronize Update Events , Synchronize Remove Events	CHART Data Exporter Synchronization SD CHARTMap.Handlers.CameraInventoryHandler CD
SR12.2.4	The Service shall allow the display of closed circuit television (CCTV) data.	DataSyncCamera	Synchronize Add Events , Synchronize Update Events , Synchronize Remove Events	CHART Data Exporter Synchronization SD CHARTMap.Handlers.CameraInventoryHandler CD
SR12.2.4.1	The Closed Circuit TV (CCTV) Layers shall include the feed number and site location information for	DataSyncCamera	Synchronize Add Events , Synchronize	CHART Data Exporter Synchronization SD CHARTMap.Handlers.CameraInventoryHandler CD

Tag	Text	Feature	Use Cases	Other Design Elements
	viewing.		Update Events , Synchronize Remove Events	
SR13	Lane Config GIS Web Service Requirements	GISLaneConfig	N/A	N/A
SR13.1	Common	GISLaneConfig	N/A	N/A
SR13.1.1	The System shall process incoming requests sent as Http GET or POST requests in XML format.	GISLaneConfig	Find Nearby Lane Configurations by Location / Store Lane Configuration for a Location & Route Information	/CHARTMapping/Release6/GISLaneConfigWS /GISLaneConfigHelper CD
SR13.1.2	The System shall provide output in XML format.	GISLaneConfig	Find Nearby Lane Configurations by Location / Store Lane Configuration for a Location & Route Information	/CHARTMapping/Release6/GISLaneConfigWS /LaneConfigGISWebService:getLaneConfigura tion storeLaneConfiguration SD
SR13.1.3	The System shall include a detailed error message in the output XML if the request processing fails.	GISLaneConfig	Find Nearby Lane Configurations by Location / Store Lane Configuration for a Location & Route Information	/CHARTMapping/Release6/GISLaneConfigWS /GISLaneConfigHelper CD

Tag	Text	Feature	Use Cases	Other Design Elements
SR13.1.4	The System shall include the request parameters in the output XML.	GISLaneConfig	Find Nearby Lane Configurations by Location / Store Lane Configuration for a Location & Route Information	/CHARTMapping/Release6/GISLaneConfigWS /GISLaneConfigHelper CD
SR13.1.5	The System shall ignore any undefined parameters passed with the request.	GISLaneConfig	Find Nearby Lane Configurations by Location / Store Lane Configuration for a Location & Route Information	/CHARTMapping/Release6/GISLaneConfigWS /LaneConfigGISWebService:getLaneConfigura tion storeLaneConfiguration SD
SR13.1.6	The output XML returned by the System shall conform to a published XSD Schema.	GISLaneConfig	Find Nearby Lane Configurations by Location / Store Lane Configuration for a Location & Route Information	/CHARTMapping/Release6/GISLaneConfigWS /LaneConfigGISWebService:getLaneConfigura tion storeLaneConfiguration SD
SR13.1.7	The System shall indicate whether the request has been processed successfully in the output XML.	GISLaneConfig	Find Nearby Lane Configurations by Location / Store Lane Configuration for a Location & Route Information	/CHARTMapping/Release6/GISLaneConfigWS /LaneConfigGISWebService:getLaneconRslt SD

