CHART II
Software Development Guide

Version 1.0
Final

CHART Contract Number BCS99-30A
Work Order Number 4

Prepared For

Office of CHART
CHART Statewide Operations Center
7491 Connelley Drive
Hanover, MD 21076

Prepared By

PB Farradyne
3200 Tower Oaks Boulevard
Rockville, MD 20852

January 30, 2003

Table of Contents

List of Tables

List of Figures

Document Revision History

Date	Revision	Primary Author	Description
10/29/02	1.0 Draft	PB Farradyne	Initial document creation.
01/30/03	1.0 Final	PB Farradyne	Per MDSHA, added more technical detail in Section 6.1 for the Graphical User Interface and Section 7 for how to add code for new CHART II features.

1. Introduction

CHART (Coordinated Highways Action Response Team) is a joint effort of the Maryland Department of Transportation and the Maryland State Police, in cooperation with other federal, state and local agencies. CHART's mission is to improve "real-time" operations of Maryland's highway system through teamwork and technology. The CHART program relies on communication, coordination, and cooperation among agencies and disciplines, both within Maryland and with neighboring states, to foster the teamwork necessary to achieve transportation management goals. This is consistent with Maryland's State Highway Administration's (MDSHA) overall mission, which is to provide Maryland with an effective and efficient highway system.

The CHART program is Maryland's entry into the ITS (Intelligent Transportation System) arena, and started in the mid-1980's as the "Reach the Beach" initiative, focused on improving travel to and from Maryland's eastern shore. It has become so successful that it is now a multi-jurisdictional and multi-disciplinary program, and its activities have extended not just to the busy Baltimore-Washington Corridor, but into a statewide program. The program is directed by the CHART Board, consisting of senior technical and operational personnel from SHA, Maryland Transportation Authority, (MdTA), Maryland State Police (MSP), Federal Highway Administration (FHWA), the University of Maryland Center for Advanced Transportation Technology (UMD-CATT), and various local governments. The board is chaired by the Chief Engineer of the SHA. This comprehensive, advanced traffic management system is enhanced by a newly constructed state-of-the-art command and control center called the Statewide Operations Center (SOC). The SOC is the "hub" of the CHART system, functioning 24-hours-a-day, seven days a week, with satellite Traffic Operations Centers (TOC's) spread across the state to handle peak-period traffic.

This document is a Software Development Guide that discusses the methodologies, processes, tools, and other guidelines that were used for the development of CHART II software for the Maryland State Highway Administration. This information is provided for use by anyone wishing to modify the CHART II software as directed by MDSHA. This document is organized as follows:

- Section 1 presents an introduction to the CHART program and the research methodology used in developing this Guide.
- Section 2 discusses the evolution of the CHART II software and provides information on the CHART II software subsystems, multi-site deployment and CORBA (Common Object Request Broker Architecture) usage, and tools used in developing the CHART II software.
- Section 3 identifies the CHART II system documentation pertinent to software development activities.
- Section 4 provides an overview of the CHART II system development methodology and discusses key design principles and guidelines for the CHART II software development lifecycle. It also includes information on the usage of various CHART II tools and identifies the typical CHART II software development work products.
- Section 5 illustrates the CHART II development, testing, and operational environments.

- Section 6 provides additional guidance on key areas of the CHART II system, including the Graphical User Interface (GUI), system services, database organization, Field Management Station (FMS) implementation, simulator creation and usage, system and device logging, and interfaces to the CHART II web site.

- Section 7 provides guidance for adding new features to the CHART II system, such as adding a new device, adding a new model of an existing device, adding a new traffic event type, and adding a new algorithm to the system.

- Section 8 provides references to additional CHART II software development resources, including the CHART II developers web site, the CHART II web site Reading Room, and suggested third-party ITS and software documentation.

- Appendixes include a list of acronyms and descriptions of System Property Files.

The information in this Guide is current as of Release 1, Build 3 (R1B3) of the CHART II system, except where noted.

1.1 RESEARCH METHODOLOGY

The research methodology used to create the CHART II Software Development Guide was based on a four-step approach:

Task 1 – Define CHART II Software Development Areas for Analysis
This task defined the functional and architectural areas of the CHART II software to be analyzed and documented for the purpose of providing guidance on future CHART II software development efforts. CHART II software development personnel were consulted to ensure that all pertinent areas were included. The resulting areas provided a "roadmap" of the key topics to be addressed in the CHART II Software Development Guide.

Task 2 – Identify Existing CHART II Software Development Documentation
An abundance of documentation has already been produced during the course of CHART II software development. Through discussions with CHART II software development staff, the project team identified the existing documentation that is currently available to guide CHART II development in the areas defined in Task 1. The documents were reviewed and categorized, and references to the documents were made in the Guide. This task also included the development of a current listing of the standard requirements, design, and development tools/products used in developing CHART II system software.

Task 3 – Assess and Supplement Existing CHART II Software Development Documentation
This task concentrated on an assessment of the adequacy of the existing CHART II software development documentation and determined the need for additional/supplemental documentation. The project team conducted interviews with new CHART II software development staff, as this group was utilizing the existing CHART II documentation to develop new system functionality. The interviews focused on this group's software development approach, their experiences with the existing system documentation, any lessons learned, and suggestions for improvements to the documentation. Discussions with seasoned CHART II software development staff also occurred to gain additional information.

Task 4 – Develop Draft/Final CHART II Software Development Guide

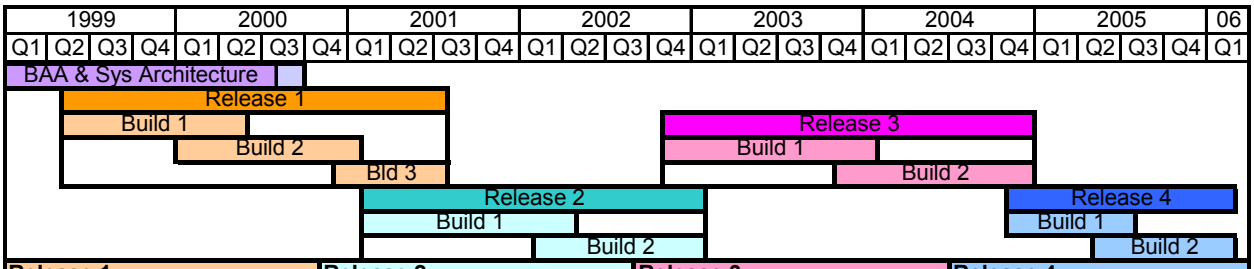
The findings of the research activities were documented in a Draft CHART II Software Development Guide. Before submission to MDSHA, this document was reviewed by selected CHART II software development staff members for accuracy and completeness. MDSHA staff reviewed the draft document and provided comments. Comments were addressed, and a Final CHART II Software Development Guide was submitted to MDSHA.

2. CHART II Software Background

CHART II was designed to meet Maryland SHA's need for a flexible, expandable, and integrated traffic management system that can support distributed operations. Using experience gained from previous traffic management software implementations, SHA provided an initial set of requirements for CHART II that were validated and enhanced by an extensive business architecture development effort. The results of this effort are documented in the CHART II Business Area Architecture (BAA) Report. The goals set forth for the CHART II system are as follows:

- Enable quick detection and response to traffic problems associated with incidents and special events.
- Foster communications with other State and local agencies and service providers to support implementation of effective, regional transportation management strategies.
- Work reliably and quickly diagnose and report system malfunctions.
- Is easy for operators to learn and use and enhances their productivity.
- Is scalable and easy to maintain.

The process used to develop the CHART II software included extensive review and consultation with the users of the system. Prototyping was also used extensively both to verify usability and to test technical approaches for complex functions. The system was developed using a Release-Build-Version model. The BAA identified four Releases. Releases 2 through 4 are planned to comprise two Builds each. Release 1 included three Builds. Each Build is designated to include a set of features defined in the BAA. As a Build progresses through the cycles of testing and review, a Version number is incremented. As of this writing, Release 1 Build 3 Version 10 is operational. Figure 1, below, shows the original CHART II Release/Build Schedule and distribution of functions across Releases. Note that this schedule has been altered subsequent to Release 1.



Release 1
Build 1:
- Basic Navigator & Admin
- DMS (4 types)
- Banned words
- Basic library & plans
- Operations log

Build 2:
- DMS -- 3 more types
- New FMS Architecture
- Acceptable Word List
- "CHART Chat"
- Com log & Event Log
- Event-based control

Build 3
- Highway Advisory Radio
- Shazam flashers
- Arbitration Queue

Web Highlights
- RTMS Data

Release 2
Build 1
- Map GUI
- Alerts & Escalation
- Notification (fax, page)
- System health monitor

Build 2
- Data export web interface
- EORS & SCAN interface
- Integrate AVCM

Web Highlights
- AVCM Integration Ph1
- CHART II Incident Integration
- Conditions-at-a-Glance
- MAA/EORS Interface
- SHADE
-US Wireless demo project

Release 3
Build 1
- Rules-based response
- FITM plans
- Interface to signals

Build 2
- Interface to AVL
- CHART Lite

Web Highlights
- AVCM Integration Ph2
- EORS wireless data export
- EORS/CHART II Interface
- Hand-held Data Collection
- AVL Pilot Project

Release 4
Build 1
- Additional external agencies

Build 2
- Simulation support
- Ad hoc archive queries

Web Highlights
- Shop portable inventory
- Pavement sensor data archive

Figure 1 – Original CHART II Release/Build Schedule

2.1 CHART II SOFTWARE SYSTEMS

The CHART II system consists of four major software systems:

1. CHART II Traffic Management System – This is the heart and brain of the CHART II system. It provides the operations interface and traffic management functions.
2. Field Management Station (FMS) – This system provides device communications and device data distribution functions for CHART II field devices.
3. Asynchronous Transfer Mode (ATM) Video Control Manager (AVCM) – This system provides CCTV camera control and video distribution services.
4. Archive – This system archives CHART II event and operations related data and provides query and reporting functions.

Figure 2, below, shows a high level view of the CHART II system.

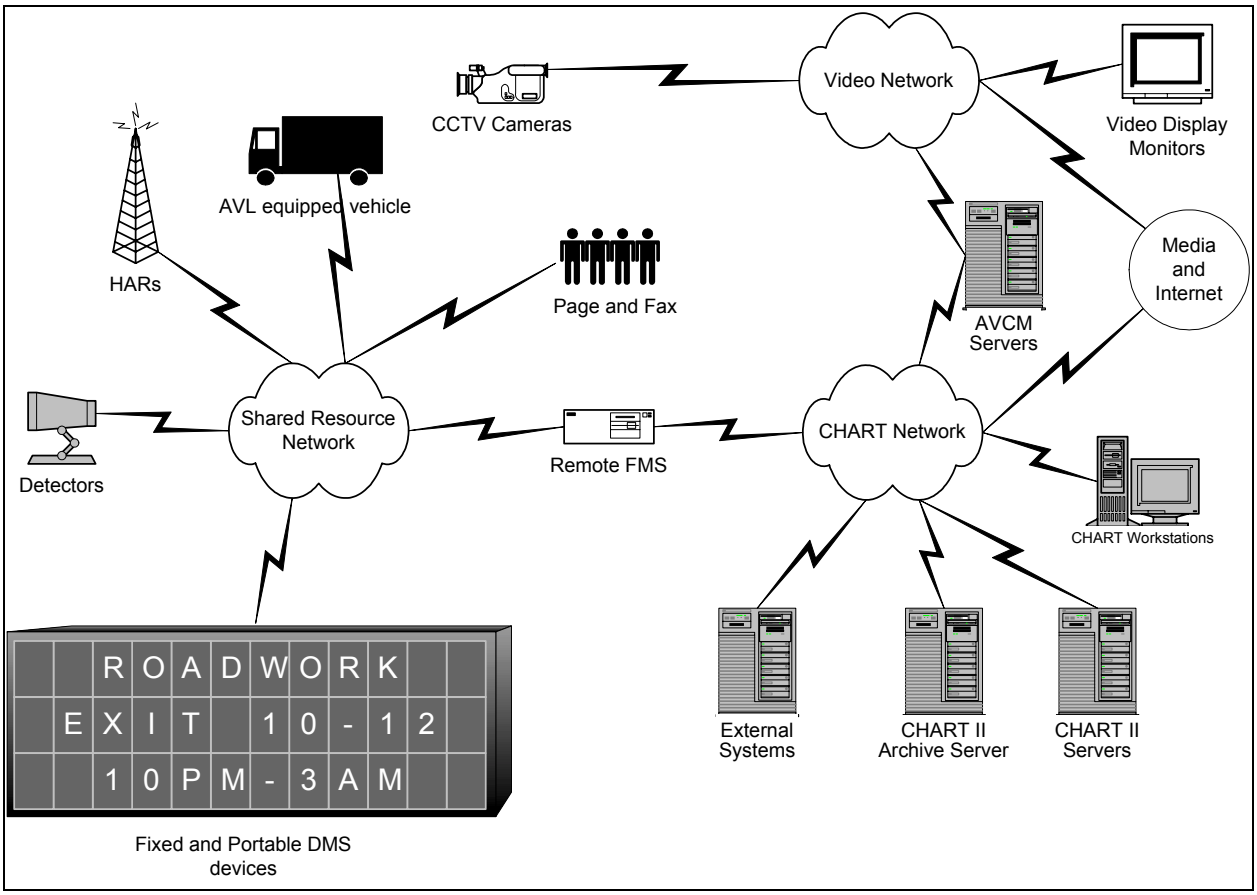


Figure 2 – CHART II System Overview

The major external interfaces to the CHART II system consist of:

1. Archival Data Users – Any user or external system that communicates with the CHART II Archive system.
2. CCTV Cameras – Field-deployed cameras for traffic monitoring.
3. CHART Web Server – Receives information from the CHART II system for publishing on the Web.
4. Emergency Operations Reporting System (EORS) – Legacy system providing information on road closures and road status.
5. Field Devices – Field-deployed traffic detectors, Dynamic Message Signs (DMS'), Highway Advisory Radios (HAR's), and SHAZAM's.
6. Media – Commercial and public broadcasters.
7. Other Agencies – Any other agencies or organizations receiving traffic or highway status information from the CHART II system but are not CHART II sites themselves. These include (but are not limited to):
   - I-95 Information Exchange Network (IEN)
   - TRANSCOM (Transportation Operations Coordinating Committee)

8. Notification Recipients – Recipients of FAX, page, or email alerts from the CHART II system.

9. SCAN (Surface Condition Analyzer) – SHA legacy system supplying weather sensor data.

10. Weather Information Suppliers – Sources for weather reports and alerts.

The high-level data flow diagram for the CHART II system is shown in Figure 3, below. Note that some of the data flows have not yet been developed (e.g., weather sensor data from the SCAN system to CHART).

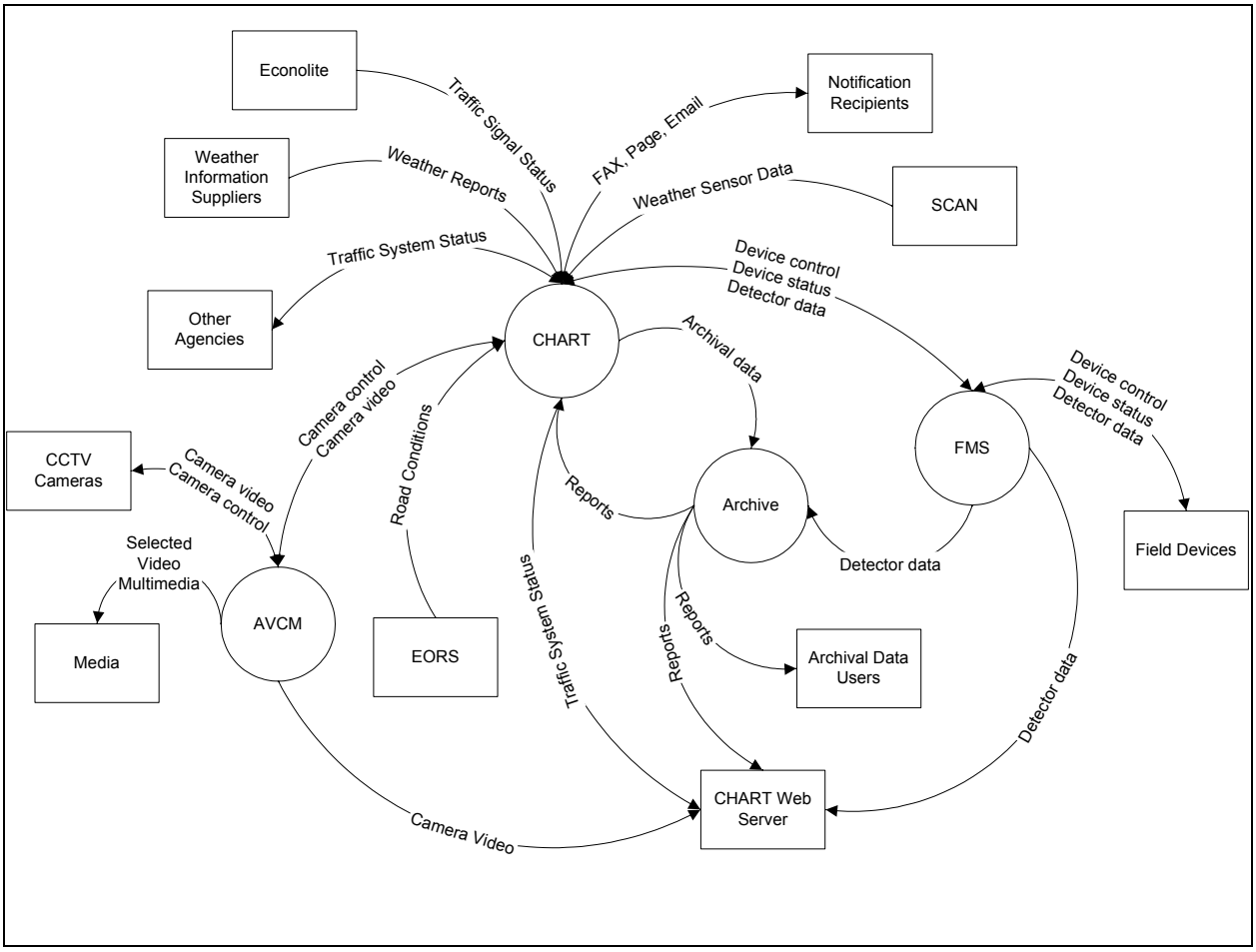


Figure 3 – CHART II System High-Level Data Flows

Note that all remaining references to CHART II in this document refer to the CHART II Traffic Management System.

The individual software subsystems comprising the CHART II system are briefly described below. Some of these subsystems are fully implemented, while others are currently only in the

design stage. The specific CHART II release/build in which the subsystem was implemented (or planned for implementation) is shown where applicable.

Those subsystems requiring direct interaction with the user will have both a server and client side component. The client side implements the GUI functionality required to convey or obtain from the user the required information.

Alert Management (planned for R2B1)
This subsystem provides alert management and processing functions. It provides the methods to support the creation and delivery of alerts and maintains the status of alerts in the system. Alerts may be automatically created by applications or manually created by users. Alerts may be directed to an operations center where acknowledgement by a user is required. Alerts may also be caught by an application for automatic processing (e.g., a weather sensor alert may initiate the creation of a weather sensor alert event by the Traffic Event Management subsystem and the sending of a notification by the Notification Management subsystem). Some example CHART II alerts are listed below.

- Device Failure – used to alert centers of device failures
- Transfer of Responsibility – provides an alert to the receiving center of a transfer of responsibility to that center (e.g., transfer of responsibility for an open event)
- Incident from Detector – alerts a center that detector data indicates a possible incident
- Mayday from AVL (Automatic Vehicle Location) – generated when an AVL equipped vehicle sends a Mayday message
- Weather Sensor – generated when a weather sensor reports data outside of a set range (e.g., temperature below freezing). Alerts that require a response within a specified time period are escalated up the center hierarchy if not acknowledged within the set time period.

The client side of alert management provides the user with the capability to manually generate an alert and to respond to alerts they receive.

Audio (R1B3)
This subsystem provides distributed access to a text-to-speech (TTS) engine that is utilized by the HAR (Highway Advisory Radio) subsystem for the conversion of text format messages into audible data that can be downloaded to the HAR device for broadcast. It also provides the ability to stream audio data back to requesting clients for message preview purposes.

AVL (planned)
This subsystem provides the interface between the AVL COTS (Commercial Off-The-Shelf) application and the CHART II system. It is responsible for obtaining vehicle position and status information from the AVL COTS application and providing a conduit for any two-way communications between an AVL equipped vehicle and the CHART system.

Communications Log Management (R1B3)
This subsystem provides a general logging mechanism for operators to record communications and activities in a central repository. The communications log replaces a paper log and is used to record all communications, including calls from the public, calls from CHART field units, other centers, etc. All recorded communications are made available to all other operators in near real-time through the user interface. The communications log also provides a filtered searching capability that allows an operator to select entries for viewing. Users may select entries to convert to a traffic event. These entries will become the base entries in the traffic event's history log.

Data Export Management (planned)
The Data Export Management subsystem provides a mechanism to make CHART data available to agencies that are not permitted or do not wish to obtain near real-time data via the CHART CORBA implementation. This subsystem will periodically, or on demand, generate XML (Extensible Markup Language) formatted data streams with pre-defined content. This data can be provided to external users through a web server or via FTP (file transfer protocol) from files in a data staging area.