Tag	Text	Feature	Use Cases	Other Design Elements
SR13.1.9	The Input request shall be in XML format and shall conform to a published XSD Schema.	GISLaneConfig	Find Nearby Lane Configurations by Location / Store Lane Configuration for a Location & Route Information	/CHARTMapping/Release6/GISLaneConfigWS /LaneConfigGISWebService:getLaneConfigura tion storeLaneConfiguration SD
SR13.2	The system shall provide a service to obtain lane configurations for nearby locations.	GISLaneConfig	Find Nearby Lane Configurations by Location	/CHARTMapping/Release6/GISLaneConfigWS /LaneConfigGISWebService:getLaneConfigura tion SD
SR13.2.1	The system shall accept input parameters necessary to provide lane configurations.	GISLaneConfig	Find Nearby Lane Configurations by Location	/CHARTMapping/Release6/GISLaneConfigWS /GISLaneConfig CD
SR13.2.1.1	GetNearbyLaneConfigs shall be processed as a request to get Lane Configurations by Location, Radius, and Route.	GISLaneConfig	Find Nearby Lane Configurations by Location	/CHARTMapping/Release6/GISLaneConfigWS /LaneConfigGISWebService:getLaneConfigura tion SD
SR13.2.1.2	The System shall require Location and Radius in order to look up nearby lane configurations.	GISLaneConfig	Find Nearby Lane Configurations by Location	/CHARTMapping/Release6/GISLaneConfigWS /LaneConfigGISWebService:getLaneConfigura tion SD
SR13.2.1.2.1	The System shall require Location (Latitude/Longitude) and Radius in thousandths of a mile in order to look up the nearby Lane Configurations.	GISLaneConfig	Find Nearby Lane Configurations by Location	/CHARTMapping/Release6/GISLaneConfigWS /LaneConfigGISWebService:getLaneconRslt SD
SR13.2.1.3	The System shall accept optional parameters which specify a primary route in order to refine lists of nearby lane configurations.	GISLaneConfig	Find Nearby Lane Configurations by Location	/CHARTMapping/Release6/GISLaneConfigWS /GISLaneConfig CD

Tag	Text	Feature	Use Cases	Other Design Elements
SR13.2.1.3.1	The System shall also use the Route Type, Route Number, Route Prefix, Route Suffix and Route Name in order to filter the nearby Lane Configurations on the supplied Route, if available.	GISLaneConfig	Find Nearby Lane Configurations by Location	/CHARTMapping/Release6/GISLaneConfigWS /LaneConfigGISWebService:getLaneconRslt SD
SR13.2.4	The System shall provide the nearby lane configurations for the requested location and a route.	GISLaneConfig	Find Nearby Lane Configurations by Location	/CHARTMapping/Release6/GISLaneConfigWS /GISLaneConfigHelper CD
SR13.2.4.2	The System shall provide the Shoulder,Traffic Lane,Collector Distributer,Tunnel Lane,Toll Lane,Center Turn Lane,Right On Ramp,Right Off Ramp,Right Merge Lane,Right Acceleration Lane,Right Turn Lane,Right Deceleration Lane,Left On Ramp,Left Off Ramp,Left Acceleration Lane,Left Merge Lane,Left Turn Lane,Left Deceleration Lane,Double Yellow Line and Median information as part of each Lane Configuration, if available.	GISLaneConfig	Find Nearby Lane Configurations by Location	/CHARTMapping/Release6/GISLaneConfigWS /GISLaneConfig CD
SR13.2.4.3	The System shall indicate the source of a Lane Configurations as being from the mapping database or user specified lane configuration.	GISLaneConfig	Find Nearby Lane Configurations by Location	/CHARTMapping/Release6/GISLaneConfigWS /GISLaneConfig CD
SR13.2.4.7	The System shall provide the lane types, relative direction,lane description and lane reference direction as part of the Lane Configuration.	GISLaneConfig	Find Nearby Lane Configurations by Location	/CHARTMapping/Release6/GISLaneConfigWS /GISLaneConfig GISLaneConfigHelper CD