Detector Management (R1B2)
This subsystem provides control and data handling functions for traffic detector and speed measurement devices. Historical data summaries are compiled and archived. Current traffic detector information is compared with historical traffic detector information, and alerts are generated for conditions exceeding specified tolerances.

Device Management (R1B2)
This subsystem handles the control of device state functions (online, offline, maintenance mode) and the management of device arbitration queue entries.

Dictionary (R1B2)
This subsystem provides administrator-managed collections of banned and known words. Banned words are those words that are not allowed to be displayed or broadcast on traveler information devices. Known words are used to provide spell checking and substitution suggestions when unknown words are detected.

DMS (Dynamic Message Sign) Control (R1B1)
This subsystem provides DMS control capabilities. It supplies support for multiple device manufacturer protocols and will expand to include NTCIP (National Transportation Communication ITS Protocol) support when these devices are acquired. In addition, this subsystem provides the business logic required for arbitration of a particular DMS between competing traffic events.

HAR (Highway Advisory Radio) Control (R1B3)
This subsystem provides HAR control capabilities. It supplies support for manufacturer protocols used by the MDSHA HAR devices. In addition, this subsystem provides the business logic required for arbitration of a particular HAR between competing traffic events.

HAR Notification (R1B3)
This subsystem provides management functions for the control of HAR notification devices such as SHAZAM's and DMS devices used as SHAZAM's.

Message Library Management (R1B2)
This subsystem provides message library management capabilities. It supports the creation of multiple message libraries for user defined stored messages, examples of which include DMS and HAR messages. Each message in a library can be assigned a category for user classification purposes.

Notification Management (design)
This subsystem provides capabilities for managing the notification of personnel via FAX, page, or email.

Plan Management (R1B2)
This subsystem provides the ability to create macro type collections of device control commands. Each item in a plan associates a stored message with a traveler information device. These plans can be used to quickly construct traffic event response plans for traffic events that are recurring in nature or can be planned for ahead of time.

Resource Management (R1B2)
This subsystem provides for management of user login sessions and the control of shared resources.

Schedule Management (planned for R2B1)
This subsystem supports the creation, management, and execution of lists of actions to be performed at predetermined times.

Signals (planned)
This subsystem provides an interface to the signals system in order to obtain traffic signal status information for use by the CHART II system.

Simulation (planned)
The Simulation subsystem is provided by the University of Maryland and integrates with the CHART II system.

System Monitor (R1B2)
This subsystem provides system health monitoring processes that are run on a periodic basis. Each service application is monitored to determine if it is currently available. Alerts are generated when services are found to be unavailable and self-recovery is attempted.

Traffic Event Management (R1B2)
This subsystem provides for the management and recording of information pertaining to traffic events that are currently being worked on by system operators. It also provides for the control of traveler information devices via a traffic event's response plan. The response plan is composed of system actions, including device control commands. When the plan is executed, the system actions are performed, and any device control actions result in an entry being placed on the arbitration queue for the target device. Each traffic event maintains a running history log of actions performed and user comments. Additionally, each traffic event maintains records of devices controlled, resources notified and utilized, and a list of related events for offline reporting and statistical analysis purposes.

User Manager (R1B1)
This subsystem provides the capability to create and manage user profiles and access rights.

Utility (various releases/builds)
The Utility subsystem provides various utility functions for the CHART II system and collects processes that do not have a home elsewhere. These include notepad management, the chat function, FITM (Freeway Incident Traffic Management) plan management, GIS (Geographic Information System) map update functions, etc.

2.2 MULTI-SITE DEPLOYMENT AND CORBA USAGE

The architecture for the CHART II system distributes complete system functionality to a number of districts throughout the State of Maryland. Each of these complete systems can provide full functionality for the devices connected to the system and objects created within that system (such as traffic events), and provides functionality for other district's systems that are available. Thus, the absence of one district's server does not affect the ability of another district to operate their own system or other systems that are available. Although the server deployment is spread across multiple sites, the GUI presents a view to the user of one large system, using CORBA to pull together objects served from the many deployment sites. Note that as of this writing, only two server sites have been deployed.

The GUI is able to locate the software objects at all deployment sites through the use of the CORBA Trading Service. As Figure 4, below, shows, a CORBA Trading Service exists at each deployment site. Each service that publishes CORBA objects offers the objects through its local CORBA Trading Service. Using the link feature of the CORBA Trading Service, each Trading Service is linked to all other Trading Services in the system. Each GUI is configured to utilize its local (or an assigned) Trading Service for object discovery. Through the use of linked (federated) Trading Services, the GUI discovers objects that are deployed on the same site as the Trading Service as well as objects published in all other trading services in the system. This allows the GUI to provide a unified view of the system, even though the system is actually distributed over multiple deployment sites.

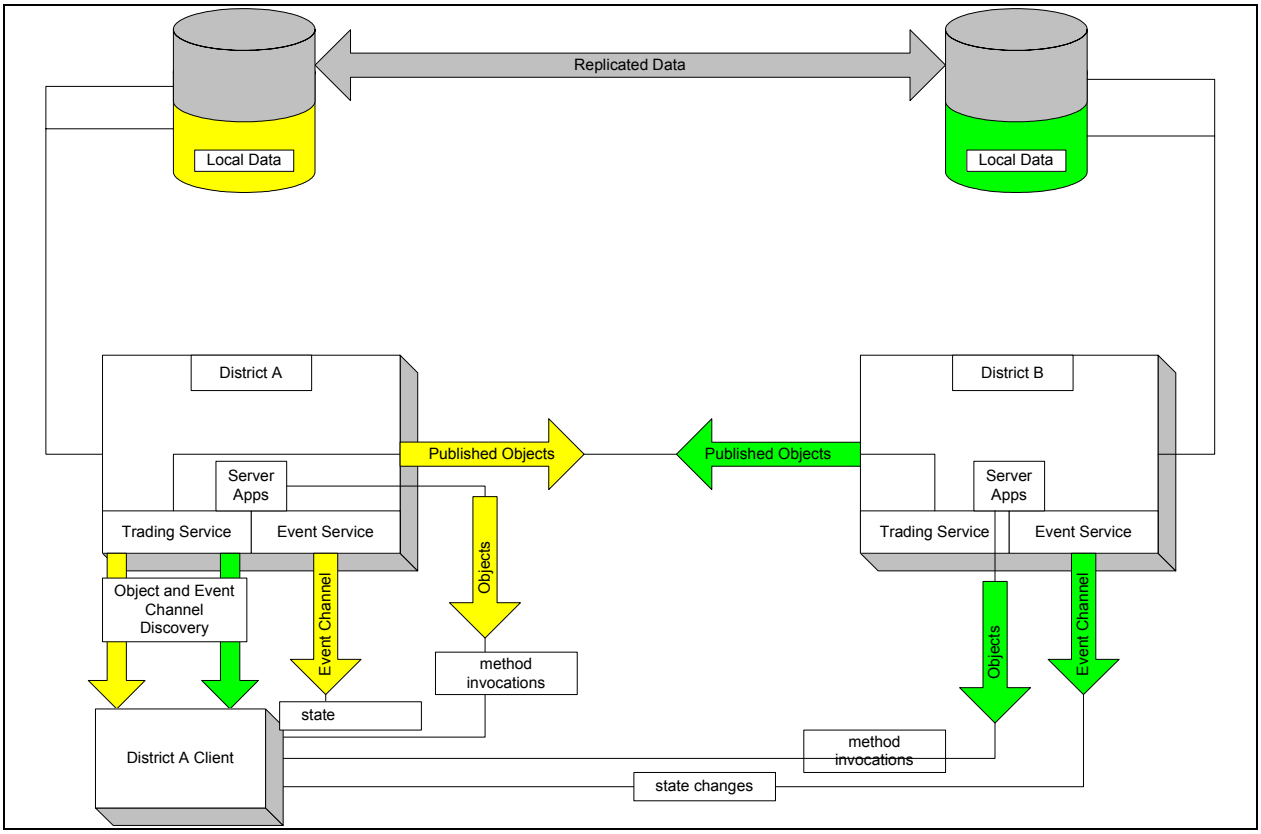


Figure 4 – CORBA Trading and Event Services

In addition to showing the software objects throughout the system on a single GUI, it is also necessary to reflect the current state of the software objects as they are changed during real-time operations. The CORBA Event Service is used to allow objects to push changes in their state to the GUI (or other interested CORBA clients). Each deployment site has an instance of a CORBA Event Channel Factory, which is an extension of the CORBA Event Service that allows multiple event channels. Each CHART II service whose objects are subject to real-time changes will create one or more Event Channels in its local Event Channel Factory. Each event channel is earmarked for a specific class of events (such as DMS events). Each service that creates channels in the CORBA Event Channel Factory publishes the event channel in the CORBA Trading Service and then uses the channel to push events relating to object state, alarms, etc.

Since the CORBA Event Service does not provide for a linking mechanism (such as that of the Trading Service), a GUI that wishes to listen for events at a system-wide level discovers all of the event channels via the CORBA Trading Service and registers itself as a consumer on each of the event channels. Using this scheme, a GUI uses the Trading Service to discover all software objects and Event Channels regardless of their deployment site. The GUI may then provide the user with a unified view of the system, both in the objects presented and the ability to show near real-time updates of these objects. Since the nature of the system is dynamic, processes discover new objects from known districts via event channels. However, the system relies on a periodic process to discover objects from new districts that have recently joined the system.

Most CHART II software objects used in this system are typical distributed software objects. Each of these objects is served from one and only one deployment site. The data inside an object pertains only to the instance of the object, and operations pertain only to the instance of the object on which they are performed. Other parts of the system (such as the GUI) must go to the instance of an object to view the object's data or perform operations (via method invocations) on the object. For example, there is one and only one software object in the system that represents a specific DMS in the field. If an operation such as setting the message needs to be done to the Field DMS, the GUI must perform the operations on the one and only software object that represents the DMS.

The system includes classes whose instances do not act as the typical objects described above. Instead, each instance of the class provides access to exactly the same data. Multiple instances of the class serve as replicated software objects. Some examples of this type of object are the Dictionary, UserManager, and Communications Log. These objects are different than the rest of the objects in the system because it is required that the dictionary, user data, and communications log be shared throughout all deployment sites in the system. Using the same dictionary data throughout the system provides consistency in messages displayed on DMS'. Using the same user data throughout the system allows a user to log in at any site, even in the event of a catastrophe at the user's normal operating site.

While the design could accomplish this use of shared data through using single instances of the objects, this type of design would include single points of failure. Thus, if the one and only one Dictionary object was unavailable, messages could not be placed on a DMS anywhere in the system since the message contents could not be checked for banned words. To overcome these single points of failure, the replication feature of the DBMS is used to replicate data to each deployment site's database. Each deployment site has its own instance of the Dictionary, UserManager, and Communications Log objects that front-end the replicated database. The

system takes advantage of these redundant objects by first attempting to retrieve the object from the client's home site. Failing that, it will fail-over to an alternate site's instance of the target object.

2.3 COTS TOOLS

Table 1, below, shows the COTS (Commercial Off-The-Shelf) software that has been used in developing the CHART II software.

COTS software development libraries (i.e., ORB (Object Request Broker), Java, and Oracle) are stored on the CHART2-SVR3\CORBA area on the CHART II server. Other shared software utilities are located on the CHART2-SVR4 and CHART2-SVR5 areas.

Table 1 – Tools Used for CHART II Software Development

Product	Version (as of R1B3)	Vendor	
Adobe Acrobat	4.0	Adobe Systems Inc.	Publishing Documents to CHART Web Site
Ant*	1.4.1	The Jakarta Project	Enforces Software Build Dependences (expected to be used post-R1B3)
ArcServeIT	6.5	Computer Associates (CA)	Backup Software
ClearCase Multisite	4.1+ (2000.02.10)	Rational Software	Configuration Management Tool for Document and Software Version Control
ClearQuest	2.0 (2001.03.00)	Rational Software	Problem Reporting, Risk Management Database, Deviation/Waiver Request Database, Action Item Database, COTS Upgrade Database
Dialogic System Software	5.0	Dialogic	Telephony board software
DOORS PC (Dynamic Object Oriented Requirements System)	5.2	Telelogic (acquired Quality Systems & Software (QSS))	Requirements Management
DOORSNet	4.1.4	Telelogic (acquired Quality Systems & Software (QSS))	DOORS Interface to Web
InstallShield*	5.0 (Multi-platform Professional 4.0 expected to be used post-R1B3)	InstallShield Software Corp.	Software Installation Management
Java Communications API	2.0	Sun Microsystems, Inc.	Java RS232 serial port support
JavaHelp	1.1	Sun Microsystems, Inc.	Help Authoring
JBuilder	3.0	Inprise Corp.	Java Coding. PRIMARY DEVELOPMENT TOOL.
JDK (Java Developer's Kit)	1.3.1 (1.4.0 expected to be used post-R1B3)	Sun Microsystems, Inc.	Java Runtime Environment

Product	Version (as of R1B3)	Vendor	Usage
JProbe	3.0	KL Group	Developer Tool for Detecting Memory Leaks and Other Code Refinement
JTest	4.0	Parasoft	Unit Level Testing Tool
Krakatau Professional	2.2	Power Software	Source Code Evaluation Tool. Run by JHU/APL and output sent to Development Team (on request basis) – done at end of R1B3, only used once or twice.
MS Access	97	Microsoft Corp.	Database Reports
MS Excel	97	Microsoft Corp.	Change Suggestion Form, Metrics Spreadsheet, Database Reports
MS Project	2000	Microsoft Corp.	Software Project Scheduling
MS Visual C++	6.1	Microsoft Corp.	C++ Coding (used for application modeling and prototyping)
MS Word	97	Microsoft Corp.	Documentation, User Manual
Notification Service*	1.0 (2.0 expected to be used post-R1B3)	IONA	CORBA Service
OMake	V4.1+	Rational Software	Enforces Build Dependencies (used through R1B3)
Oracle Designer	6.0.3.6.0	Oracle Corp.	Database Design (Entity-Relationship Diagrams, Table Definition Report)
Oracle Enterprise Manager	2.2.0.0.0 with patch EM_22_1747199	Oracle Corp.	Database Maintenance
Oracle RDBMS	8.1.5.1.1 for Windows NT	Oracle Corp.	Database Management System
ORBacus*#	JOB-4.0.5 (JOB-4.1.0 expected to be used post-R1B3)	IONA	CORBA Development Tool
RealSpeak	1.10.00	Lernout and Hauspie	Text-to-Speech Conversion
SQLPlus	8.1.5.0.0	Open Source (ships with Oracle)	Database Management System
Tau UML Suite (formerly Cool:Business Team and Cool:Jex)	4.6	Telelogic (acquired Sterling Software)	Object Oriented Design and Business Process Re-Engineering
TCL	7.6	Open Source (ships with Oracle)	Database Management System
Textpad	4.0	Helios Software Solutions	Basic Integrated Development Environment for Java Coding
UNICenter TNG Remote Control	5.2	Computer Associates	Tool used to remotely control a workstation or server
Windows NT	4.0 Service Pack 5	Microsoft Corp.	Standard operating system for CHART II Servers and Workstations

* These tools are in the process of being upgraded. See the Version column for the expected post-R1B3 version.

Due to the high license cost of the ORBacus CORBA development tool ($65,000+), the PB Farradyne software development team has been investigating the use of an open source ORB, but MDSHA has not yet decided to move towards a new ORB product for CHART II. **It is strongly encouraged for CHART II software development teams to consult with MDSHA, who may have ORB licenses available for use.**

3. Existing CHART II System Documentation

An abundance of documentation has been produced during the course of CHART II software development, representing both past efforts and those activities currently in progress. A shared directory structure was established on one of the CHART II servers (CHART2-SVR1\MODELS) to organize this material for CHART II project staff, as shown in Table 2, below. The first-level directories established are shown for all categories of documentation, and second-level directories are shown only where the materials therein are widely useful to the project team.

Table 2 – CHART II Server Directory Structure

1st Level	Significant 2nd Level	Content
Administrative	Distribution Lists	Email addresses for notifications of document review and approval, POC (point-of-contact list)
	Program Forms	Disbursement requests, travel authorization requests, etc.
Tool Reports & Data	Business Team	Business Team materials by release
	Object Team	Analysis information by release/build
Methodology	Catalyst	CSC (Computer Sciences Corporation) Sources Toolkit, reference materials
Delivery		CD Images and Delivery Packages for each delivered system
Sites & Configurations	CI List	Current CHART II Configured Item (CI) list
	CHART Network	As-built and survey information
	Hardware Inventory	In-scope sites and equipment
	Operational Installations	CHART II and remote FMS installations
Project Management Materials	Inspection & Peer Review Records	Records of inspections and peer reviews of code, IDL (Interface Definition Language), design, etc. for each build/release
	Project Action Items	Records of open and closed action items
	QA Assessments & Audits	Materials related to CMM (Capability Maturity Model) assessments, FCA's (Functional Configuration Audits) and PCA's (Physical Configuration Audits), life-cycle audits, and schedule reviews
	Schedule	Master schedule (archive, baseline, in work, and previous baseline versions)
	Monthly Management Status Reviews	Materials related to monthly task management reviews
Meetings & Presentations	Minutes & Agreements	Records of all meetings and significant agreements
	MITS	Minutes and presentations related to the MD ITS Working Group
	Technical Presentations	Presentations on CHART II design, requirements, etc.
	CHART II Client Monthlies	Materials related to CHART II monthly meetings with customer
	CHART II Client Weeklies	Materials related to CHART II weekly meetings with customer
Testing		Testing summaries, etc.
Training		Training plans and materials

1st Level	Significant 2nd Level	Content
Database		Design and load documentation as well as migration and installation information, etc.
Project Documentation	Current Approved	Current approved versions of project documentation
	Draft	In-work versions of project documentation
	Old Approved	Old approved versions of project documentation
Project Standards & Procedures	Current Approved	Current approved versions of project S&P's (standards and procedures)
	Draft	In work versions of project S&P's
	Old Approved	Old approved versions of project S&P's
Reference	RFP	Original request for proposal (RFP)
	Oracle	Oracle reference downloads
Transfer		Files to be made available to other project members
Vision and Strategy		All materials related to Visioning

Table 3, below, lists the pertinent documentation on the CHART II server along with the group(s) responsible for creating/maintaining each document and the network location and date of the file. Note that the N: drive shown for the document location is mapped to the CHART2-SVR1\MODELS area on the CHART II server.

Table 3 – Existing CHART II System Documentation

Document/Deliverable	Responsibility	Location	Date
Requirements			
CHART II System Architecture	System Architect, Development Team	• N:\Project Documentation\Current Approved\Design\System Architecture\CHART System Architecture.doc • www.chart.state.md.us/readingroom/readingroom.asp	• 9/5/00
Business Area Architecture Report	Business Architect	• N:\Project Documentation\Current Approved\Requirements\BAA Report\chart2-final-baa_Aug31.doc	• 8/22/00
System Requirements	System Architect	• See DOORS PC requirements tool on CHART2-SVR4 server. • N:\Project Documentation\Current Approved\Requirements\System Requirements\CHART II System Requirements.doc	• N/A • 5/5/00
Design			
High-Level Design (each design document includes only the changes from the previous release so the entire chain of documents is listed here)	Development Team	• See Tau UML Suite on CHART2-SVR5 server. • N:\Project Documentation\Old Approved\Design\High Level Design\R1B1\Server High Level Design\R1B1HighLevelDesign.doc • N:\Project Documentation\Old Approved\Design\High Level Design\R1B1\GUI High Level Design\R1B1GUIHighLevelDesign.doc • N:\Project Documentation\Old Approved\Design\High Level Design\R1B2\R1B2 High Level Design\R1B2HighLevelDesign.doc • N:\Project Documentation\Old Approved\Design\High Level Design\R1B2a\R1B2A High Level Design\R1B2A HighLevelDesign.doc • N:\Project Documentation\Current Approved\Design\High Level Design\R1B3\R1B3HighLevelDesign.doc	• N/A • 7/16/99 • 1/21/00 • 4/17/00 • 10/16/00 • 1/16/01

Document/Deliverable	Responsibility	Location	Date
Detailed Design (each design document includes only the changes from the previous release so the entire chain of documents is listed here)	Development Team	• See Tau UML Suite on CHART2-SVR5 server. • N:\Project Documentation\Old Approved\Design\GUI Detailed Design\R1B1\R1B1GUIDetailedDesign.doc • N:\Project Documentation\Old Approved\Design\Server Detailed Design\R1B1\R1B1DetailedDesign.doc • N:\Project Documentation\Old Approved\Design\GUI Detailed Design\R1B2\R1B2GUIDetailedDesign-final.doc • N:\Project Documentation\Old Approved\Design\Server Detailed Design\R1B2\R1B2ServersDetailedDesign-final.doc • N:\Project Documentation\Old Approved\Design\Detailed Design\R1B2A\R1B2A Detailed Design\R1B2ADetailedDesign.doc • N:\Project Documentation\Current Approved\Design\GUI Detailed Design\R1B3\R1B3GUIDetailedDesign.doc • N:\Project Documentation\Current Approved\Design\Server Detailed Design\R1B3\R1B3 Servers Detailed Design.doc	• N/A • 1/21/00 • 1/21/00 • 7/19/00 • 5/26/00 • 10/26/00 • 3/16/01 • 3/16/01
Logical Database Design	Database Team	• N:\Database\Database Docs\R1B3 DB Design\Logical_diagram_c2arch3.doc	• 1/14/02
Physical Database Design	Database Team	• N:\Database\Database Docs\R1B3 DB Design\Physical_diagram_c2arch3.doc	• 1/14/02
Software Development			
Software Development Plan	Development Team	• N:\Project Documentation\Current Approved\Management Plans\Software Development Plan Rev2\CHART II SDP Rev3.doc	• 3/21/02
Java Software Coding/Implementation	Development Team	• N:\Project Standards & Procedures\Current Approved\Software Development\Coding Standards\Java Coding Standard.doc	• 10/6/99
IDL Coding	Development Team	• N:\Project Standards & Procedures\Current Approved\Software Development\Coding Standards\IDL Coding Standard.doc	• 7/15/99
C++ and Visual C++ Coding/Implementation	Development Team	• N:\Project Standards & Procedures\Current Approved\Software Development\Coding Standards\C++ Coding Standards.doc	• 7/2/99
Sample Unit Test Plan	Development Team	• N:\Project Management Materials\Inspection & Peer Review Records\Chart\Unit Testing\Test Plans\TSSMgmtTestPlan.xls	• 11/29/00
Integration Test Plan (includes regression testing for previous versions)	Development Team, Database Administrator	• See ClearQuest tool on CHART2-SVR4 server. • N:\Project Documentation\Current Approved\Integration Testing\R1B3\R1B3 Integration Test Plan.doc	• N/A • 9/30/02
System and Acceptance Testing			

Document/Deliverable	Responsibility	Location	Date
System Test Readiness Review	System Test Team	• N:\Testing\SystemTesting\Readiness Reviews\R1B3\R1B3.03 systest RR Report.doc	• 11/15/01
System Test Plan	System Test Team	• N:\Project Documentation\Draft\System and Acceptance Testing\R1B3\Test Plan\R1B3 Test Plan Archive\CHART II R1B3 Test Plan.doc	• 5/15/01
System Test Procedure	System Test Team	• N:\Project Documentation\Current Approved\System and Acceptance Testing\Test Procedures\R1B3 Test Procedures.doc	• 12/7/01
System Test Report	System Test Team	• N:\Project Documentation\Current Approved\System and Acceptance Testing\Test Reports\R1B3 Test Report\R1B3 System Test Report.doc	• 1/18/02
Acceptance Test Readiness Review	System Test Team	• N:\Testing\AcceptanceTesting\Readiness Reviews\R1B3\R1B3.10 ORR Report 030602.doc	• 3/6/02
Acceptance Test Plan	System Test Team	• N:\Project Documentation\Current Approved\System and Acceptance Testing\Test Plans\CHART II R1B3 Test Plan.doc	• 11/6/01
Acceptance Test Procedure	System Test Team	• N:\Project Documentation\Current Approved\System and Acceptance Testing\Test Procedures\R1B3 Test Procedures.doc	• 12/7/01
Acceptance Test Report	System Test Team	• N:\Project Documentation\Current Approved\System and Acceptance Testing\Test Reports\R1B3 Test Report\R1B3 System Test Report.doc	• 1/18/02
Defect Tracking	CM	• See ClearQuest tool on CHART2-SVR4 server.	• N/A
Software Control Notice (SCN)	CM	• N:\Delivery\R1B3_10 Delivery Package	• 2/25/02
Transition to Operations and Deployment			
Transition Readiness Review	Transition Team	• N:\Meetings & Presentations\Technical Presentations\R1B3 Transition Readiness Review\20020111\TRR.ppt	• 1/11/02
Operations Readiness Review	Transition Team	• N:\Meetings & Presentations\Technical Presentations\R1B3 ORR\ORR.ppt	• 2/5/02
Transition Plan	Transition Team	• N:\Project Documentation\Current Approved\Transition Plans\Transition Plan\R1B3 Transition Plan\ R1B3 Transition Plan.doc	• 12/6/01
Data Migration Plan	Database Administrator	• N:\Project Documentation\Current Approved\Transition Plans\Data Migration Plan\R1B3 Data Migration Plan\R1B3 Data Migration Plan.doc	• 8/31/01
DBA Guide	Database Administrator, Documentation Team	• See Operations and Maintenance Guide (Section 3.9 and Appendix A.2). • Also see N:\Testing\Acceptance Testing\SCN's\Transition R1B3.09\Attachment L – Database Instructions.doc	• 1/11/02
User Manual			
Operations and Maintenance Guide	System Engineering	• N:\Project Documentation\Current Approved\Operations\Operations and Maintenance Guide\R1B3 Operations and Maintenance Guide R1.doc	• 2/12/02

Document/Deliverable	Responsibility	Location	Date
Delivery CD	Task Manager, CM	• N:\Delivery\R1B3_10 CD Image	• 2/25/02
Training Plan	System Test Team	• N:\Project Documentation\Current Approved\Training Plans and Materials\Training Plan\R1B3 Training Plan.doc	• 5/17/01
Training - Administrator	System Test Team	• N:\Project Documentation\Current Approved\Training Plans and Materials\Training Materials\R1B3 Admin Course\Admin.ppt	• 10/26/01
Training - Operator	System Test Team	• N:\Project Documentation\Current Approved\Training Plans and Materials\Training Materials\R1B3 Operator Course\operator R1B3.ppt	• 10/25/01
Configuration Management			
System Problem Reporting and Tracking	CM	• See ClearQuest tool on CHART2-SVR4 server.	• N/A
Project Planning			
Risk Management Procedure	Task Manager	• N:\Meetings and Presentations\CHART II Client Monthlies\CHART II Monthly Meeting Presentations\2000\10 October\riskdb-cq-oct30.00.htm	• 10/30/00

4. Software Development Lifecycle Guidelines

4.1 CATALYST METHODOLOGY

Catalyst is Computer Sciences Corporation's (CSC) proprietary structured methodology. It was used to implement the technical and management approach for the CHART II project. A total methodology for business change and for complex system development, Catalyst facilitates and guides application system development, integration, deployment, and operational services. It provides the formal structure for CHART II software development.

CSC developed the Catalyst methodology and has used Catalyst for over 19 years to support a wide range of commercial and Government customers. Catalyst supports the criteria of Carnegie Mellon University's Software Engineering Institute (SEI) Capability Maturity Model (CMM) for Software Development.

Phases are Catalyst's structure for sequencing and describing the work of business change and system development. They are a convenient mechanism for establishing an overall sequence, are the basis for planning and estimating, facilitate project- and program-wide synchronization of activities, and chart decision points for management. There are six phases:

- Five for system development – Vision and Strategy, Architecture, Development, Integration, and Deployment.
- One for system operations – Operational Services, which covers all aspects of operations and maintenance support.

Paths are Catalyst's approach to the development and maintenance of application systems. Separate paths flow horizontally through the architecture and development phases. Each path satisfies different customer needs and initial conditions for the effort.

See Figure 5, below, for a graphical depiction of the Catalyst methodology.

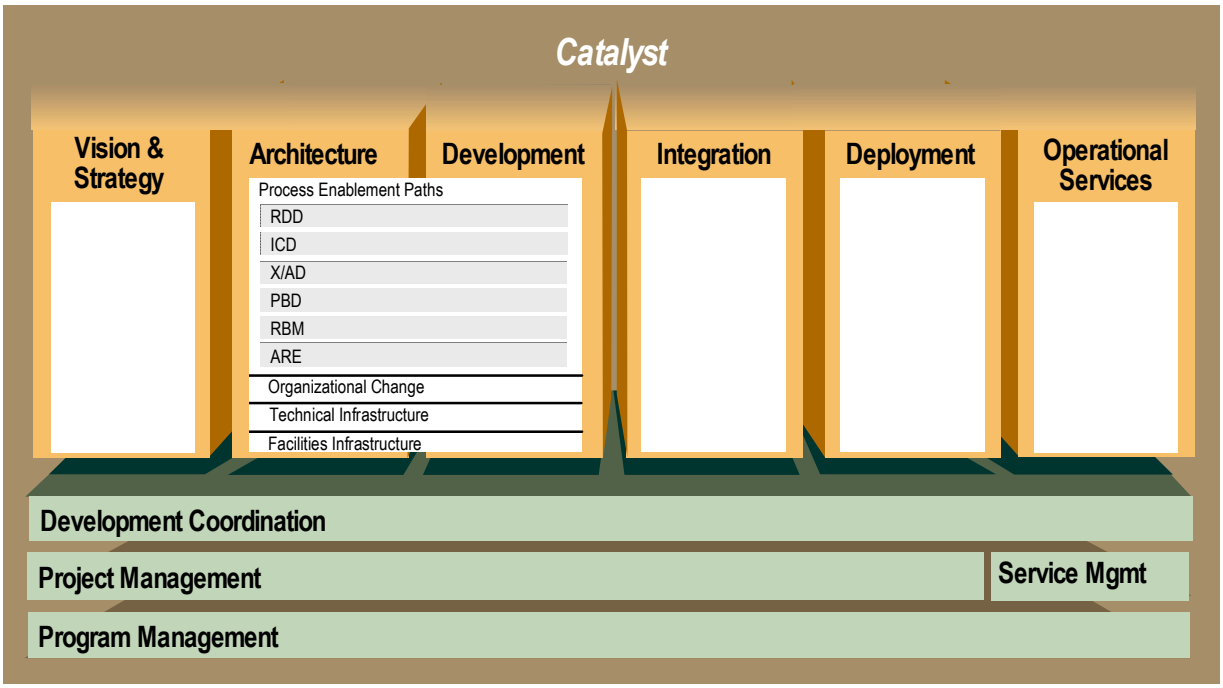


Figure 5 – Catalyst Methodology

For CHART II, the combination of two process enablement paths are employed to meet operational needs – accelerated application development and iterative custom development:

- **Accelerated Application Development.** X/AD is the Catalyst path to rapid application development. It combines workshops, labs, prototyping, full-time user participation, and timebox management in an approach that rapidly develops a prototype and transforms it into a production application.

- **Iterative Custom Development.** ICD designs business processes at a detailed level and develops new application software on a custom basis. While it is somewhat more conventional than X/AD, it is by no means old-fashioned. It uses facilitated workshops and prototyping to develop requirements, to design the user system interface, to validate system functions, and to produce a working prototype. It provides for application armor plating and testing to occur either in parallel with prototyping or in a separate sub-phase after prototyping has been finished.

Further information about Catalyst is available on the CHART2-SVR1 server in the MODELS\PROJECT DOCUMENTATION\CURRENT APPROVED\ PROJECT MASTER PLAN.DOC file.

4.2 KEY DESIGN PRINCIPLES

A number of key principles were developed and adopted in designing and developing the CHART II software:

- Exception Processing

- Long Running Operations
- Access Control
- Parameter-Driven System Properties

4.2.1 Exception Processing

Since CHART II is a distributed object system, it is expected that any call to a remote object could cause an exception to be thrown. The system provides two levels of exception handling. The first is aimed at providing the user with immediate feedback on the failure status of the requested operation. The second is aimed at maintaining a log of system errors to enable system administrators to trace and correct problems. Each application maintains a running log file of software system status. Exceptions thrown by the applications contain a user displayable text status and a more detailed debug text status that is recorded in the application log file.

4.2.2 Long Running Operations

Many device control operations cannot be executed immediately. In some cases, this might cause the operator to suspect the operation failed when it has not. Therefore, the software has been designed to perform these operations in an asynchronous fashion. The initiator of a long running operation is provided the opportunity to supply a callback status object. This allows the application to supply progress information back to the initiating client as the operation proceeds. Each operation provides a final status that indicates overall success or failure. A typical example is putting a message on a device such as a DMS. The system dials up the device, obtains a connection between modems, confirms that the DMS controller is responding, sends the message to the device, and finally disconnects the communications path. At each point in this process, status information is available to the initiator via the callback status object. This allows, for example, the display of a progress window in the GUI to inform an operator of the status of their request to put a message on a DMS. The operator is able to see the progression from dialing to communicating to successful response and finally, terminating the connection.

4.2.3 Access Control

Users gain access to the system through a login process. As a result of this process, they are provided an access token which contains a description of the functional rights that the user has previously been granted by a system administrator. The token also contains information describing the operations center that they are acting on behalf of. Each restricted system operation requires this token to be passed for functional right verification purposes. If the token contains the appropriate functional right to perform the operation, the system will then verify that the user is logged in to the operations center that is currently responsible for any targeted shared resources.

The system provides for the concept of a shared resource. A shared resource is any resource that can be owned by a particular organization and is only allowed to be controlled by one operations center at a time. Access to a shared resource is controlled through the functional rights of the user attempting to gain control of the resource and through an arbitration scheme that prioritizes requests to the resource.

4.2.4 Parameter-Driven System Properties

Each service in the system has a Properties File (also called a Props File) that controls its behavior. This Java-standard approach of putting service parameters and their values in an ASCII file provides a convenient way for the system administrator to document and change the behavior of a service at start-up. Props Files should only be used to specify parameters that cannot change while the service is running. The service must be restarted for any changes to its Props File to take effect.

Property Files are preferred over storing these values in a database because a database is not always available. Also, ASCII files facilitate documentation of system properties.

See Appendix B for a listing of System Property Files and associated contents.

4.3 HIGH-LEVEL DESIGN

Guidelines for the high-level design of the CHART II software are shown below.

High-Level Design Goals

- Gain a full understanding of the requirements for the release and verify that understanding with the client/end user.
- Choose between competing technologies for implementation of requirements.
- Determine and document interfaces between system components.
- Determine and document deployment strategy.
- Describe user (functional) rights scheme. What rights grant access to which features.

Process

- Use Case Design
- Usability Design
- Architectural Prototyping
- Class Diagrams and Sequence Diagrams
- Other Diagrams
- Internal Reviews
- Document Key Design Concepts
- External Design Review

Use Case Design

- Every requirement must map to a use case.
    - Verify requirement coverage.

- Every use case must have at least one requirement mapped to it.
    - Eliminate scope creep.
    - Forces us to update requirements to match what we plan to do. This aids the test team later.

- Each use case and actor must have descriptive text that describes, in user language, the purpose of the use case.
    - Facilitates agreement between Development Team and client on interpretation of requirements.

- Use case design leads to requirements questions.
    - Questions must be answered, and requirements/use cases/mapping must be updated.

Usability Design

- Screen Captures
    - Provide screen mock-ups for new windows.
    - Aids in customer understanding of system.
    - Aids designers by reminding them of what data needs to be available.

- Usability Prototyping
    - Used for highly complex interfaces such as map, traffic event dialogs, etc.

Architectural Prototyping

- Used for proof of technology/concept.
    - Quick functional program written to prove concept.
    - Not code that will be used in final system.
    - Not usually shown to client.
    - Past examples
        - CORBA architecture
        - Java map feasibility (OpenGL)
        - Audio streaming
        - Field Communications Architecture (Port Managers)

Class/Sequence Diagrams

- Create class diagrams that contain potential interfaces and major classes.

- Create sequence diagrams.
    - At least one sequence diagram for every use case.
    - Leads to changes in class diagrams as new interface needs are discovered.
    - Show how major classes/interfaces collaborate to accomplish requirements of use case.

Other Useful Diagrams

- Deployment Diagram
    - Shows which components/objects are expected to be running on which nodes in the deployed system.

- Packaging Diagram (Component Diagram)
    - Used to document packaging of classes/interfaces for the release.

- State Diagrams
    - Used to document states an object can assume and stimuli that can cause state transitions.

- Activity Diagrams

Internal Reviews

- Full team review of use cases and mapping to requirements.

    - Facilitates team understanding of what the system needs to do.

- Full team review of all sequence/class diagrams.
    - Facilitates collaboration and team buy-in.

External Review
- Produce Design Document.
    - Handwritten introduction including description of key design concepts.
    - Create functional rights matrix.
    - Pull all diagrams and text for actors, classes, use cases, etc, into Models section of the design document.

- Produce Design Review Presentation.
    - Pick 3-5 major sequence diagrams to walk through.
    - Pick 2-3 major class relationships to highlight from class diagrams.

- Design Review Meeting.
    - Presentation of design to client, IV&V (Independent Validation and Verification), DB team, QA team, project management, etc.

4.4 DETAILED DESIGN

Guidelines for the detailed design of the CHART II software are shown below.

Detailed Design Goals
- Provide sufficient details to allow developers to quickly/easily implement release.
- Document intended implementation to allow wider scrutiny/exposure.
- Identify risky and difficult portions of the system to implement.
- Identify exact dependencies on outside designs such as database interface or external systems.
- Identify utility classes/methods that should be moved to common packages for use by multiple system components.

Work Products
- GUI Detailed Design Document
    - Key design concepts
    - Package descriptions
    - Class diagram per package
    - Sequence diagram per non-trivial method in a class within a package
    - Notes describing user interactions with system

- Server Detailed Design Document
    - Key design concepts
    - Package descriptions
    - Class diagram per package
    - Sequence diagrams
        - One for each non-trivial method of a class within the package.

Process

- Class and sequence diagrams for server modules
- Class and sequence diagrams for GUI modules
- Other diagrams
- Internal reviews
- Produce documents
- External reviews

Diagrams for Server Modules

- Create a class diagram for each server module needed.
    - Module class implements framework interface.
    - Properties class (if necessary) to front-end the getting of system/application properties.
    - DB class to provide interface to DB.
    - IMPL classes to implement IDL interfaces.

- Provide sequence diagrams for non-trivial IDL and framework interface methods.
    - Startup/shutdown module interface methods should have diagrams.
    - CORBA ORB should be invoking actor on sequence diagrams detailing IDL implementation methods.
    - Diagrams should show all calls necessary to implement method and should show all possible returns/exceptions.
    - Diagrams should result in full documentation of DB class interface, which should be coordinated with DB team frequently.

Diagrams for GUI Modules

- Create a class diagram for each GUI module needed.
    - Module class implements application framework interface.
    - Properties class (if necessary) to front-end the getting of system/application properties.
    - If module is large, can be helpful to show one class diagram for module and "wrapper" classes and another class diagram for dialog classes, etc.

- Provide sequence diagrams for non-trivial IDL calls and framework interface methods.
    - Startup/shutdown/discovery should have sequence diagrams.
    - Sequence diagrams for all IDL methods.
        - should have user as actor.
        - should show class/user interactions needed to build data structures for IDL call.
        - should culminate in CORBA call to IDL method of interface class and should show handling of all possible returns/exceptions.

Other Diagrams

- Creation of sequence diagrams should lead to identification of utility classes.
    - Result in creation of new class diagrams or update of existing class diagrams.
    - Often results in additional sequence diagrams to show implementation details for utility.

- Use any other UML (Unified Modeling Language) based diagrams to provide documentation and analysis of how the implementation should be done.

Internal Reviews
- Diagrams for each package must be reviewed.
    - Each class/sequence diagram must have descriptive text.
    - Each class must have descriptive text.
    - Each method on sequence diagram must be shown as method of class on class diagram.
    - To call a method of a class, the calling class must have access to an object (or call must be static).
        - Class diagram must show a relationship between the classes, or
        - Sequence diagram must show a call to a related class that returns an instance of the class to be used.
    - Synchronization/threading should be checked closely.

Produce Documents
- Write base document for GUI and Server detailed design documents.
    - Package descriptions
    - Key design concepts

- Pull models
    - Pull class and sequence diagrams for each package into document.

External Review
- Distribute documents for external review.

- Produce Design Review Presentations (one for GUI, one for Servers).
    - Pick 3-5 major sequence diagrams to walk through.
    - Pick 2-3 major class relationships to highlight from class diagrams.

- Design Review Meeting
    - Presentation of design to client, IV&V, DB team, QA team, project management, etc.

4.5 IMPLEMENTATION

Guidelines for the development of the CHART II software are shown below.

Implementation Goals
- High Quality
    - Unit tested with documented test plan.
    - Reviewed by peer(s).
- Maintainable
    - Well documented.
    - Matches design to maximum possible extent.
    - Conforms to coding standards.
- On time/within budget
    - Should always be aware of time constraints for current task.

Process
- Code/unit test iteratively.

- Integrate into system build when code is ready for initial check-in.
- Code Review.
- Bug Fixing.

Coding/Unit Testing

- Code to the design
    - The design should be referenced during every step of coding.
    - Variance from the design must be discussed with Team Lead.

- Get code running early
    - Implement startup/shutdown methods.
    - Provide stubbed implementations where necessary.

- Unit test as you go
    - When a new feature is coded, unit test it and document the test cases/results in the unit test plan.

- Inform Team Lead early if running behind schedule.

Integration

- Add new code/package to system build.
    - If new class in existing package, check in new make file.
    - If new package, add to \build\makefile.mak twice (once for packages, once for javadoc).
    - If new server, check in jar script, props file and template props files as well.
    - Checked in version of code must build from this point forward.

- Make new features available for others.
    - Work with Team Lead to make new features available for others to use.
        - Add module or service to set of running services on public server.
        - Notify others of existence of new features so they can use them and provide feedback.
    - Periodically update “released” version as coding is completed on additional features.

Code Reviews

- Submit code for review
    - Include completed unit test plan.

- Reviewer should look for
    - Correctness of code.
    - Conformance to design.
    - Conformance to coding standard.
    - Thoroughness of unit testing.

Bug Fixing

- Level C database in ClearQuest.
- Bugs found when using “released” features from “integration” environment are submitted.
- Project has documented procedure for handling Level C bugs.

ClearCase (CM tool) Usage
- Coding done in local tip view.
- Add of new files to be approved by Team Lead.
- Check-in code during development or when complete.
- Check-in comment must contain release and build or PR (problem report) number from ClearQuest.
- Checked-in code must build in tip view.
- Apply UNIT_TESTED label on the version of code that has been unit tested and updated after code review.
- Check build in UNIT_TESTED view after labeling code.