Tag	Text	Feature	Use Cases	Other Design Elements
SR13.2.4.8	The System shall use Outer Loop, North or East as a reference direction	GISLaneConfig	Find Nearby Lane Configurations by Location	/CHARTMapping/Release6/GISLaneConfigWS /GISLaneConfig CD
SR13.2.4.9	The System shall provide the Lane Configurations nearby to the given location within a supplied radius.	GISLaneConfig	Find Nearby Lane Configurations by Location	/CHARTMapping/Release6/GISLaneConfigWS /GISLaneConfigHelper CD
SR13.2.4.10	The System shall filter the provided Lane Configurations to include only those for the specified Route if route information is provided in the request.	GISLaneConfig	Find Nearby Lane Configurations by Location	/CHARTMapping/Release6/GISLaneConfigWS /LaneConfigGISWebService:getLaneConfigura tion SD
SR13.2.4.11	The System shall provide the distance from the input location for each lane configuration.	GISLaneConfig	Find Nearby Lane Configurations by Location	/CHARTMapping/Release6/GISLaneConfigWS /LaneConfigGISWebService:getLaneconRslt SD
SR13.3	The System shall store the User specified lane configuration.	GISLaneConfig	Store Lane Configuration for a Location & Route Information	/CHARTMapping/Release6/GISLaneConfigWS /LaneConfigGISWebService:storeLaneRecords SD
SR13.3.1	The System shall accept input parameters necessary to store lane configurations.	GISLaneConfig	Store Lane Configuration for a Location & Route Information	/CHARTMapping/Release6/GISLaneConfigWS /LaneConfigGISWebService:storeLaneConfigu ration SD
SR13.3.1.10	The system shall require the lane configuration that is to be stored to be specified.	GISLaneConfig	Store Lane Configuration for a Location & Route Information	/CHARTMapping/Release6/GISLaneConfigWS /LaneConfigGISWebService:storeLaneConfigu ration SD
SR13.3.1.10.1	The lane configuration shall include a list of lanes.	GISLaneConfig	Store Lane Configuration for a Location &	/CHARTMapping/Release6/GISLaneConfigWS /LaneConfigGISWebService:storeLaneRecords SD GISLaneConfig CD

Tag	Text	Feature	Use Cases	Other Design Elements
			Route Information	
SR13.3.1.10.1.1	The following lane types shall be supported: Shoulder,Traffic Lane, CollectorDistributer, Tunnel Lane, Toll Lane, Center Turn Lane, Right On Ramp, Right Off Ramp, Right Merge Lane, Right Acceleration Lane, Right Turn Lane, Right Deceleration Lane, Left On Ramp, Left Off Ramp, Left Acceleration Lane, Left Merge Lane, Left Turn Lane, Left Deceleration Lane, Double Yellow Line, and Median.	GISLaneConfig	Store Lane Configuration for a Location & Route Information	/CHARTMapping/Release6/GISLaneConfigWS /GISLaneConfig CD
SR13.3.1.10.1.2	The system shall require that a relative direction be specified for each lane.	GISLaneConfig	Store Lane Configuration for a Location & Route Information	/CHARTMapping/Release6/GISLaneConfigWS /LaneConfigGISWebService:storeLaneConfiguration SD
SR13.3.1.10.1.2.1	The relative direction for a lane can be indicated as being the same as the reference direction of the lane configuration, opposite to the reference direction of the lane configuration, or no direction.	GISLaneConfig	Store Lane Configuration for a Location & Route Information	/CHARTMapping/Release6/GISLaneConfigWS /LaneConfigGISWebService:storeLaneRecords SD
SR13.3.1.10.1.3	The system shall require that a description be included for each lane.	GISLaneConfig	Store Lane Configuration for a Location & Route Information	/CHARTMapping/Release6/GISLaneConfigWS /GISLaneConfig CD