4.6 TESTING

Guidelines for integration, system, and acceptance testing of CHART II software are shown below.

Integration Test
- Tests the interaction of components.
- Installation on local systems.
- Test Cases written by developers.
- Test Cases executed by developers.
- Multiple iterations.
- ClearQuest (problem reporting tool) Usage
    - ClearQuest database used to track bugs – Level C.
    - Bugs and enhancements entered by any developer, assign to lead.
    - Lead does analysis, assigns to developer.
    - Developer does further analysis, provides a fix.
    - When code is ready, developer moves PR to CODE_COMPLETED.
    - Lead includes code in build and verifies the PR has been resolved.

System Test
- Third-party testing.
- Installation on dedicated test environment.
- Test cases written by test group based on requirements.
- Test cases executed by test group.
- Multiple iterations.
- ClearQuest Usage
    - ClearQuest database used to track bugs – Level B.
    - Bugs and enhancements entered by testers.
    - Problem Review Board decides what to include in each build.
    - Lead assigns PR's to developer.
    - Developer completes code and marks PR as CODE_COMPLETED.
    - Lead includes code in build.
    - Testers verify PR has been resolved.

Acceptance Test
- Tests that system will run properly in field environment.
- System test cases are run.
- Minimal developer involvement.

- Multiple iterations if necessary.
- Bugs tracked in ClearQuest (Level A).

4.7 INSTALLATION/SYSTEM UPGRADES

Due to the geographic distribution of the system and the phased implementation and deployment approach, an upgrade strategy was considered in the overall CHART II system design and architecture to prevent significant disruption of the operational system that could occur during the rollout of new releases. The design and architecture of the CHART II system minimizes the impact to operations and provides flexibility in the scheduling of the installation of new hardware and software. This is accomplished in several ways.

- Mixed Revision Levels – The design and architecture of the system allows installations to run different revision levels of the software. This allows a new release of a component to be installed in a phased manner rather than requiring a one-shot upgrade of all systems. It also allows for a selective upgrade strategy. For instance, if new protocols were added to support new field devices, only those FMS systems providing support for the devices would need the new software. Other FMS systems could continue to run the old version of the software until it is convenient or necessary to upgrade them as well.

- Remote Installation – Through the use of COTS applications such as UNICenter TNG Remote Control, the installation of software can be performed remotely. In addition, the SAN (Storage Area Network) provides the capability to upgrade SAN-attached server systems from a central site by temporarily reallocating SAN disks, upgrading the software on the disk, and then returning the disk to its default server.

- Modular Design – The modularity of the design allows CI's (configuration items) to be upgraded independently. FMS, CHART, AVCM, and the CHART Archive can each be upgraded independently of the others. In addition, the client side GUI's can be upgraded independently of the server side applications. Dependencies do exist between CI's, and in some cases, a change in one CI may necessitate a corresponding change in another, but these dependencies are kept to a minimum and are isolated to the interfaces between CI's.

Guidelines for the creation of the Installation package for CHART II are shown below. Note that the InstallShield software has been used to develop the Installation program for CHART II.

<u>General</u>
- Procedure should not require any special knowledge beyond being a system administrator.
- Create separate installation packages for servers, GUI, FMS.
- Use ServerSettings.exe to prompt installer for required system parameters.
    - ServerSettings.exe is launched from InstallShield.
    - Combine template Props File with installer responses to create server-specific or GUI-specific Props File.
- Keep system parameters to a minimum – most settings should be configurable on-the-fly.
- Be able to execute installation remotely – especially the GUI.
- Assume installation must be done on multiple servers.

<u>Server</u>
- Each service must be independently installable.

- Keep dependencies between services to a minimum – avoids illegal service combinations.
- Ensure services may live on any computer in the network.

GUI
- Keep procedure simple – there are many workstations to install.
- May not be necessary once browser-based GUI is installed (future release/build).
- All GUI modules are separately installable.

FMS
- Keep the FMS installation package small – FMS' have slow links.

The system installation process is documented in the CHART II Operations and Maintenance Guide.

4.8 SOFTWARE DEVELOPMENT WORK PRODUCTS

Table 4, below, provides information regarding CHART II software development work products for each phase of the software development lifecycle. For each product, a brief description is provided along with a reference to the applicable project document standard. The tools used to create the product are also indicated. The standards are located on the CHART2-SVR1 server under the PROJECT STANDARDS & PROCEDURES\CURRENT APPROVED directory.

Table 4 – CHART II Software Development Work Products

Phase	Products	Description	Standards	Tools
Business Area Architecture	Requirements Prototypes	• Software Prototypes to identify and verify requirements	• M361-PR-003 – Change Control for Requirements	• Inprise JBuilder • MS Visual C++
Business Systems Design	High-Level Design (Architecture Document) – Draft and Final	• Use Cases • Class Diagrams • Sequence Diagrams • Packaging/Deployment Diagram • Accompanying Text	• M361-ST-018 – Design Document Standard	• Telelogic Tau UML Suite • MS Word
	Detailed Design – Draft and Final	• Interface Definition Language • Class Diagrams • Sequence Diagrams • Accompanying Text	• M361-ST-018 – Design Document Standard • M361-ST-004 – IDL Coding Standard	• Telelogic Tau UML Suite • MS Word
	Feasibility Prototypes	• Software Prototypes to determine the feasibility of a development approach	• M361-PR-009 – Configuration and Use of ClearCase for CHART II Prototype Software	• Inprise JBuilder • MS Visual C++
Application Development	Software Code	• Software Code according to defined coding and process standards	• M361-ST-003 – Java Software Coding/ Implementation • M361-ST-004R0 – IDL Coding Standard • M361-ST-005R0 – C++ and Visual C++ Coding/ Implementation Standard	• Sun JDK • Inprise JBuilder • Iona ORBacus • MS Visual C++ • Microsoft Foundation Class (MFC) • Rational ClearCase
	Unit Test Plans	• Plans to prove software capability in accordance with requirements and design	• M361-PR-011 – Unit Testing	• MS Excel
	Unit Tested Results	• Documented unit test results and approval	• M361-PR-004 – Inspections and Peer Reviews	• MS Excel
Testing – Integration	Integration Test Plan	• Test Plan with Test Cases	• M361-ST-014 – Integration Test Plan Template	• MS Word

Phase	Products	Description	Standards	Tools
	Integration Testing Results	• Report of Integration Test results, integrated and tested software, problem reports	• M361-PR-016 – Software Control Notice	• MS Word • Rational ClearQuest
Testing – System	System Test Plan	• Test Plan with Test Cases	• M361-ST-008 – CHART II Test Plan Template • M361-ST-009 – CHART II Test Procedure Template	• MS Word
	System Test Results	• Report of System Test results, integrated and tested software, problem reports	• M361-PR-016 – Software Control Notice	• MS Word • Rational ClearQuest
Testing – Acceptance	Acceptance Test Plan	• Test Plan with Test Cases	• M361-ST-008 – CHART II Test Plan Template • M361-ST-009 – CHART II Test Procedure Template	• MS Word
	Acceptance Test Results	• Report of Acceptance Test results, integrated and tested software, problem reports, client acceptance	• M361-PR-016 – Software Control Notice • M361-PR-005 – Configuration and Use of ClearCase for CHART II Software Development	• MS Word • Rational ClearQuest • Rational ClearCase
Transition	Transition Plan	• Plan for transitioning operation to new system	• M361-ST-015 – Transition Plan Standard	• MS Word
Operation	Support	• Software support for analysis and correction of defects	• CM Process	• Rational ClearQuest

5. System Environments

5.1 SOFTWARE DEVELOPMENT ENVIRONMENT

The CHART II software development environment, shown in Table 5, below, includes both design and development tools. Both the software and hardware used in the development environment are based upon a platform similar to that used for deployment. In most cases, software is shown without specific versions due to the dynamic nature of COTS packages. The latest version information is maintained by the Configuration Management Team.

Table 5 – CHART II Software Development Environment

Component	Hardware	Software
CHART II GUI Workstation	Dual Pentium, Dual Monitor, 256MB RAM	• Windows NT 4.0 with Service Packs • Telelogic DOORS • Telelogic Tau UML Suite • Rational ClearCase • Rational ClearQuest • Microsoft Developer Studio • Sun JDK • Inprise JBuilder • Iona ORBacus
Development Server	Dual Pentium CHART II Server Configuration	• Windows NT 4.0 with Service Packs • CHART II Oracle Database • Oracle 8i • Lernout and Hauspie RealSpeak
Development Workstation	Pentium	• Windows NT 4.0 with Service Packs • Telelogic DOORS • Telelogic Tau UML Suite • Rational ClearCase • Rational ClearQuest • Microsoft Developer Studio • Sun JDK • Inprise JBuilder • Iona ORBacus
Development Workstation for Text-to-Speech Conversion	Pentium	• Windows NT 4.0 with Service Packs • Telelogic DOORS • Telelogic Tau UML Suite • Rational ClearCase • Rational ClearQuest • Microsoft Developer Studio • Sun JDK • Inprise JBuilder • Iona ORBacus • Lernout and Hauspie RealSpeak

5.2 INTEGRATION TEST ENVIRONMENT

Figure 6, below, shows the CHART II software development and integration testing configuration used at PB Farradyne (PBFI) and Computer Sciences Corporation (CSC).

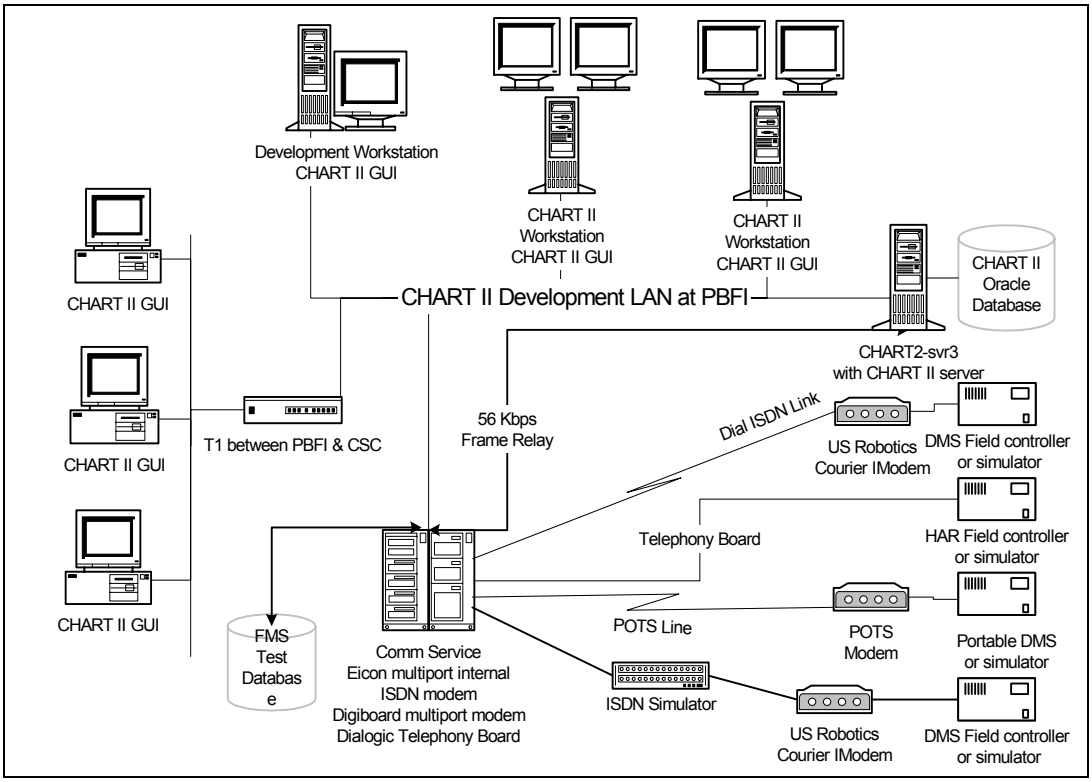


Figure 6 – CHART II Software Development and Integration Test Configuration

5.3 SYSTEM TEST ENVIRONMENT

Figure 7, below, shows the CHART II system test environment at the Maryland State Lab located at Computer Sciences Corporation. This environment was used for System testing.

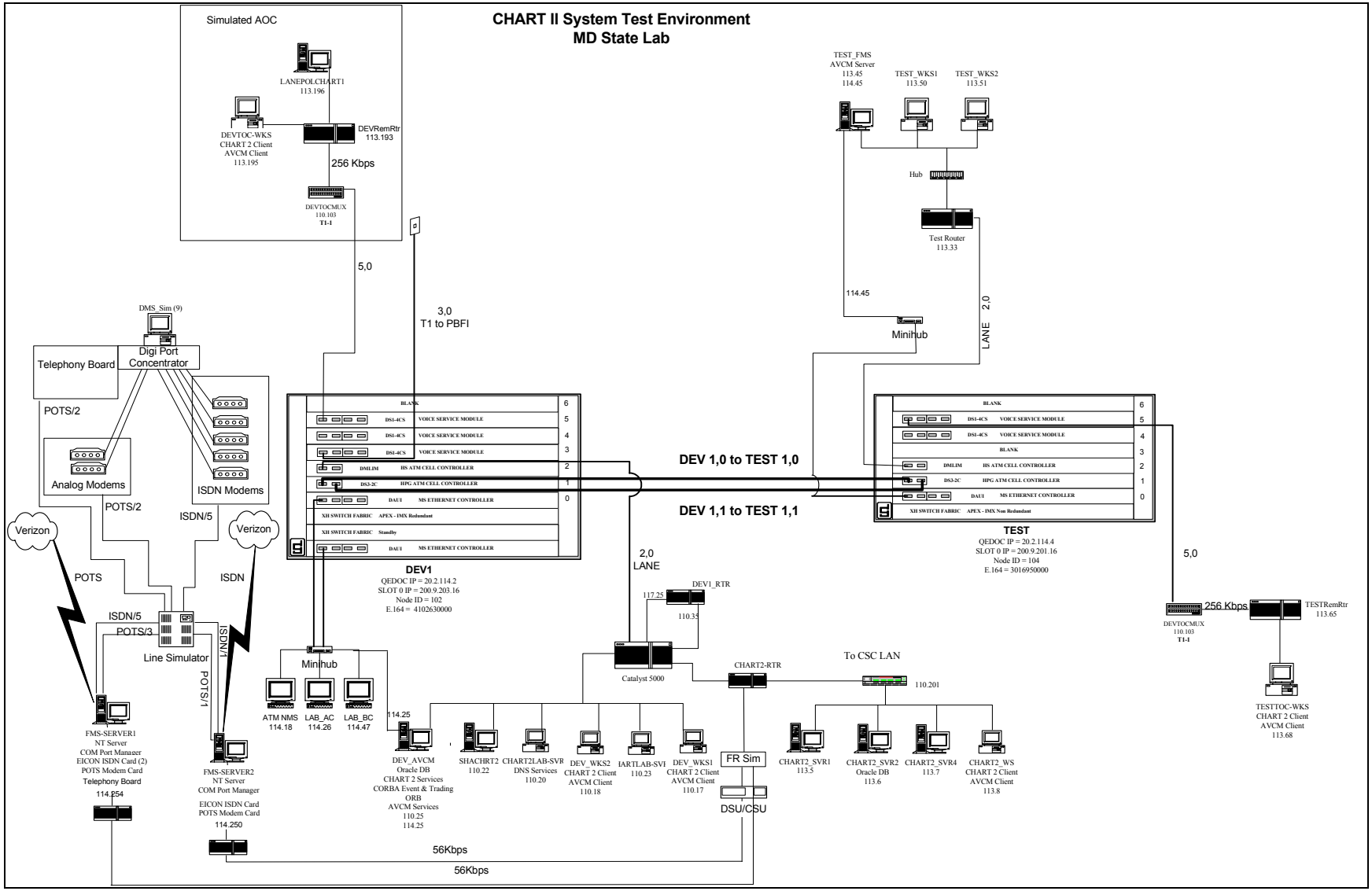


Figure 7 – CHART II System Test Environment

5.4 OPERATIONAL ENVIRONMENT

Figure 8, below, illustrates the operational environment of the CHART II system.

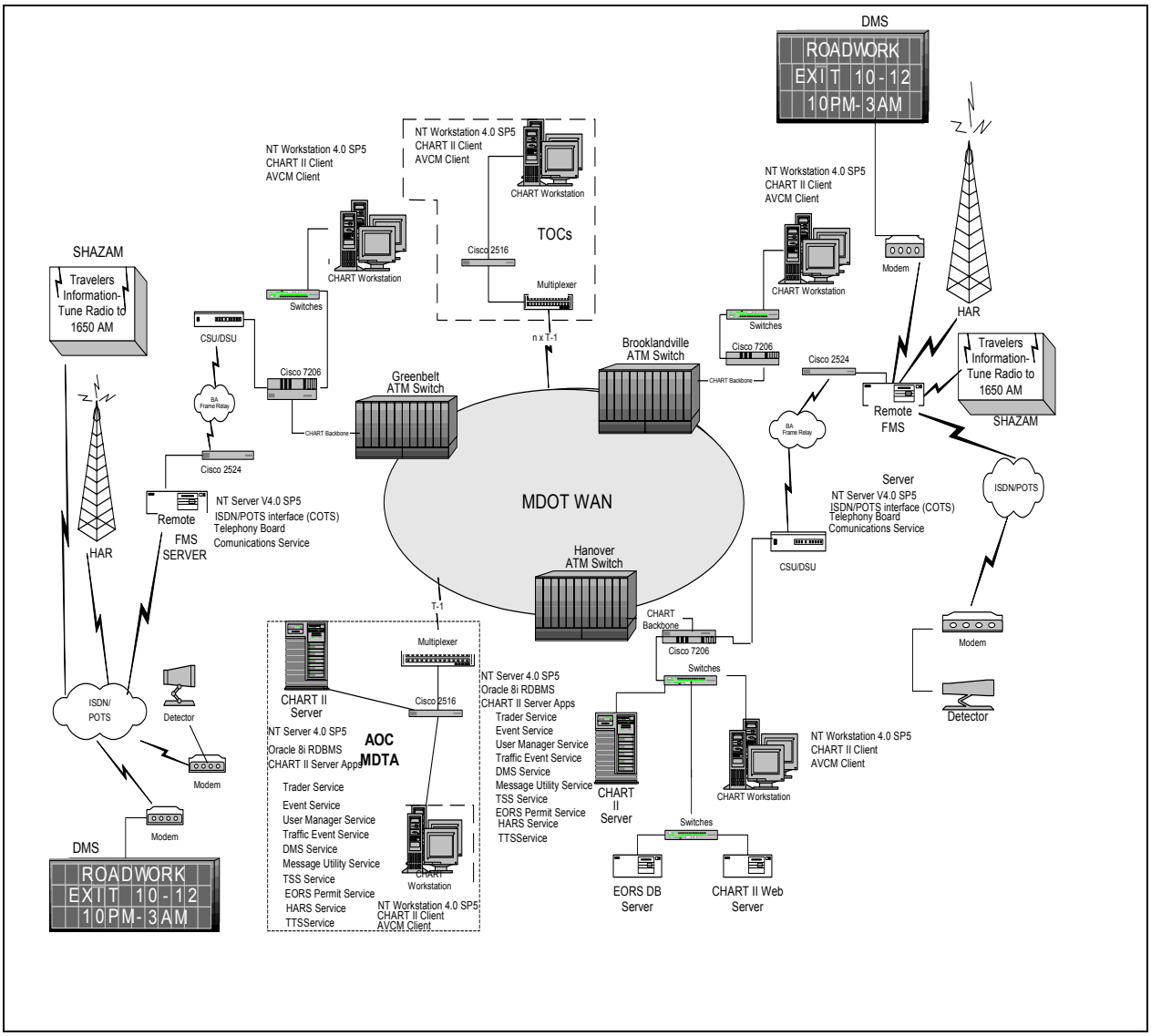


Figure 8 – CHART II Operational Environment

6. Additional CHART II Software Development Guidance

6.1 GRAPHICAL USER INTERFACE

The CHART II Graphical User Interface (GUI) application framework is created and designed with the dual purpose of providing a unified User Interface metaphor for the end-user and providing a collection of reusable software components that facilitates the addition of new features into the application by a programmer.

An essential component of a GUI application framework is a well-designed data model. The data model is the core element that stores all data objects and allows other software components (windows) interested in the data that those objects provide to attach as observers. The model also includes a mechanism that allows an object to indicate that it has been modified and provide hints as to how it has been modified. In this way, the data model provides a mechanism by which objects may easily update all windows in the system when they have received a state change event from a service application.

The data model in the CHART II GUI application provides observers with the ability to attach varying priority levels which determine the acceptable delay between the point in time when a data object detects a state change and that state change is conveyed to the end-user. This capability allows windows that must be updated very rapidly to attach at the highest priority level and windows that can tolerate some delay to attach at lower priority levels. The data model aggregates updates during the delay period in order to avoid excessive repainting of GUI screens. Thus, if a particular object is modified three times within a one-second period, a highest priority window may render all three state changes while a lower priority window may render only the final state of the object at the end of the period.

A central philosophy of the CHART II GUI application is the use of the object action paradigm. Object action involves allowing the user to select the object or objects upon which he/she would like to act and then allowing them to select from a list of actions which are pertinent to the selected object(s). Another policy upheld throughout the GUI application is the concept of object consistency. This policy states that if the user selects to operate upon an object, regardless of which window that object has been selected from, the object will show a consistent state and will allow the operator to perform consistent operations. These policies imply that the objects within the CHART II GUI application must be capable of reporting their state at any given point in time, and must be capable of creating a menu of currently available operations. The menu infrastructure addresses the latter of these items. The framework provides the Menuable interface that any object that would like to display a popup menu must implement. This interface provides the object with information, such as the current user's access token, which it will need in determining which of its possible operations should be added to a menu at this time.

During the course of his/her work, an end-user of the CHART II GUI may need to execute commands that will take a significant amount of time for the system to execute. The user is provided with two windows for viewing the status of outstanding requests and commands: the command status view and the command failures view. The command status view provides the user with a list of operations that they have attempted that have not yet completed. When a

command completes, it will be removed from the command status view. If it completed with a failure, it will be added to the command failures view. This allows an operator to view a list of system requests that have failed. Each entry in this window will contain a description of the request, the date and time the request was issued, and text describing the reason the command failed.

The CHART II GUI implements an installable module framework. Modules manage collections of closely related objects and functions that can be easily added to the GUI application. This framework also provides for the customization of CHART Workstation installations by providing the capability to omit modules if they are not required.

A concerted effort was made to avoid pop-up dialogs in the CHART II GUI. Pop-ups distract busy operators because they force an operator to stop what they are doing to clear the pop-up. Most dialogs have a status pane at the bottom that is used to display status for synchronous commands that are executed from the dialog. Status/failure messages for asynchronous commands are displayed in the Command Status and Command Failures windows, which can be minimized or closed until needed. Popup windows may be used when a user invokes a synchronous command that is only available from a menu item and the command fails. In this case, a popup dialog can provide helpful feedback, since the user has to wait for the command to complete and the feedback can be displayed immediately. However, status messages for synchronous commands invoked from menus are also displayed in the Command Status and Command Failures windows for consistency with asynchronous commands. The Unhandled Resources dialog is the only dialog that pops up without any action being invoked by the user. However, this window can be minimized to keep it from popping up again.

A strict demarcation of responsibility is made between the servers and the GUI's. The servers only concern themselves with content and the GUI's only concern themselves with presentation. Although this seems an intuitive concept, it is sometimes difficult to implement. For instance, should each operator be allowed to format a report differently or should the servers enforce a system-wide standard? In all cases for CHART II, the GUI has complete control over presentation. Adherence to this rule allows for the planned creation of a browser-based GUI with no changes to the services.

Although the GUI is dependent on the servers for all operations, it is not dependent on any particular server. This is evident in the fact that the GUI has no idea how many server instances currently exist or should exist. It only knows the system objects that exist (e.g., DMS', HAR's, Traffic Events, etc.) based on what CORBA offers by services and what it can do with them. This also means that servers may go away and come back at any time, and the GUI's will be unaffected. Obviously, any objects served by missing servers are unavailable until the server returns; however, the GUI is not adversely affected by existing servers disappearing or new servers becoming active.

The planned browser-based GUI uses the exact same interface to the servers as the existing CHART II GUI. The implication of this is that the servers have no idea which user interface an operator is using. The "push"-based update mechanism of the CHART II data model, which is useful for driving window updates in the CHART II GUI, is not needed in a browser-based GUI as browsers can only "pull" the data from the web server. In this case, it is only necessary to update a cache containing the state of the system objects. However, there will only be one cache to update for each web server in the system instead of one cache (data model) per user that is used in the CHART II GUI. Therefore, a single connection to the CORBA

event/notification service is required to keep the cache updated for many users, as opposed to one connection per user as required by the CHART II GUI.

6.2 SERVICES

The CHART II services are a collection of executables that manage system objects. System objects are things like devices, traffic events, traffic plans, or operators. Each server is responsible for managing all aspects of one type of object from object creation to deletion. All servers have a few common responsibilities regardless of what type of object they manage:

- Creation – create new objects
- Deletion – delete existing objects
- Persistence – object storage and retrieval between service restarts
- Make Offers – advertise objects in the CORBA Trading Service for discovery by the GUI and possibly other servers
- Retract Offers – remove CORBA offers once an object is deleted
- State – push object updates through event channels to keep all listeners up to date
- Object Support – provide object-specific actions such as retrieving current configuration
- Security – ensure only permitted users are allowed to perform object actions

Services are independent of each other in that they do not require any other service to be active in order to run. This means a service may be restarted without having to restart any other service. This is demonstrated by the ability to upgrade a service by simply stopping it, replacing a JAR file, and restarting it. No other service is affected.

Once running, one service may make a request to another service, but it will continue on if that service does not exist. Obviously, any operations on one server that require operations from another server will only work when both servers are running.

There is no requirement for all services to run on the same computer in the CHART II system. Although historically one computer hosts all of the services for a given operations center, there is complete flexibility in where services live. Each service may live on its own computer, or many operations centers' worth of services may live on one computer. All that is required is that all services can see each other and the CORBA services (Trader, Event, and Notification) on the network.

6.3 DATABASE

The CHART II database design consists of six major areas:

- User/System Management – These entities consist of the complete suite of information to tie together the users, roles and functional rights with the center's identification and logical grouping.

- Resource Configuration – These entities define the configuration of the system resources, including devices (e.g., DMS', HAR's, SHAZAM's, detectors, and cameras), and the equipment inventory.

- Automated Planning – These entities represent planning objects such as message libraries, schedules, and response plans.

- Logging – Various system logs are maintained for communications, operations, and events such as incidents, disabled vehicles, and weather advisories. See Section 6.6 for more information pertaining to Logging.

- System Operation – These entities consist of information required for system operation that are either part of the system parameters needed to run an automated system or data received from an outside source but stored within the CHART II system.

- External Systems – This refers to the external systems identified as interfacing with the CHART II database that are not part of the CHART II system but provide information that may be stored as logging or tracking data.

Most of the CHART II data is replicated from the main SOC in Hanover to the AOC (Authority Operations Center) at MdTA's location for recovery purposes. The replicated data includes the data classified as static data and logging. Static data is set up by an administrator for operation of the system and is infrequently modified. Logging is part of system operation and tracks traffic events. Logging data is typically maintained in the operational system for a period of two weeks. Replication of static data is performed in order to provide a consistent device configuration. Replication of traffic event information is performed to recover in the event of a server failure. The entities that are not replicated are basically those that could be refreshed at restart from the source database.

The database contains all current objects in the system. This includes all configuration data necessary to reconstitute an object if that object's server is suddenly restarted. The one piece of an object that should not usually be stored in the database is its Interoperable Object Reference or IOR. An IOR is the globally unique name that CORBA gives an object. The problem with storing an IOR in the database is that the IOR contains the name of the computer where the object's service lives. If the given service is ever moved to a different computer (e.g., in a failover scenario), the IOR will no longer work, and the object will cease to exist. If a situation absolutely requires an IOR be stored in the database, logic should be included in the process using the IOR to time out unresolved IOR references and look for new objects in the Trader based on the object's CHART ID. Each object in the system has a CHART ID that is unique within the system.

Note that CHART II software development is performed on a local copy of the database which is updated periodically from the CHART II production database.

6.4 FIELD MANAGEMENT STATION

The Field Management Station (FMS) provides communications support functions for traveler information devices, traffic detection devices, and other telecommunications support required by the CHART II system. The FMS software, like the CHART II software, uses a distributed architecture communicating via CORBA to provide a highly available system. Each FMS server is a standalone system capable of communicating with any field device for which it has a matching communications port type. All user interaction with the FMS is handled through the CHART GUI.

The FMS servers support the following types of communications ports:

- ISDN (Integrated Services Digital Network) – used for communications with fixed DMS' and RTMS (Remote Traffic Microwave Sensor) detectors.
- POTS (Plain Old Telephone Service) – used for communications with portable DMS'.
- Telephony – used for communications with HAR and SHAZAM.

A telecommunications study revealed that ISDN was the most cost-effective medium for communications with field devices because telephone tariffs allow unlimited calling within a telephone company central office (CO) for only a fixed monthly charge. As a result, FMS servers are located throughout the state at locations that meet the criteria for these tariffs. Some are located in SHA or other agency buildings. Others are located in roadside huts.

This distribution of communications capabilities also provides redundancy. Field devices may be configured with both primary and backup FMS communication server designations. If the primary server fails with certain errors a specified (configurable) number of times, the communications to the device is attempted through another server.

The basic functioning of FMS servers in operations is as follows. When communications with a field device is required either through an operator action or regular poll, the device object requests a port of the needed type (e.g., an ISDN port) and passes the telephone number to the port manager. Communications between the device object residing on the CHART II application server and the port manager on the FMS server take place using CORBA over the MDSHA wide area network.

The port manager checks to see if a port of the requested type is available and attempts to make the connection. If successful, the device object then carries out its communications with the field device. The FMS server simply passes all communications in each direction. Additional details of this operation may be found in the design documentation.

Similar operations occur for the other port types, except that the telephony board must use DTMF (Dual Tone Multi-Frequency, i.e., touch tone) tones to control the HAR and SHAZAM. It must also send and receive audio. Text-to-speech servers used for HAR messages are co-located in two of the FMS servers. One is at the SOC and the other at TOC3 to coincide with telephone company LATA (Local Access and Transport Area) boundaries. This arrangement allows for reduced costs for calls to HAR's by keeping the calls within the two LATA's where the HAR's are currently located.

6.5 SIMULATORS

During the initial development stages of CHART II, hardware simulators were available for the FP9500 DMS, HAR and SHAZAM. Software simulators were developed for the other sign models and were used for later development and testing after the hardware simulators were returned to MDSHA and to provide for testing with multiple devices simultaneously. The software simulators are also used in the CHART II production system for training.

The software device simulators were developed using a framework developed in Java that consists of a number of utility classes that can be reused for different protocols and devices. Software simulators exist for:

- Addco DMS
- FP1001 DMS
- FP2001 DMS
- FP9500 DMS
- PCMS (Display Solutions) DMS
- Sylvia DMS
- TS3001 DMS
- ISS AP55 HAR
- Viking SHAZAM

The software device simulators are typically installed on a separate computer or computers that have the desired communications capability (ISDN or POTS). This permits testing using a configuration that is as near to the production configuration as is possible. The simulators are usually installed as system services. Instructions for installing and configuring the simulators and the simulator installation packages are available on CHART2-SVR1\MODELS\DEV TEAM INTERNALS\SIMULATOR INSTALLS.

6.6 SYSTEM AND DEVICE LOGGING

All services produce a log file containing significant activities. These logs are named with the format of <service>_<date>.txt (e.g., DMSService_021225.txt). A parameter in each service's Properties File specifies the path where the log lives.

Each service's Properties File also specifies what logging level is desired. Two levels are defined for all services: Production and Debug. Production logging is used for error conditions and high-level informational messages such as starting, stopping, attempting to discover new objects, and object creation and deletion. Debug level logging includes the Production level and is generally used to produce lower level messages for the same issues reported with the Production level.

Beyond Production and Debug, individual services are free to define their own logging flags. For instance, the DMS Service has over 40 flags defined.

For services that talk to devices, an additional log can be created on a per-device basis. This log is enabled by the operator on the device's properties dialog.

6.7 WEB INTERFACES

MDSHA maintains a public web site (www.chart.state.md.us) to inform users about traffic and weather emergencies, traffic conditions, and both general and technical information about CHART. The traffic information provided in real-time on the web site includes:

- Traffic flow information (current speed from RTMS vehicle detectors)
- Incident information
- DMS messages

- Indication of DMS and RTMS detector status (on/off-line, message active)

The information is provided to the web site via a CORBA interface that implements the IDL. The web site software registers with the CORBA Event Service to receive events that are pushed for the DMS, Event, and TSS (traffic sensor system) objects. Thus, any changes to the IDL that affect these objects must consider the impact on the public web site.

7. Adding New Features to CHART II

7.1 CHART II APPLICATION FRAMEWORK CLASSES

CHART II Service and GUI Application frameworks provide a set of reusable classes which provide the same basic structure and functionality for all CHART II modules. Following is a brief description of all framework classes that are widely used in various CHART II service and GUI modules:

- **ArbitrationQueue**: This abstract class defined in the CHART2.DeviceUtility package manages the display of online messages on a device. This class must be extended by device implementations to tailor message arbitration rules to meet their specific needs.

- **CHART2Service**: The CHART2Service defined in the CHART.CHART2Service package is an application that helps in installation and termination of the modules in the CHART II system.

- **CommandQueue**: This class defined in the CHART2.Utility package provides functionality to add a command to the queue and execute commands added to the queue in a first-in-first-out fashion.

- **CorbaUtilities**: This class defined in the CHART2.Utility package contains a collection of static CORBA utility methods that can be used to register service types, publish objects in the CORBA Trader service, and get the server port number from the ORB.

- **DataModel**: This class defined in the CHART2.DataModel package is used as the subject side of the subject/observer pattern. This class contains methods that allow for storage, efficient lookup, and updating of objects. It also provides for the attachment of observers at various priority levels.

- **DBConnectionManager**: This class defined in the CHART2.Utility package 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.

- **DefaultJDialog**: This class defined in the CHART2.GUI package provides the default implementation of the WindowManageable interface. It handles all interaction with the WindowManager for attaching and detaching, as well as saving, the window position.

- **DefaultJFrame**: This class defined in the CHART2.GUI package provides a default implementation of the WindowManageable interface, and may be used as a base class for other frame windows in the GUI.

- **DefaultServiceApplication**: This class defined in the CHART.Utility package is the default implementation of the ServiceApplication interface. This class is passed a property file during construction. This property file contains configuration data used by this class to set the ORB concurrency model, determine which ORB services need to be available, provide database connectivity, etc. The property file also contains the class names of service

modules that should be served by the service application. During startup, the DefaultServiceApplication class instantiates the service application module classes listed in the property file and initializes each.

The DefaultServiceApplication class provides methods to publish objects in the trader and maintains a file of offers that have been published in the Trading Service. It uses its offer ID file to clean-up old offers prior to initializing modules during its next start. This keeps multiple offers for the same object from being placed in the trader. This class also provides methods to manage event and notification service channel discovery, creation, and registration.

- **Droppable**: This interface defined in the CHART2.DataTransfer interface must be implemented by any object wishing to take part in a drag and drop operation.

- **FunctionalRightType**: This class defined in the CHART2.Utility package lists the types of functional rights possible in the CHART2 system.

- **GUI**: This class defined in the CHART2.GUI package is the core of the CHART2 GUI application. It is a singleton which starts up the application and handles all of the framework functionality such as startup, shutdown, login, logout, and management of the modules, among other things. It is also a container for the important unique objects in the system such as the DataModel, etc.

- **GUIMenuItems**: This interface defined in CHART2.GUI package is used to define the text of all menu items which appear in the context menu. All menu items used by all modules and objects should be added to this interface.

- **GUIModelObserver**: This interface defined in the CHART2.DataModel package must be implemented by GUI components that would like to observe changes to the data model.

- **IdentifierGenerator**: This class defined in the CHART2.Utility package is used for generating unique identifiers to be used to identify all identifiable CHART2 objects.

- **InfoReportable**: This interface defined in CHART2.GUI package must be implemented by any object which would like to allow the user to view it's detailed status in an InfoReport window.

- **InstallableModule**: This interface defined in the CHART2.GUI package should be implemented by any GUI module wishing to become created and installed by the GUI and to participate in the GUI framework.

- **Log**: This class defined in the CHART2.Utility package is used by any CHART II system objects to log system trace messages and stack trace to a text file.

- **Menuable**: This interface defined in the CHART2.GUI package must be implemented to support a context menu for the object via the GUI's makeMenu() method.

- **NavClassFilter**: This class defined in the CHART2.GUIUtility package is used to handle the display of a list of navigator objects of a given class or subclass.

- **Navigator**: This class defined in the CHART2.Navigator package represents a window which contains a tree on the left side and a list on the right side. Tree elements represent groups of objects and the list on the right side represents the objects in the selected group.

- **NavListDisplayable**: This interface defined in the CHART2.Navigator package must be implemented by any object wishing to be displayed in the list (right) side of a Navigator window.

- **NavTreeDisplayable**: This interface defined in the CHART2.Navigator package must be implemented by any object wishing to be displayed in the tree (left) side of the navigator window.

- **OperationsLog**: This class defined in the CHART2.Utility package is used by CHART II service application objects to log the operations performed by the users of the system.

- **PushEventConsumer**: This class defined in the CHART2.Utility package represents a client side connection for a CORBA push event channel.

- **PushEventSupplier**: This class defined in the CHART2.Utility package represents a server side connection for a CORBA push event channel.

- **PushNotifyConsumer**: This class defined in the CHART2.Utility package represents a client side connection for a CORBA push notification channel.

- **PushNotifySupplier**: This class defined in the CHART2.Utility package, represents a server side connection for a CORBA push notification channel.

- **QueueableCommand**: This interface defined in the CHART2.Utility package must be implemented by any device command in order that it may be queued on a CommandQueue.

- **Service**: This interface defined in the CHART.Utility package must be 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.

- **ServiceApplication**: This interface defined in the CHART.Utility package must be implemented by objects that can provide the basic services needed by a CHART II service application. These services include providing access to basic CORBA objects that are needed by service applications, such as the ORB, POA (Portable Object Adapter), Trader, and Event Service.

- **ServiceApplicationModule**: This interface defined in the CHART.Utility package must be implemented by all 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 ServiceApplication to perform actions such as object creation and publication.

- **ServiceApplicationProperties**: This class defined in the CHART2.Utility package is used to provide access to configuration parameters stored in the service application's property file.

- **TokenManipulator**: This class defined in the CHART2.Utility package contains all functionality for manipulating the CHART II access tokens which are required to invoke all of the restricted access functionality in the system.

- **WindowManageable**: This interface defined in the CHART2.GUI package should be implemented by anyone controlling a frame window in the CHART II system.

- **WindowManager**: This class defined in the CHART2.GUI package manages WindowManageables that are added to it and provides functionality for saving the window positions and closing all of the windows.

7.2 ADDING A NEW DEVICE

To add a new device to the CHART II application, the following steps should be followed.

- Define the IDL interfaces to support new device functionality.
- Create any new tables in the database and/or update existing tables.
- Add a new server module to implement the interfaces defined in the IDL using the CHART II service application framework.
- Add an installable GUI module to add device functionality.

The following sections explain each of these steps in detail.

7.2.1 IDL Definition

The first step in the process of adding a new device is to define the IDL for the device with the system interfaces discovered during the High Level Design. IDL coding standards should be followed while defining the IDL. The following rules should be followed when defining the IDL.

- Interfaces and data structures used should be well defined to describe all operations that can be performed on the device.
- Define the service types for all objects that are planned to be published in the trader.
- If required, define the names of event channels that will be published in the trader.
- Define all the exceptions that each operation defined in the interfaces can throw.
- Define all CORBA events that will be pushed by the server module.
- IDL interfaces should return as much information as possible related to the operation in order to prevent the client from having to make follow up calls to get the supporting data.
- Define factory interfaces to gain access to and manage the CORBA objects that implement the interfaces defined in the IDL. For example, use DMSFactory to manage DMS'.
- Interfaces defined to describe the device functionality should extend the UniquelyIdentifiable, GeoLocatable, and CommEnabled interfaces. Refer to the design document for a description of these interfaces.
- If the new device being added requires the arbitration queue, the IDL interface defined should extend the ArbitrationQueue interface.
- If the control of the new device being added can be transferred from one operations center to another, the IDL interface defined should extend the TransferableSharedResouces interface.

- Avoid using ValueTypes because they cannot be used in events being pushed on CORBA Notification Service channels.

7.2.2 CHART II Service Application Module

The CHART II service application uses an installable module framework. A module manages initialization and shutdown of CORBA objects that implement the interfaces defined in the IDL. This framework also provides for the customization of CHART II service installations by providing the capability to pick and choose the modules that need to run on the same machine if required.

The new CHART II service application module should implement the ServiceApplicationModule interface and handle the initialization and shutdown of all the CORBA objects that it serves. Steps involved in initialization of a module are as follows.

1. Register all service types of objects that will be published in the trader by this module. All service types registered should also register the super types for each of the extending interfaces defined in the IDL. For example, if you are registering a service type for an interface named CCTVCamera which extends the UniquelyIdentifiable, GeoLocatable and CommEnabled interfaces, in addition to registering the service type for CCTVCamera, you should also register the service types for the UniquelyIdentifiable, GeoLocatable and CommEnabled interfaces as super types of CCTVCamera.
2. Read any properties defined in the property file that are specific to the module that is being added.
3. Create and register any event channels that are needed to push device events defined in the IDL. A PushEventSupplier should be created for each event channel and should be published in the trader using the ServiceApplication object.
4. Create a CommFailureDB class object to log all communication failure errors in the database.
5. If required, create the factory implementation object that creates all device implementation objects with the data read from the database. Activate the factory with POA using the ID of the object and publish the factory object in the trader. The ID of the factory object should be retrieved by calling the getPersistentObjectID() method of the CorbaUtilities class in the Utility package by passing the name of the file in which the ID would be persisted. Typically, a factory ID would be stored in a file matching the name of the class with the ".id" extension, e.g., a DMSFactory ID would be stored in a file named "DMSFactory.id". If the ID has not been previously persisted, the getPersistentObjectID() method creates a new ID and persists it in a file with the given name. It is important to activate the factory object with a persisted ID during initialization in order to create the same IOR reference if the service is restarted.