Tag	Text	Feature	Use Cases	Other Design Elements
SR13.3.1.10.2	The lane configuration shall include a description.	GISLaneConfig	Store Lane Configuration for a Location & Route Information	/CHARTMapping/Release6/GISLaneConfigWS /GISLaneConfig GISLaneConfigHelper CD
SR13.3.1.10.3	The lane configuration shall include a reference direction.	GISLaneConfig	Store Lane Configuration for a Location & Route Information	/CHARTMapping/Release6/GISLaneConfigWS / GISLaneConfig GISLaneConfigHelper CD
SR13.3.1.11	The System shall require the location of the lane configuration (latitude/longitude) to be specified.	GISLaneConfig	Store Lane Configuration for a Location & Route Information	/CHARTMapping/Release6/GISLaneConfigWS /GISLaneConfig CD
SR13.3.1.12	The System shall require the route that contains the lane configuration to be specified.	GISLaneConfig	Store Lane Configuration for a Location & Route Information	/CHARTMapping/Release6/GISLaneConfigWS /GISLaneConfig CD
SR13.3.2	The System shall process a input parameters necessary to store the lane configurations.	GISLaneConfig	Store Lane Configuration for a Location & Route Information	/CHARTMapping/Release6/GISLaneConfigWS /LaneConfigGISWebService:storeLaneConfiguration SD
SR13.3.2.1	The System shall determine the road segment for which the lane configuration is to be stored by finding the road segment that is closest to the given location and is on the given route.	GISLaneConfig	Store Lane Configuration for a Location & Route Information	/CHARTMapping/Release6/GISLaneConfigWS /LaneConfigGISWebService:storeLaneRecords SD

Tag	Text	Feature	Use Cases	Other Design Elements
SR13.3.2.2	The System shall store the given lane configuration as the user specified lane configuration for the road segment.	GISLaneConfig	Store Lane Configuration for a Location & Route Information	/CHARTMapping/Release6/GISLaneConfigWS /LaneConfigGISWebService:storeLaneRecords SD
SR13.3.2.3	The System shall record the date/time the lane configuration is stored.	GISLaneConfig	Store Lane Configuration for a Location & Route Information	/CHARTMapping/Release6/GISLaneConfigWS /LaneConfigGISWebService:storeLaneRecords SD
SR13.3.2.4	The System shall return an error if it cannot locate the specified route using the given route information.	GISLaneConfig	Store Lane Configuration for a Location & Route Information	/CHARTMapping/Release6/GISLaneConfigWS /LaneConfigGISWebService:storeLaneRecords SD
SR13.3.3	The System shall provide the output neccessary to indicate the result of the request.	GISLaneConfig	Store Lane Configuration for a Location & Route Information	/CHARTMapping/Release6/GISLaneConfigWS /LaneConfigGISWebService:storeLaneRecords LaneConfigGISWebService:storeLaneConfigur ation SD
SR13.3.3.1	The System shall provide an error or success message as a output to the request.	GISLaneConfig	Store Lane Configuration for a Location & Route Information	/CHARTMapping/Release6/GISLaneConfigWS /LaneConfigGISWebService:storeLaneRecords LaneConfigGISWebService:storeLaneConfigur ation SD
SR13.3.5	The System shall authenticate the every store lane configuration request using client ID and signature.	GISLaneConfig	Store Lane Configuration for a Location & Route Information	/CHARTMapping/Release6/GISLaneConfigWS /LaneConfigGISWebService:storeLaneConfigu ration SD

16Acronyms/Glossary

GIS	**Geographic Information System** (GIS) is any system that captures, stores, analyzes, manages, and presents data that are linked to location
Home Page Map	The map component shown on the home page of the CHART user interface.
Integrated Map	The mapping components that are being built into the CHART user interface as part of Release 6 of the CHART application.
Intranet Map	The CHART Mapping application that is not integrated into the CHART user interface.
Location Alias	A pre-defined location (lat/lon) that has been stored with some name attributes to allow operators to utilize the location repeatedly.
Maintenance Portal	A customized version of the CHART GUI tailored to device maintenance personnel.
Nearby Devices Map	Map shown on the details page for a traffic event that shows only the target traffic event and the devices that are near it.
Object Location Map	Map component that is used in conjunction with the object location form when setting the location of a traffic event or device.
Open Layers	Open source JavaScript mapping API utilized by the integrated map components in the CHART GUI.
REST	Representational State Transfer - a web services architecture style used in CHART that leverages web technologies such as http and XML
Standard GUI	The CHART GUI when ***not*** accessed via the maintenance portal.
WMS	A Web Map Service (WMS) is a standard protocol for serving georeferenced map images over the Internet that are generated by a map server using data from a GIS database.