A factory implementation class should implement the corresponding factory operations class generated from the IDL. Upon creation, a typical device factory class retrieves device data from the database, creates and activates all device objects implementing the device interfaces defined in the IDL, and provides implementation for all operations defined in the factory IDL interface. It should also handle any tasks such as pushing events periodically and checking for uncontrolled resources using Timer objects. At shutdown, the factory object should gracefully shutdown all device objects and any timer threads being used.

A class implementing the device interface should provide implementation for all operations declared in the IDL device interface. It maintains device configuration and status data. It should create a protocol handler class that implements the device specific protocol to communicate with the field device. Some of the things to remember while implementing operations defined in the interface are as follows.

- Check each input parameter for null because CORBA allows clients to pass null as an input parameter to the server.
- Throw only those exceptions that are defined in the IDL for the operation.
- An operation is considered to be a success if the any changes to the object's state resulting from the operations are persisted in the database. Do not throw an exception for non-critical failures such as failure to push an event. If you are throwing an exception, make sure that you have rolled back any changes to the object state to the prior state.
- Use synchronization for operations that change the object state because CORBA allows objects to be called from multiple threads.
- Implementations of all IDL operations that require access verification should first verify if the caller has the necessary privileges granted to perform the operation using the token passed to them.
- Implementations of all IDL operations should inform the clients about the current status of the operation including any failures using the command status reference passed to them as an input parameter. The server object should first check that the command status reference is not null before using it because a client that is not interested in the command status of an operation can pass a null for the command status object. Make sure the completed() method of the command status object is called when the operation is complete with or without any failures.

7.2.3 New GUI Module

A typical CHART II GUI module extends PushConsumerPOA to receive CORBA events from various events channels to which it is connected and handles the event data by updating the related GUI objects in the data model. It implements the InstallableModule interface of the CHART II GUI application framework in order to be created and installed when the GUI is started. It also implements any additional supporter interfaces needed in order to allow other modules to access services provided by this module. For example, a GUICCTVModule should implement the NavFilterSupporter interface in order to support the functionality to create filters for CCTV objects that are shown in the Navigator. If the CCTV objects can be used in response plans, the GUICCTVModule module needs to implement the GUIResponsePlanItemCreator interface to let the GUITrafficEventModule support the functionality to create response plan items with CCTV objects. At startup, the GUICCTVModule module should register with the GUITrafficEventModule using the API provided as a response plan item creation supporter.

All GUI modules should implement the discoverObjects() method defined in the InstallableModule interface to discover the objects in the trader they are interested in and create GUI wrapper objects with the actual CORBA object to call for executing commands and add them to the data model. For example, GUICCTVModule would discover CCTVCamera objects and create a GUICCTVCamera objects that wraps the CORBA object. All GUI modules must also (1) implement the discoverEventChannels() method defined in the InstallableModule

interface to discover event channels in the trader they are interested in using the event channel names declared in the IDL, and (2) register the PushConsumer with the event channel using the API provided in the GUI class to receive the events pushed on those channels by the server.

To display the group of newly added devices in the Navigator, implement a class that extends the NavClassFilter and implement the Menuable interface. This class should define the columns to be displayed on the Navigator list table, handle context menu creation when the mouse is clicked on them, and handle options selected on the context menu. All GUI objects that are displayed in the Navigator should implement the NavListDisplayable interface defined in the Navigator package and/or extend the NavTreeFilter class (or one of its subclasses) depending upon which side of Navigator they are displayed. If the objects need to be displayed on the tree (left) side of the Navigator, they should extend NavTreeFilter or one of its subclasses. If the objects need to be displayed on the list (right) side of the Navigator, they should implement NavListDisplayable.

All context menu options to be displayed when the mouse is clicked over the Navigator object should be declared in the GUIMenuItems interface defined in CHART2.GUI package. Objects displayed in the Navigator list should implement the Menuable interface in order for the Navigator to call the object on which the mouse was clicked to construct the context menu. The class implementing the Menuable interface should construct the context menu with valid options that the user can perform based on functional rights granted to the user and the current state of the object.

GUI wrapper objects that wrap the CORBA objects should cache the current state of the CORBA object in the GUI for fast access. GUI wrapper objects should call the CORBA object to get the initial status when the GUI starts up. After this, the state should be maintained by processing the CORBA event/notification service events. However, a Refresh command should also be provided to allow the user to query the current state of the object in case the CORBA event/notification service has dropped events. Any part of the GUI that is interested in the object's state should call the wrapper object to get the current state.

Wrapper objects should provide API's to other GUI objects similar to the operations defined in the IDL for the CORBA object being wrapped so that interested GUI objects can use the CORBA object. The wrapper objects should handle CORBA exceptions, such as COMM_FAILURE and TRANSIENT, when calling the CORBA object in addition to the exceptions defined for the operation in the IDL. Wrapper objects should throw GUIException for any exception thrown by a CORBA call. The use of GUIException indicates to the catcher that the error has been logged and that the error message is user friendly and can be displayed to the user.

All new properties dialogs/screens required for the new device should extend the DefaultJFrame class defined in the GUI package, which provides functionality to save the window positions when the windows are closed.

7.3 ADDING A NEW MODEL OF AN EXISTING DEVICE

The CHART II system provides the framework to support multiple models of DMS and TSS devices. To add a new model of a DMS or TSS, the steps below should be followed.

- Add IDL modifications specific to the model being added.
- Modify the database to support the new model.
- Modify the server module to support creation and control of the device of the new model.
- Modify the GUI module to support the new model type.

7.3.1 IDL Modifications

CHART II DMS and TSS control modules were designed taking into consideration the need to support multiple models of DMS and TSS devices. To add a new model, follow the steps below.

- Modify the IDL to add the new device model to the list of models in the IDL.
- Add a new interface to the IDL to represent the new device model that extends the base device interface.
- Add a new status structure with model-specific data extending the base device structure.
- Add any new definitions, such as extended status information that can be displayed to the user in addition to the basic status already defined. For example, to add a NTCIP model of a DMS to the CHART II system, the following modifications need to be made to the IDL:
    - Add a model ID for the NTCIP model to DMSModelID enumeration defined in "CHART2DMSControl.idl".
    - Add a new NTCIPDMS interface that extends the Chart2DMS interface with any new operations that can be performed on the device.
    - Add a new NTCIPDMSStatus valuetype that extends the Chart2DMSStatus valuetype.
    - Add new definitions to describe any extended status information made available by the NTCIP protocol.

7.3.2 Server Module Modifications

To add a new model of an existing device to the CHART II system, the following steps should be used.

- Add a new model-specific protocol handler class that implements the base protocol handler interface which specifies a common set of API's that must be supported by all models of the device. For example, to add a protocol handler for a NTCIP DMS, the protocol handler class will implement the DMSProtocolHdlr interface defined in the CHART2.DMSProtocols package using the NTCIP protocol. Protocol handlers communicate with the field device using a DataPort object, which is a port (direct connect, POTS or ISDN modem port) that allows binary data to be sent and received.
- Implement the interface defined for the new model in the IDL. This involves extending the class that implements the base device interface and implementing the operations defined in the IDL interface for the new model. The Impl class should be registered with the POA using the tie class generated from the IDL, and the resulting CORBA object reference should be registered in the CORBA trading service. The Impl class should create an instance of the protocol handler class for communications with the field device.
- Implement any valuetype default factories and default implementations for new valuetypes defined in the IDL. Make sure the default valuetype factory and implementation class naming conventions are followed in order for the ORB to find the classes automatically when

it tries to marshall/unmarshall the data during transport. For example, if an NTCIPDMSStatus valuetype is defined in DMSControl IDL, default factory and default implementation classes should be named as NTCIPDMSStatusDefaultFactory and NTCIPDMSStatusImpl respectively, and should be in CHART2.DMSControl package.
- Modify the code to create the objects of new model when the server starts up and for the IDL operation to create the device. For example, for a DMS, the code to create the DMS objects at startup and on-the-fly using the IDL interface is in the createChart2DMSImpl() method of the DMSControlDB class. Modify this method to create a NTCIPDMSImpl if the given model ID is NTCIP.

7.3.3 GUI Module Modifications

To add a new model of an existing device, follow the steps below.

- Add a new model supporter class that implements the base model supporter interface that can create a GUI wrapper object for the new model. For example, for a NTCIP DMS, you should add a new class such as GUINTCIPDMSModelSupporter which implements the GUIDMSModelSupporter interface and register the class with the GUIDMSModelFactory object. This supporter will be called when a new GUINTCIPDMS object needs to be added to the system.
- Add a new class to display the properties dialog of the new model. The new class should extend the DefaultJFrame class and should implement the GUIModelObserver interface to get updates about any changes to the wrapper object while the dialog is open. For example, for a NTCIP DMS, add a class that extends DefaultJFrame, and implements the GUIModelObserver interface. The NTCIP DMS properties dialog class may also implement the DMSPixelStatusListener interface based on whether the NTCIP protocol provides pixel status or not.

7.4 ADDING A NEW TRAFFIC EVENT TYPE

To add a new event type to the CHART II system, the steps below should be followed.

- Modify the TrafficEventManagement IDL to add a new interface and new event type
- Modify the database to support the new event type.
- Modify the server module to support creation and control of the new event type.
- Modify the GUI module to support the new event type.

7.4.1 IDL Modifications

To add a new event type, make the following modifications to the TrafficEventManagement IDL.

- Declare a service type for the new event type.
- Add any new events that need to be pushed for the new event type to the TrafficEventEventType enumeration.
- Add any new fields that need to be added for the new event type to the TrafficEventDataChanged enumeration.
- Define any data structures needed for the new event type.

- Modify the TrafficEventEvents union to push appropriate data for the new event type events.
- If needed, define a new valuetype that extends the BasicEventData valuetype for the new event data.
- Add an event type ID to the TrafficEventTypeValues interface for the new event.
- Add a new interface to the IDL to represent the new event that extends the base TrafficEvent or RoadwayEvent interface.

7.4.2 Server Module Modifications

To add a new event type to the CHART II system, the steps below should be followed:

- Implement the interface defined for the new event in the IDL. This involves extending the class that implements the base event interface and implementing the operations defined in the IDL interface for the new event. The Impl class should be registered with the POA using the tie class generated from the IDL, and the resulting CORBA object reference should be registered in the CORBA trading service. The Impl class should provide implementation for the methods defined in the new event IDL interface.
- Implement any valuetype default factories and default implementations for new valuetypes defined in the IDL. Make sure the default valuetype factory and implementation naming conventions are followed in order for the CORBA framework to find the classes automatically when trying to marshall/unmarshall the data during transport.
- Modify the code in the createTrafficEventImpl() method of the TrafficEventGroup class to create the objects of new event type when the server starts up and for the IDL operation to create the event.

7.4.3 GUI Module Modifications

To add a new event type to the CHART II GUI, follow the steps below.

- Add the new event type to GUIMenuItems to add the context menu option for creating an event of the type new event.
- Add a button to the CommLogDialog class to create an event of type new event.
- Update EventNavGroup to handle an action to create an event of type new event.
- Add a GUI wrapper object to wrap the functionality of the CORBA object of the new event.
- Update the GUITrafficEvent class to add type name and value for the new event.
- Modify GUITrafficEventHolder to handle creation of the event of type new event.
- Modify TrafficEventPushReceiver to process CORBA events pushed for the new event.
- Create a JPanel class to display event data that is specific to the new event and add it to the event dialog.

7.5 ADDING A NEW ALGORITHM

The steps involved in adding a new algorithm to the CHART II system are similar to the steps for adding a new device. The various steps involved are as follows.

- Define the IDL.

- Add a service application module to create the CORBA objects that implement the algorithm.
- Add an installable GUI module to display the algorithm-specific screens.

8. Additional CHART II Software Development Resources

8.1 CHART II DEVELOPERS WEB SITE

During the course of development, a CHART II developers web site has been created and maintained to store information identified as useful for training or reference. It was created to provide ready access to electronic versions of useful documentation and to serve as a source for information that might be available via various reference documents but not easily accessible. It contains information gained through experience (e.g., which functions of CORBA are useful and how do they work) and information that was often related during training or coaching by more senior developers on the project.

The site contains the following:

- Links to design documents for the latest CHART II release/build.
- CHART II Contacts List.
- Links to JavaDocs for the Java Development Kit, CHART II and the CORBA ORB – JavaDocs are code documentation produced by a Java documentation tool in an HTML format that facilitates navigation.
- Links to project coding standards for both Java and IDL code.
- CHART II Programming Style Guidelines – This is a set of “home grown” guidance gleaned from experience that is intended to supplement the coding standards with “best practice” or “project practice” information.
- A link to the results of code reviews from the latest release of CHART II.
- A link to Unit Test plans including a template, test plans and test results for the latest release.
- An individual’s note on why and how to use the JBuilder Java development tool.
- A link to the database design for the latest release.
- A tip for how to resolve some common database problems.
- A CHART II CORBA primer.
- A cross-reference of developer machine names and IP addresses (outdated but useful during testing).
- A list of useful utilities.

Please contact MDSHA for information on accessing the web site.

8.2 CHART II WEB SITE READING ROOM

The "Reading Room" on the CHART II web site also contains additional documentation related to development of the CHART II system, including architecture, design, integration and test, and operations and maintenance resources. The "Reading Room" web site is located at www.chart.state.md.us/readingroom/readingroom.asp.

8.3 RECOMMENDED THIRD-PARTY DOCUMENTATION

Table 6, below, identifies additional ITS and software documentation and resources suggested for CHART II software development.

Table 6 – Recommended Third-Party Documentation

Category	Document/Resource
Java Programming	www.java.sun.com
CORBA Development	Java Programming with CORBA, Gerald Brose, Andreas Vogel, and Keith Duddy, 2001. www.iona.org, www.corba.org
Oracle Database Development	www.oracle.com
Object Oriented Design and Development	UML Distilled, Applying the Standard Object Modeling Language, Martin Fowler with Kendall Scott, June 1998.
TMDD (Transportation Management Data Dictionary) Standards	www.ite.org/tmdd, www.tmdd.org
NTCIP Standards	www.ntcip.com

Appendixes

APPENDIX A – LIST OF ACRONYMS

AOC	Authority (MdTA) Operations Center
ATM	Asynchronous Transfer Mode
AVCM	ATM Video Control Module
AVL	Automatic Vehicle Location
BAA	Business Area Architecture
CCTV	Closed Circuit Television
CHART	Coordinated Highways Action Response Team
CI	Configuration Item
CM	Configuration Management
CMM	Capability Maturity Model
CO	Central Office
CORBA	Common Object Request Broker Architecture
COTS	Commercial Off-the-Shelf
CSC	Computer Sciences Corporation
DB	Database
DBA	Database Administrator
DBMS	Database Management System
DMS	Dynamic Message Sign
DNS	Domain Name System
DTMF	Dual Tone Multi-Frequency
EORS	Emergency Operations Reporting System
FCA	Functional Configuration Audit
FHWA	Federal Highway Administration
FITM	Freeway Incident Traffic Management
FMS	Field Management Station
FTP	File Transfer Protocol
GIS	Geographic Information System
GUI	Graphical User Interface
HAR	Highway Advisory Radio
ICD	Iterative Custom Development
IDL	Interface Definition Language
IEN	Information Exchange Network
IOR	Interoperable Object Reference
ISDN	Integrated Services Digital Network
ITS	Intelligent Transportation Systems
IV&V	Independent Validation and Verification
JDK	Java Developer's Kit
JHU/APL	Johns Hopkins University/Applied Physics Laboratory
LATA	Local Access and Transport Area
MDSHA	Maryland State Highway Administration
MdTA	Maryland Transportation Authority
MFC	Microsoft Foundation Class
MSP	Maryland State Police
NTCIP	National Transportation Communication ITS Protocol
ORB	Object Request Broker

PBFI	PB Farradyne Inc.
PCA	Physical Configuration Audit
POA	Portable Object Adapter
POC	Point of Contact
POTS	Plain Old Telephone Service
PR	Problem Report
QA	Quality Assurance
QSS	Quality Systems and Software
R1B3	Release 1, Build 3 of the CHART II System
RFP	Request for Proposal
RTMS	Remote Traffic Microwave Sensor
S&P	Standards and Procedures
SAN	Storage Area Network
SCAN	Surface Condition Analyzer (from Surface Systems, Inc.)
SCN	Software Control Notice
SDP	Software Development Plan
SEI	Software Engineering Institute (Carnegie Mellon University)
SOC	Statewide Operations Center
TMDD	Transportation Management Data Dictionary
TOC	Transportation Operations Center
TRANSCOM	Transportation Operations Coordinating Committee
TSS	Traffic Sensor System
TTS	Text-To-Speech
UMD-CATT	University of Maryland Center for Advanced Transportation Technology
UML	Unified Modeling Language
X/AD	Accelerated Application Development
XML	Extensible Markup Language

APPENDIX B – SYSTEM PROPERTY FILES

Every service in the CHART II system has a Property File associated with it. These files provide each service with parameters the service should use at startup. If these values are changed while the service is running, the service must be restarted for the new value to take effect. These properties give a system administrator access to the various service options.

The following Property Files are listed in this Appendix:

- CHART2GUI.props
- DMSService.props
- EORSService.props
- HARService.props
- MsgUtilityService.props
- Notserv.props
- TrafficEventService.props
- TSSService.props
- UMService.props

CHART2GUI.props

```
#
# This is the properties file used by the CHART 2 GUI Application.
# All properties in this file are added to the application properties.
# Vendor code like the ORB and Oracle access properties from the
# system properties and thus including their properties in this
# file allows all properties needed by the application to reside
# in the same file.
#


#
# Installable modules to load
#
GUI.ModuleClassName0 = CHART2.GUIUserManagementModule.GUIUserManagementModule
GUI.ModuleClassName1 = CHART2.GUIDMSModule.GUIDMSModule
GUI.ModuleClassName2 = CHART2.GUILibraryModule.GUILibraryModule
GUI.ModuleClassName3 = CHART2.GUIDictionaryModule.GUIDictionaryModule
GUI.ModuleClassName4 = CHART2.GUIPlanModule.GUIPlanModule
GUI.ModuleClassName5 = CHART2.GUIServiceModule.GUIServiceModule
GUI.ModuleClassName6 = CHART2.GUITSSModule.GUITSSModule
GUI.ModuleClassName7 = CHART2.GUIHARModule.GUIHARModule
GUI.ModuleClassName8 = CHART2.GUISHAZAMModule.GUISHAZAMModule
GUI.ModuleClassName9 = CHART2.GUITrafficEventModule.GUITrafficEventModule
GUI.ModuleClassName10= CHART2.GUIFieldCommsModule.GUIFieldCommsModule


#
# Level of logging for the log file.  Valid values are DEBUG or PRODUCTION
#
GUI.LogLevel = PRODUCTION

#
# Number of days to keep the log files around.
#
GUI.LogKeepDays = 7

#
# Number of seconds between discovery cycles after the GUI has
# been running for a long time (discovery may happen more frequently
# after the GUI is first started).  This is used to find new event
# channels and new objects in the system.
#
GUI.DiscoveryPeriodSeconds = 7200

#
# Indicates how long (in seconds) to display command status objects after
# the command has been completed.
#
GUI.CompletedCommandStatusRemovalDelaySeconds = 60

#
# Name of the operations center that this GUI should log into
#
GUI.OperationsCenterName = AOC

#
# This command is used to start the Chart Chat program. The button for
# Chart Chat will not be visible on the toolbar unless this command
# is populated.  If the Chart Chat program is not installed comment
# out the command, as shown below, using the #.The command must be
# of the same format used to start the program from the command line
# while in the Chart2GUI directory.
# An example is shown below.
#
```

```
GUI.ChartChat = javaw -classpath ../../lib/ChartChat.jar ChartChat.Client.ChartChat
host=

#
# This command is used to start the SHADE. The button for
# SHADE will not be visible on the toolbar unless this command
# is populated.  If the SHADE program is not installed comment
# out the command, as shown below, using the #.The command must be
# of the same format used to start the program from the command line
# while in the Chart2GUI directory.
# An example is shown below.
#
#GUI.Shade = c:/Program Files/Internet Explorer/iexplore http://www.shadesite.org

#
# Directory where image files are located for the GUI
#
#GUI.ImagesDirectory = /images

#
# CHART2 HelpSet Location for online help.
#
GUI.HelpSetLocation = Help/CHART2HelpSet.hs



#
# The frequency with which each service is pinged
# to check whether it is alive, in seconds.
# This is used only if the GUIServiceModule is installed.
#
#GUIServiceModule.ServicePollIntervalSec = 60


#
# The frequency with which the GUIServiceModule
# queries the trading service for new services, in seconds.
# This is used only if the GUIServiceModule is installed.
#
#GUIServiceModule.ServiceDiscoveryIntervalSec = 900


#
# Interval that the GUI will wait when attempting to contact a server.
# This property governs only the initial connect time to the service in question.
Once
# the connection has been established, the GUI timeout is dependent on the ORB
# request timeout set below.
#
# If this value is not specified, the timeout will be infinite.  This could cause
eternal
# hangs in the GUI if a service machine is shutdown.
#
GUI.ORB_connect_timeout_millis = 5000

#
# Interval that the GUI will wait when attempting to contact a server.
# This property governs the total time to respond to the service in question.
#
# If this value is not specified, the timeout will be infinite.  This could cause
eternal
# hangs in the GUI if a service machine is shutdown.
#
GUI.ORB_request_timeout_millis = 30000

#
# List of DMS message formatters.  These default formatters will be added to the list
```

```
# of DMS geometries in the system to compile a list of possible formatter options for
# the user when he/she is editing a DMS stored message.  Ideally, these should be set
# to match the sign geometries in the deployed system.  If this is the case, the user
# will only see the default formatters if no DMS objects have been discovered yet.
#
# Each formatter should contain the following comma delimited information
# Formatter type                - String Valid values are - SHA
# Formatter Character Width     - Integer represents the vertical pixels for each
#                                 character in the sign's font.  Specified in NTCIP
#                                 as vmsCharacterWidthPixels.
# Formatter Character Height    - Integer represents the horizontal pixels for each
#                                 character in the sign's font.  Specified in NTCIP
#                                 as vmsCharacterHeightPixels.
# Formatter Total Pixel Width   - Integer represents the total horizontal pixels on
the sign.
#                                 Specified in NTCIP as vmsSignWidthPixels.
# Formatter Total Pixel Height  - Integer represents the total vertical pixels on the
sign.
#                                 Specified in NTCIP as vmsSignHeightPixels.
# Formatter Max pages           - Integer Specified in NTCIP as vmsMaxPages.
#
GUIDMSModule.MultiFormatter0 = SHA, 5, 7, 50, 21, 2
GUIDMSModule.MultiFormatter1 = SHA, 5, 7, 90, 21, 2
GUIDMSModule.MultiFormatter2 = SHA, 5, 7, 100, 21, 2
GUIDMSModule.MultiFormatter3 = SHA, 5, 7, 105, 21, 2


#
# List of DMS font files to load.
#
GUIDMSModule.DMSFont0 = fw07x5s.fnt


#
# Name of the default lane configuration that appears in
# the list of lane configurations when a new event is opened.
# If this value is not specified, it will default to: 4 lanes each direction with
shoulders
#
GUITrafficEventModule.DefaultLaneConfigName = 4 lanes each direction with shoulders

#
# Value of the delay after the gui starts up when the commlog
# should be refreshed with the values from the database
#
GUITrafficEventModule.DelayAfterStartupForCommLogRefresh = 1800

#
# Value of the default delay for subsequent commlog refresh's
#
GUITrafficEventModule.DelayForSubsequentCommLogRefresh = 14400

#
# Settings for the Orbacus ORB.  Refer to the ORB documentation
# for usage.
#

#
# Indicates the output level for diagnostic messages printed by the ORB.
#
#ooc.orb.trace.connections=10

#
# Indicates the client-side concurrency model.
#
ooc.orb.conc_model=threaded
```

```
#
# Indicates the server-side concurrency model
#
ooc.orb.oa.conc_model=thread_pool

#
# Indicates how many threads are in the thread pool;
#
ooc.orb.oa.thread_pool=50

#
# This setting is mainly for preventing user from bringing up several instances
# of the GUI on the same machine. (see LEVA31)
#
ooc.orb.oa.port=1000

#
# Indicates the trading service that will be used to lookup objects.
#
ooc.orb.service.TradingService=corbaloc::chart2-svr5:8002/TradingService
```

DMSService.props

```
#
# This is the properties file used by the DMS Service Application.
# All properties in this file are added to the system properties.
# Vendor code like the ORB and Oracle access properties from the
# system properties and thus including their properties in this
# file allows all properties needed by the application to reside
# in the same file.
#

#
# Name of this service.
#
DefaultServiceApplication.ServiceName=DMSService

#
# Refresh Trader Offers flag.  Set this flag to true when you want
# all previous trader offers made from within this app to be withdrawn
# prior to starting service modules.  Also allows service modules
# to publish objects that are de-persisted from the database.
#
DefaultServiceApplication.RefreshOffers=true

#
# DB Connection String for Oracle Database
#
DefaultServiceApplication.DBConnectString=jdbc:oracle:thin:@chart2-svr3:1526:SOC3

#
# User name to be used to connect to the database
#
DefaultServiceApplication.DBUserName=DMSService

#
# Password to be used to connect to the database
#
DefaultServiceApplication.DBPassword=chart2dms

#
# Maximum number of connections for DBConnectionManager to allocate.
#
DefaultServiceApplication.DBMaxConnections=10

#
# Interval for DBConnectionManager to scan list of in-use connections
# to detect threads that have exited without releasing connection.
# (milliseconds)
#
DefaultServiceApplication.DBConnectionMonitorInterval=5000

#
# Interval for OperationsLog to write queued items to the database.
# (milliseconds)
#
DefaultServiceApplication.OperationLogInterval=5000

#
# Network connection site of the service
#
DefaultServiceApplication.NetConnectionSite=chart2-svr5

#
# Name of file where trader offers are stored so that they
# can be cleaned up later using the RefreshOffers flag.
#
```

```
DefaultServiceApplication.OfferFilename=DMSServiceOffers.txt

#
# Name to be prepended to the date to name the log file
#
DefaultServiceApplication.LogFileName=DMSService_

#
# Logging level. Valid values are production or debug
#
DefaultServiceApplication.LogFileLevel=production

#
# Number of days to keep old log files
#
DefaultServiceApplication.LogFileKeepDays=7

#
# Number of tries to make when resolving references for initial services. (Default 1)
#
DefaultServiceApplication.ResolveInitialReferenceTries=2

#
# Amount of time to wait for establishing a connection for requests to other services
# specified in milliseconds.
#
DefaultServiceApplication.ORBConnectTimeoutMillis=5000

#
# Amount of time to wait for requests to other services to complete, specified
# in milliseconds.
#
DefaultServiceApplication.ORBRequestTimeoutMillis=120000

#
# Class names of ServiceApplicationModules to be instantiated
# and run by this service.
#
DefaultServiceApplication.ModuleClassName0=CHART2.DMSControlModule.DMSControlModule

#
# Property key used to specify if the service needs
# notification event service access.
# By default it is set to false. Refer ServiceApplicationProperties.jav
#
DefaultServiceApplication.InitNotificationService=true

#
# System profile cache refresh interval, i.e. the amount of time this service should
refresh its system
# properties cache with the actual system properties (in minutes)
#
DMSControlModule.SystemPropertiesRefreshMins = 60

#
# This property controls the behavior of the arbitration queue and, therefore,
# affects the message that is displayed on an online device.
# Each time the arbitration queue evaluates (either due to automatic background
# re-evaluation, or due to entries added, removed, re-prioritized) the queue
# inspects each entry to determine if it should be allowed on the device.  If
# all checks succeed, the queue checks with the entry owner (the object that placed
# the entry on the queue in the first place) and verifies that the entry should be
# allowed.  If the owner can be reached it will return either VALID or INVALID.  If
# it cannot be reached, however, then the entry is considered marginal.  Meaning, the
# owner may no longer want this message on the device, but we cannot tell.  This
# property controls how the arbitration queue will deal with such entries.  If this
# property is set to 'true', then marginal entries will be allowed to be displayed
```

```
# on the DMS.  In this case the entry will be seen by motorists until the arb queue
# entry times out, at which time the entry will be removed from the queue entirely.
# If, on the other hand, this property is set to false the arbitration queue will not
# allow the message for a marginal entry to be displayed.  The entry will remain
# on the queue until it times out, but will not be allowed to be active unless a
# successful verification call is made to the entry owner..
#
# Valid values:
#  true - Marginal entries are displayed on the DMS.
#  false - Marginal entries are not displayed on the DMS.
#
# Default value: true
#
# see DMSControlModule.EvaluateQueueSecs for frequency of queue background evaluations
# see DMSControlModule.QueueEntryTimeoutMins for duration of queue entry timeout
#
DMSControlModule.AllowMarginalQueueEntriesToBeActive=true

#
# This property specifies the delay, in seconds between executions of the
# CheckCommLossTask.  This is NOT the Comm Loss Timeout.  Each DMS holds and
# controls its own Comm Loss Timeout.  (The Comm Loss Timeout is a configurable
# property of each DMS, although depending on the DMS model, it may or may not
# be relevant or changeable).  This CommLossTimerDelaySecs property specifies
# how often each DMS will be checked to see if it has exceeded its own Comm Loss
# Timeout.  Note that when a DMS exceeds its Comm Loss Timeout, it is not
# immediately recognized by the Chart II software, and it will not be recognized
# until the the DMSes are next checked.  How often that happens is controlled by
# this property.  So if the Comm Loss Timeout for a particular DMS is 5 minutes,
# and this property is set to 30 seconds, it may virtually immediately or it be
# up to 30 seconds after the timeout occurs before the DMS is checked again and
# the timeout is detected.  The setting of this parameter is a tradeoff between
# accurate reflection of reality and the expense of checking for the timeouts.
# However, in this case checking for timeouts is relatively inexpensive (i.e.,
# no comms to signs are attempted in checking for Comm Loss Timeout), so this
# property can be set fairly low without incurring significant overhead.  30
# seconds is the default.  5-30 seconds is a reasonable range.

#
DMSControlModule.CommLossTimerDelaySecs=30

#
# This property specifies the delay, in seconds, between executions of the
# PollDMSTask.  This is NOT the polling interval.  Each DMS holds and controls
# its own polling interval (which is a configurable property of each DMS).  This
# property specifies how often each DMS will be asked to see if it has exceeded
# its own timing interval.  Note also that this property should be considerably
# smaller than the smallest polling interval for any DMS.  If a DMS is set for a
# polling interval of 4 minutes and this parameter is set to 20 seconds, that
# DMS will sometimes go as much as (or slightly more than) 4:19 between polls.
# (If this parameter were set to 120 seconds, that same DMS might go as much as
# [or slightly more than] 5:59 between polls -- not good if that DMS has a Comm
# Loss Timeout of five minutes.)  Note that there may be other slight delays,
# amounting to a perhaps few seconds or so, in the polling process.  Therefore,
# take care that each DMS's polling interval plus the value of this property is
# less than the DMS's Comm Loss Timeout by a safe margin.  The setting of this
# parameter is a tradeoff between the consistency of steady, rhythmic polling
# (not too important as long as it occurs often enough in all cases) and the
# expense of checking each DMS for the need.  While checking for the need to
# poll is relatively inexpensive (no comms to signs are attempted unless it
# actually is time to poll and even that occurs asynchronously, on a separate
# thread), there is no reason to attempt to achieve precise, highly rhythmic
# polling.  This parameter needs to be set only low enough to achieve polls
# often enough.  20 seconds is the default.  15-30 seconds is a reasonable
# range.
#
DMSControlModule.PollTimerDelaySecs=20
```

```
#
# This property specifies how long the background polling task should wait for a
# port to use to poll a DMS.  Note that this property does not affect how long
# to wait for a port for a user command to a DMS (including a poll command
# requested by the user).  This affects background polling only, which, because
# it has the lowest priority for acquiring a port, may need to wait longer than
# a DMS waits for user-directed commands to the DMS.  Because background polling
# is a low priority task, if this property is not set long enough, polls will
# fail when all the ports on a PortManager are being kept busy by user commands
# (perhaps entirely for other DMSes accessed from the same PortManager).  This
# could cause DMSes to go into Comm Loss timeout and blank themselves.  The
# default for this parameter is 90 seconds.  This parameter should be set to at
# least two times the typical usage time of the slowest type of modem (POTS,

# about 25 seconds per usage): that is, at least 50-60 seconds.  A maximum
# greater than a Comm Loss timeout is of little value.  The default is 90
# seconds.
#
DMSControlModule.PollPortWaitTimeSecs=90

#
# This property specifies the interval, in seconds, used by the Chart2DMSFactory
# to determine how frequently to check for DMSs that are controlled by a
# Operations Center which has no users logged in.  A DMS is controlled by an
# Operations Center if it is online with a message on it or if it is in
# maintenance mode.  The setting of this parameter is a tradeoff between
# detecting an abandoned DMS very quickly and the expensive of checking for
# abandoned DMSes.  In this case, the necessity for immediate detection of
# abandoned DMSes is low, the risk incurred by delayed detection is relatively
# low, and the cost to check is moderate.  It does require some communication
# between Operations Centers to check.  300 seconds (5 minutes) is the default.
#
DMSControlModule.SharedResMonIntSecs=300

# This property specifies how often (in seconds) the DMS Service will write a
# timestamp out to disk.  This timestamp is used when the DMS Service starts up
# in order to estimate the time the DMS Service has been down.  This property should
# be set small enough that the time down estimate is reasonable, but not so small
# that the DMS Service is constantly writing timestamps.  A value between 15 and 60
# seconds is reasonable.  The default is 30 seconds.
#
DMSControlModule.RecoveryTimerDelaySecs=30

# This property specifies the type of debug-level logging to do.  This
# property has no effect if DefaultServiceApplication.LogFileLevel is
# set to production.  This property is intended for the use of
# software engineering staff only.  This property, if set, will change
# the types of additional debug messages written to the DMSService log
# file.  These messages would have little meaning for operational
# staff.  (For programmers: the meanings of each character are defined
# in Chart2DMSImpl.log().
#
###DMSControlModule.LogFlags=

#
# Property key used to determine the system wide maximum number of pages
# in a combined message, if one is not defined for a DMS specifically. If
# this value is -1, the maximum number of pages for the dms is used. If
# this value is defined and > 0, either this value or the maximum pages
# for dms is used whichever is smaller. If this value is 0, it is invalid.
# The default value is 2.
#
DMSControlModule.MaximumPagesInCombinedMessage=2

#
# Amount of time, specified in minutes, that an arb queue entry is allowed to stay on
```

```
# the arbitration queue even though its owner cannot be contacted to verify that it is
# still a valid entry.  Entries in this state will not affect the message that ends up
on
# the DMS, but will slow evaluation of the queue until they are removed.  If this
property
# is not specified, it will default to 60 minutes.  This value can be set to an
arbitrarily
# large value to keep the entries from timing out.  The max allowable value is (2^32)-
1 =
# 2147483647.  Using a value this large would allow the entries to timeout after
existing
# on the queue for approximately 4085 years.
#
DMSControlModule.QueueEntryTimeoutMins=60

#
# Frequency with which the Arbitration queue should be re-evaluated.  The system will
re-evaluate
# the arbitration queue for a device each time this interval expires.  If that
evaluation results
# in the same message as the message currently on the device, the system will take no
further action.
# If the evaluation results in a different message than the message currently on the
device, the
# system will attempt to put the message resulting from the evaluation on the device.
This process
# is only performed when the device is in online mode.  This value will default to 5
minutes (300) if
# not specified or if the value specified is invalid.  This property also has a
minimum allowed value
# of 5 minutes (300).  If a value less than 5 minutes is specified, it will be ignored
and the minimum
# 5 minute value will be used.
#
DMSControlModule.EvaluateQueueSecs=300

#
# This property specifies whether FP9500 type of DMSs should check for 48V battery
failure
* in the status data.
# Note (02/07/02): this property has been added for PR LevA115. The protocol
# was going to be changed later by the vendor to support this but a fix is
# being included ahead of the change so that we don't have to do another
# build just for this.  This property needs to be deleted later when
# it is no longer needed (revisit during R2B1 intergration).
#
DMSControlModule.FP9500Allow48VBatteryFailureDetection=false

#
# Oracle db driver class
#
jdbc.drivers=oracle.jdbc.driver.OracleDriver

#
# Settings for the Orbacus ORB.  Refer to the ORB documentation
# for usage.
#
#ooc.orb.trace.connections=10
ooc.orb.conc_model=threaded
ooc.orb.oa.conc_model=thread_pool
ooc.orb.oa.thread_pool=10
ooc.orb.oa.port=8004
ooc.orb.service.EventChannelFactory=corbaloc::chart2-
svr5:8001/DefaultEventChannelFactory
ooc.orb.service.NotificationService=corbaloc::chart2-
svr5:8010/DefaultEventChannelFactory
ooc.orb.service.TradingService=corbaloc::chart2-svr5:8002/TradingService
```

EORSService.props

```
#
# EORSService properties
#
# This is the properties file used by the EORS Service Application.
# All properties in this file are added to the system properties.
# Vendor code like the ORB and Oracle access properties from the
# system properties and thus including their properties in this
# file allows all properties needed by the application to reside
# in the same file.
#

#
# Name of this service.
#
DefaultServiceApplication.ServiceName=EORSService

#
# Refresh Trader Offers flag.  Set this flag to true when you want
# all previous trader offers made from within this app to be withdrawn
# prior to starting service modules.  Also allows service modules
# to publish objects that are de-persisted from the database.
#
DefaultServiceApplication.RefreshOffers=true


#
# DB Connection String for SQL Server Database
#
DefaultServiceApplication.DBConnectString=jdbc:inetdae:@EOCNTD0:1433

#
# User name to be used to connect to the database
#
DefaultServiceApplication.DBUserName=CHART

#
# Password to be used to connect to the database
#
DefaultServiceApplication.DBPassword=

#
# Name of EORS database
#
EORSInitialCatalog=EORS

#
# Maximum number of connections for DBConnectionManager to allocate.
#
DefaultServiceApplication.DBMaxConnections=3

#
# Interval for DBConnectionManager to scan list of in-use connections
# to detect threads that have exited without releasing connection.
# (milliseconds)
#
DefaultServiceApplication.DBConnectionMonitorInterval=5000

#
# Interval for OperationsLog to write queued items to the database.
# (milliseconds)
#
DefaultServiceApplication.OperationLogInterval=5000

#
```

```
# Network connection site of the service
#
DefaultServiceApplication.NetConnectionSite=chart2-svr5


#
# Name of file where trader offers are stored so that they
# can be cleaned up later using the RefreshOffers flag.
#
DefaultServiceApplication.OfferFilename=EORSServiceOffers.txt

#
# Name to be prepended to the date to name the log file
#
DefaultServiceApplication.LogFileName=EORSService_

#
# Logging level. Valid values are production or debug
#
DefaultServiceApplication.LogFileLevel=production

#
# Number of days to keep old log files
#
DefaultServiceApplication.LogFileKeepDays=7


#
# Number of tries to make when resolving references for initial services. (Default 1)
#
DefaultServiceApplication.ResolveInitialReferenceTries=2

#
# Amount of time to wait for establishing a connection for requests to other services
# specified in milliseconds.
#
DefaultServiceApplication.ORBConnectTimeoutMillis=5000

#
# Class names of ServiceApplicationModules to be instantiated
# and run by this service.
#
DefaultServiceApplication.ModuleClassName0=CHART2.EORSModule.EORSModule

#
# Determines if this service needs to use the CORBA
# event service.  This will default to true.
# Valid values : true - Uses the event service, false does not use it.
#
DefaultServiceApplication.InitEventService=false

#
# Determines if this service needs to use a DBMS.
# This will default to true.  If set to true, the app will
# use the JDBC properties in this file to connect to the database and
# will not run if the database cannot be connected to.
# Valid values : true - Uses a database, false does not use a database.
#
DefaultServiceApplication.InitDB=true

#
# SQLServer db driver class
#
jdbc.drivers=com.inet.tds.TdsDriver

#
# Settings for the Orbacus ORB.  Refer to the ORB documentation
```

```
# for usage.
#
ooc.orb.conc_model=threaded
ooc.orb.oa.conc_model=thread_pool
ooc.orb.oa.thread_pool=10
ooc.orb.oa.port=8009
ooc.orb.service.TradingService=corbaloc::chart2-svr5:8002/TradingService
```

HARService.props

```
#
# This is the properties file used by the HAR Service Application.
# All properties in this file are added to the system properties.
# Vendor code like the ORB and Oracle access properties from the
# system properties and thus including their properties in this
# file allows all properties needed by the application to reside
# in the same file.
#

#
# Name of this service.
#
DefaultServiceApplication.ServiceName=HARService

#
# Refresh Trader Offers flag.  Set this flag to true when you want
# all previous trader offers made from within this app to be withdrawn
# prior to starting service modules.  Also allows service modules
# to publish objects that are de-persisted from the database.
#
DefaultServiceApplication.RefreshOffers=true

#
# DB Connection String for Oracle Database
#
DefaultServiceApplication.DBConnectString=jdbc:oracle:thin:@chart2-svr3:1526:SOC3

#
# User name to be used to connect to the database
#
DefaultServiceApplication.DBUserName=HARService

#
# Password to be used to connect to the database
#
DefaultServiceApplication.DBPassword=chart2har

#
# Maximum number of connections for DBConnectionManager to allocate.
#
DefaultServiceApplication.DBMaxConnections=10

#
# Interval for DBConnectionManager to scan list of in-use connections
# to detect threads that have exited without releasing connection.
# (milliseconds)
#
DefaultServiceApplication.DBConnectionMonitorInterval=5000

#
# Interval for OperationsLog to write queued items to the database.
# (milliseconds)
#
DefaultServiceApplication.OperationLogInterval=5000

#
# Network connection site of the service
#
DefaultServiceApplication.NetConnectionSite=chart2-svr5

#
# Name of file where trader offers are stored so that they
# can be cleaned up later using the RefreshOffers flag.
#
```

```
DefaultServiceApplication.OfferFilename=HAROffers.txt

#
# Name to be prepended to the date to name the log file
#
DefaultServiceApplication.LogFileName=HARService_

#
# Logging level. Valid values are production or debug
#
DefaultServiceApplication.LogFileLevel=production

#
# Number of days to keep old log files
#
DefaultServiceApplication.LogFileKeepDays=7

#
# Number of tries to make when resolving references for initial services.
# Default value is 1.
#
DefaultServiceApplication.ResolveInitialReferenceTries=2

#
# Amount of time to wait for establishing a connection for requests to other
# services, specified in milliseconds.
#
DefaultServiceApplication.ORBConnectTimeoutMillis=5000

#
# Amount of time to wait for requests to other services to complete, specified
# in milliseconds.
#
DefaultServiceApplication.ORBRequestTimeoutMillis=600000

#
# This property specifies whether this service needs notification event service
# access.  (The HAR Service does.)  Refer to ServiceApplicationProperties.java.
# Default value is false.
#
DefaultServiceApplication.InitNotificationService=true

#
# Class names of ServiceApplicationModules to be instantiated and run by this
# this service.
#
DefaultServiceApplication.ModuleClassName0=CHART2.HARControlModule.HARControlModule
DefaultServiceApplication.ModuleClassName1=CHART2.SHAZAMControlModule.SHAZAMControlMod
ule

#
# System profile cache refresh interval, i.e. the amount of time this service should
refresh its system
# properties cache with the actual system properties (in minutes).
#
HARControlModule.SystemPropertiesRefreshMins = 60

#
# This property controls the behavior of the arbitration queue and, therefore,
# affects the message that is displayed on an online device.
# Each time the arbitration queue evaluates (either due to automatic background
# re-evaluation, or due to entries added, removed, re-prioritized) the queue
# inspects each entry to determine if it should be allowed on the device.  If
# all checks succeed, the queue checks with the entry owner (the object that placed
# the entry on the queue in the first place) and verifies that the entry should be
# allowed.  If the owner can be reached it will return either VALID or INVALID.  If
# it cannot be reached, however, then the entry is considered marginal.  Meaning, the
```

```
# owner may no longer want this message on the device, but we cannot tell.  This
# property controls how the arbitration queue will deal with such entries.  If this
# property is set to 'true', then marginal entries will be allowed to be broadcast
# from the HAR.  In this case the entry will be heard by motorists until the arb queue
# entry times out, at which time the entry will be removed from the queue entirely.
# If, on the other hand, this property is set to false the arbitration queue will not
# allow the message for a marginal entry to be broadcast.  The entry will remain
# on the queue until it times out, but will not be allowed to be active unless a
# successful verification call is made to the entry owner..
#
# Valid values:
#  true - Marginal entries are broadcast from the HAR.
#  false - Marginal entries are not broadcast from the HAR.
#
# Default value: true
#
# see HARControlModule.EvaluateQueueSecs for frequency of queue background evaluations
# see HARControlModule.QueueEntryTimeoutMins for duration of queue entry timeout
#
HARControlModule.AllowMarginalQueueEntriesToBeActive=true

#
# This property specifies how frequently, in seconds, the HAR Service should
# check text clips on the HARs to see if they have datestamps that need to be
# refreshed to the current day.
# Default value is 300.
#
HARControlModule.DateStampRefreshIntervalSecs=300

#
# This property specifies how many seconds to wait to obtain a port while doing
# datestamp refreshes.  Since this is a background task, it is reasonable to
# wait longer than would be appropriate for a foreground task, since there will
# not be a user waiting on this task to complete.  Also, it is expected that
# most HARs will attempt to update their datestamps at about the same time,
# shortly after midnight each day.  Waiting longer allows more calls to get
# through at the first opportunity.  This value should be substantially less
# than the DateStampRefreshIntervalSecs, by at least double the runtime expected
# to be taken by the clip(s) which contain at datestamp (otherwise some HARs
# may be unnecessarily updated twice).
# Default value is 240.
#
HARControlModule.DatestampRefreshPortWaitTimeSecs=240

#
# This property specifies the number of seconds that should elapse between
# executions of the background task evaluate queue task.  This is the task which
# continually runs to make sure that the HAR message and HAR notifiers are set
# according to the current top priority queue item(s) (which runs in online mode
# only).  This property controls the retry rate, if for instance, there are no
# ports available when a change in the queue necessitates a change to the HAR
# message, or if a SHAZAM or DMS being used as a notifier cannot be contacted
# through software on the first attempt (if the DMS service is down, for
# instance).
# Default value is 300.
#
HARControlModule.EvaluateQueueSecs=300

#
# This property specifies the number of seconds that the HAR should wait for
# collecting audio data for slot download.  It would prevent the HAR service
# from waiting forever if an Audio Clip Manager or Text To Speech process fails
# to respond to a request for HAR audio.  This value should be set to at least
# double the maximum runtime of a single clip which might ever be stored on the
# HAR.
#
HARControlModule.HARAudioStreamerTimeoutSecs=180
```

```
#
# Property that is used to specify the safety margin subtracted from total time
# believed to be remaining in RAM on the HAR, to be sure that a clip (or clips)
# about to be downloaded to the HAR are certain to fit.  Since the HAR cannot
# communicate to the software, the software would not know if the HAR has
# rejected a download due to space, so this safety margin ensures that the
# software does not attempt to come too close to filling the HAR's RAM down to
# the last second, risking an undetected overflow condition.
# Default value is 10.
#
HARControlModule.HARRuntimeSafetyMarginSecs=10

#
# This property specifies the name of the local audio clip manager, which is to
# be the preferred AudioClipManager to be used by this HAR service.  It is
# expected, but not required, that the this AudioClipManager will be on the same
# host as the HARService.  If set, this property should be set to match the
# value of the AudioClipModule.AudioClipManagerName property in the
# corresponding MsgUtilityService props file (which is suggested to be set to
# the name of the host on which it is running).  If the AudioClipManager
# specified by this property cannot be contacted, the HARService will attempt to
# use any other AudioClipManager it can find.
# Default is to use the name of the host on which the HAR is running as the
# name of the AudioClipManager (i.e., search for an AudioClipManager on the
# local host first).
#
#HARControlModule.LocalAudioClipManagerName=

#
# This property specifies the type of debug-level logging to do within the HAR
# module.  This property has no effect if DefaultServiceApplication.LogFileLevel
# is set to production.   This property is intended for the use of software
# engineering staff only.  This property, if set, will change the types of
# additional debug messages written to the HARService log file.  This messages
# would have little meaning for operational staff.  (For programmers: the
# meanings of each character are defined in HARImpl.log().)
# Default value is set to a reasonable level of debugging output.
#
#HARControlModule.LogFlags=

#
# Property key that is used to specify the maximum runtime of a HAR message, in
# seconds.
# Default value is 120.
#
HARControlModule.MaximumMessageRuntimeSecs=120

#
# This property specifies the minimum time that should elapse between two
# trader queries to rediscover Audio Clip Managers, in seconds.  This means that
# if a new Audio Clip Manager comes online, it may not be discovered and used

# for up to this period of time.
# Default value is 60.
#
HARControlModule.MinimumAudioClipManagerRediscoveryPeriodSecs=60

#
# This property specifies the minimum time that should elapse between two
# trader queries to rediscover Text-To-Speech Converters, in seconds.  This
# means that if a new Text-To-Speech Converter comes online, it may not be
# discovered and used for up to this period of time.
# Default value is 60.
#
HARControlModule.MinimumTTSConverterRediscoveryPeriodSecs=60
```

```
#
# This property specifies the total number of threads in the thread pool of the
# AudioPushThreadManager.  It should be set to a value at or slightly higher
# than the maximum number of monitor slot/monitor broadcast operations expected
# to be underway simulataneously for this one HAR service.
# Default value is 10.
#
HARControlModule.NumPushThreads=10

#
# This property specifies the number of milliseconds to wait after a reset
# command has been issued before redialing the HAR again to attempt to set up
# the HAR back to its original condition before the reset.  This property
# identifies the delay prior to the INITIAL redial.  There is another property
# which controls a (likely somewhat longer) delay prior to a "second chance"
# redial.  (Since it is relatively important that we make all attempt to
# complete the action to restore the HAR to its original condition, two redials
# may be attempted.)
# Default value is 200.
#
HARControlModule.PostResetInitialRedialDelayMillis=200

#
# This property specify the number of milliseconds to wait after a reset
# command has been issued before redialing the HAR again to attempt to set up
# the HAR back to its original condition before the reset.  This property
# identifies the delay prior a SECOND redial.  There is another property which
# controls a (likely somewhat shorter) delay prior to the intial redial.
# (Since it is relatively important that we make all attempt to complete the
# action to restore the HAR to its original condition, we allow for two
# redials.)  If this property is set to -1, the secondary redial is not
# attempted.
# Default value is 3000 (3 seconds).
#
HARControlModule.PostResetSecondaryRedialDelayMillis=3000

#
# This property specifies the name of the Text To Speech (TTS) Converter which
# is to be the preferred for use this HAR service.  It is expected, but not
# required, that the this TTS Converter will be on the same host as the
# HARService.  If set, this property should be set to match the value of the
# DefaultServiceApplication.NetConnectionSite property in the corresponding
# MsgUtilityService props file (which is suggested to be set to the name of the
# host on which it is running).  If the TTS Converter specified by this property
# cannot be contacted, the HARService will attempt to use any other TTS
# Converter it can find.
# Default is to use the name of the host on which the HAR is running as the
# name of the TTSConverter (i.e., search for an TTS Converter on the local host
# host first).
#
#HARControlModule.PreferredTTSConverter=

#
# This property specifies the amount of time, in minutes, that an arb queue
# entry is allowed to stay on the arbitration queue even though its owner cannot
# be contacted to verify that it is still a valid entry.  Entries in this state
# will not affect the message that ends up on the device, but will slow
# evaluation of the queue until they are removed.  This property can be set to
# an arbitrarily large value to keep the entries from timing out.  The max
# allowable value is (2^32)-1 = 2147483647.  Using a value this large would
# allow the entries to timeout after existing on the queue for approximately
# 4085 years.
# Default value is 60.
#
HARControlModule.QueueEntryTimeoutMins=60

#
```

```
# This property specifies the interval, in seconds, used by HAR Service to
# determine how frequently to check for HARs in maintenance mode but whose
# controlling op center has no users logged in.
# Default value is 300.
#
HARControlModule.SharedResMonIntSecs=300

#
# This property specifies the number of seconds the HAR service will wait for
# a response after sending an activate HAR Notice request to a SHAZAM (or a DMS
# acting as a SHAZAM).
# Default value is 60.
#
HARControlModule.ShazamActivateTimeoutSecs=60

#
# This property specifies the number of seconds the HAR service will wait for
# a response after sending a deactivate HAR Notice request to a SHAZAM (or a DMS
# acting as a SHAZAM).
# Default value is 60.
#
HARControlModule.ShazamDeactivateTimeoutSecs=60

#
# This property specifies the number of seconds the HAR service will wait for
# a response after sending a request to a SHAZAM to put it in maintenance mode
# as the HAR is going into maintenance mode.
# Default value is 60.
#
HARControlModule.ShazamMaintTimeoutSecs=60

#
# This property specifies the number of seconds the HAR service will wait for
# a response after sending a request to a SHAZAM to take it offline as the HAR
# is going offline.
#
HARControlModule.ShazamOfflineTimeoutSecs=60

#
# This property specifies the number of seconds the HAR service will wait for
# a response after sending a request to a SHAZAM to put it online as the HAR is
# going online.
# Default value is 60.
#
HARControlModule.ShazamOnlineTimeoutSecs=60

#
# This property specifies the number of milliseconds the telephony board will wait
# after generating a DTMF tone before generating the next tone while issuing commands
# to the HAR device.
# Default value is 500 in milliseconds (0.5 seconds)
#
HARControlModule.DTMFInterToneDelayMillis=500

#
# This property specifies the number of milliseconds the HAR service will wait
# between two successive commands while issuing commands to the HAR device.
# Default value is 500 in milliseconds (0.5 seconds)
#
HARControlModule.DTMFInterCommandDelayMillis=500

#
# This property specifies the number of milliseconds the telephony board will wait
# after generating a DTMF tone before generating the next tone while issuing commands
# to the SHAZAM device.
# Default value is 500 in milliseconds (0.5 seconds)
#
```

```
SHAZAMControlModule.DTMFInterToneDelayMillis=500

#
# This property specifies the number of milliseconds the SHAZAM service will wait
# between two successive commands while issuing commands to the SHAZAM device.
# Default value is 500 in milliseconds (0.5 seconds)
#
SHAZAMControlModule.DTMFInterCommandDelayMillis=500

#
# This property specifies the type of debug-level logging to do within the
# SHAZAM module of the HAR service.  This property has no effect if
# DefaultServiceApplication.LogFileLevel is set to production.  This property is
# intended for the use of software engineering staff only.  This property, if
# set, will change the types of additional debug messages written to the
# HARService log file.  This messages would have little meaning for operational
# staff.  (For programmers: the meanings of each character are defined in
# SHAZAMImpl.log().)
#
###SHAZAMControlModule.LogFlags=

#
# Specifies how long to wait to obtain a port while doing background refreshes.
# This does not affect forced refresh on demand requested by a user on a SHAZAM
# in maintenance mode, only regularly scheduled background refreshes.
# Default value is 25.
#
SHAZAMControlModule.RefreshPortWaitTimeSecs=25

#
# Specifies the number of seconds that should elapse between executions of the
# RefreshSHAZAMTask.
# Default value is 20.
#
SHAZAMControlModule.RefreshTimerDelaySecs=20

#
# This property specifies the interval used by SHAZAM Service to determine how
# frequently to check for SHAZAMs in maintenance mode but whose controlling op
# center has no users logged in.
# Default value is 300.
#
SHAZAMControlModule.SharedResMonIntSecs=300


#
# Oracle db driver class
#
jdbc.drivers=oracle.jdbc.driver.OracleDriver

#
# Settings for the Orbacus ORB.  Refer to the ORB documentation
# for usage.
#
ooc.orb.trace.connections=10
ooc.orb.conc_model=threaded
ooc.orb.oa.conc_model=thread_pool
ooc.orb.oa.thread_pool=20
ooc.orb.oa.port=8011
ooc.orb.service.EventChannelFactory=corbaloc::chart2-
svr5:8001/DefaultEventChannelFactory
ooc.orb.service.NotificationService=corbaloc::chart2-
svr5:8010/DefaultEventChannelFactory
ooc.orb.service.TradingService=corbaloc::chart2-svr5:8002/TradingService
```

MsgUtilityService.props

```
#
# This is the properties file used by the
# Message Utility Service Application.
# All properties in this file are added to the system properties.
# Vendor code like the ORB and Oracle access properties from the
# system properties and thus including their properties in this
# file allows all properties needed by the application to reside
# in the same file.
#

#
# Name of this service.
#
DefaultServiceApplication.ServiceName=MessageUtilityService

#
# Refresh Trader Offers flag.  Set this flag to true when you want
# all previous trader offers made from within this app to be withdrawn
# prior to starting service modules.  Also allows service modules
# to publish objects that are de-persisted from the database.
#
DefaultServiceApplication.RefreshOffers=true


#
# DB Connection String for Oracle Database
#
DefaultServiceApplication.DBConnectString=jdbc:oracle:thin:@chart2-svr3:1526:SOC3

#
# User name to be used to connect to the database
#
DefaultServiceApplication.DBUserName=MessageUtilityService

#
# Password to be used to connect to the database
#
DefaultServiceApplication.DBPassword=chart2mu

#
# Maximum number of connections for DBConnectionManager to allocate.
#
DefaultServiceApplication.DBMaxConnections=10

#
# Interval for DBConnectionManager to scan list of in-use connections
# to detect threads that have exited without releasing connection.
# (milliseconds)
#
DefaultServiceApplication.DBConnectionMonitorInterval=5000

#
# Interval for OperationsLog to write queued items to the database.
# (milliseconds)
#
DefaultServiceApplication.OperationLogInterval=5000

#
# Network connection site of the service
#
DefaultServiceApplication.NetConnectionSite=chart2-svr5

#
# Name of file where trader offers are stored so that they
```

```
# can be cleaned up later using the RefreshOffers flag.
#
DefaultServiceApplication.OfferFilename=MsgUtilityOffers.txt

#
# Name to be prepended to the date to name the log file
#
DefaultServiceApplication.LogFileName=MsgUtilService_

#
# Logging level. Valid values are production or debug
#
DefaultServiceApplication.LogFileLevel=production

#
# Number of days to keep old log files
#
DefaultServiceApplication.LogFileKeepDays=7

#
# Number of tries to make when resolving references for initial services. (Default 1)
#
DefaultServiceApplication.ResolveInitialReferenceTries=2

#
# Amount of time to wait for establishing a connection for requests to other services
# specified in milliseconds.
#

DefaultServiceApplication.ORBConnectTimeoutMillis=5000

#
# Amount of time to wait for requests to other services to complete
# specified in milliseconds.
#
DefaultServiceApplication.ORBRequestTimeoutMillis=30000

#
# Status not known on these 3 properties
#
# Property key used to specify if the service needs database access
#
#DefaultServiceApplication.InitDB=

#
#Property key used to specify if the service needs trading service access
#
#DefaultServiceApplication.InitTradingService=


#
#Property key used to specify if the service needs event service access
#
#DefaultServiceApplication.InitEventService=

#
# Class names of ServiceApplicationModules to be instantiated
# and run by this service.
#
DefaultServiceApplication.ModuleClassName0=CHART2.DictionaryModule.DictionaryModule
DefaultServiceApplication.ModuleClassName1=CHART2.MessageLibraryModule.MessageLibraryM
odule
DefaultServiceApplication.ModuleClassName2=CHART2.PlanModule.PlanModule
DefaultServiceApplication.ModuleClassName3=CHART2.AudioClipModule.AudioClipModule
#DefaultServiceApplication.ModuleClassName4=CHART2.TTSControlModule.TTSControlModule

#
```

```
# Property key used to specifiy the location of the AudioClipManager.
#
#MessageLibraryModule.PreferredAudioClipManager=

#
# Name for the property that specifies the
# number of threads in the thread pool for the audio clip module.
#
AudioClipModule.NumPushThreads=10

#
# Property key that specifies at what time lost
# clips (i.e. clips without an owner) are deleted.
#
AudioClipModule.ClipCleanupTimeOfDay=00:00:00

#
# Property key that specifies the number of seconds
# for the clip cleanup delay.
#
AudioClipModule.ClipCleanupDelaySecs=0

#
# Property key that specifies what types of debug-level logging to do.
#
AudioClipModule.LogFlags=AacDEeFHiIkOoqRSsTUvxZz?~

#
# Property key that specifies what the name of the audio clip manager.
#
#AudioClipModule.AudioClipManagerName=

#
# Property key that specifies how many days to wait before

# deleting stale audio clips from the database.
#
AudioClipModule.DaysToDeleteStaleClips=30

#
# Property key that specifies the file directory location
# where the audio files will be stored.
#
TTSControlModule.AudioFileDirLocation=.\\audio

#
# Property key that specifies the maximum file cache size
# for the audio files in mega bytes.
#
TTSControlModule.MaxCacheSize=100

#
# Property key that specifies the audio push thread pool size.
#
TTSControlModule.AudioPushThreadPoolSize=25


#
# Property key that specifies the hostname or the IP address of the machine on
# which TTSService is running.
#
TTSControlModule.TTSServerHostname =chart2-ws3

#
# Property key that specifies the port number on which the TTS socket server
# is listening for connection requests.
#
```

```
TTSControlModule.TTSServerSocketPortNumber=8012

#
# Property key that specifies the timeout value to be used for TTS server
# socket request.
#
TTSControlModule.TTSServerSocketTimeout=15000

#
# Property key that specifies the flag to indicate whether to use TTS server
# socket interface or to use TTS server COM interface for conversion of text
# to speech. Set this flag to "true" to use TTS socket interface or "false"
# to use the TTS server COM interface.
#
TTSControlModule.UseTTSServerSocketInterface=true

#
# Property key that specifies the names of the audio format configurations
# that the currently running TTSService engines are configured to convert
# text to speech. These are used to specify the audio format of the
# converted audio data.
#
# Note: Define a supported audio format property for every configuration
# name defined here. Same index should be used for both the
# TTSServerSupportedAudioConfigName property and the TTSServerSupportedAudioFormat
# property.
# Ex: Suppose if TTSControlModule.TTSServerSupportedAudioConfigName0=Config0,
# TTSControlModule.TTSServerSupportedAudioFormat0 should define the audio
# format used by Config0
#
# ***NOTE****: These properties should be updated to match the TTSEngine
# configuration everytime the TTSEngine configuration is changed to add or
# delete a audio format and the service that is serving the TTSControlModule
# shall be restarted.
#
#TTSControlModule.TTSServerSupportedAudioConfigName0=ULAWConfig
#TTSControlModule.TTSServerSupportedAudioConfigName1=

#
# Property key that specifies the audio formats being used by the configurations
# defined by the TTSControlModule.TTSServerSupportedAudioConfigName properties.
#
# The format string should be of the form "Wave: Linear 8bit 8kHz"
#
#TTSControlModule.TTSServerSupportedAudioFormat0=Wave: uLaw 8bit 8kHz
#TTSControlModule.TTSServerSupportedAudioFormat1=

#
# Oracle db driver class
#
jdbc.drivers=oracle.jdbc.driver.OracleDriver

#
# Settings for the Orbacus ORB.  Refer to the ORB documentation
# for usage.
#
#ooc.orb.trace_level=10
ooc.orb.conc_model=threaded
ooc.orb.oa.conc_model=thread_pool
ooc.orb.oa.thread_pool=10
ooc.orb.oa.port=8005
ooc.orb.service.EventChannelFactory=corbaloc::chart2-
svr5:8001/DefaultEventChannelFactory
ooc.orb.service.TradingService=corbaloc::chart2-svr5:8002/TradingService

#----- Properties of 'MessageLibraryModule' ----------------------------------
#
```

```
#
# (boolean) Property key to turn on special tests and logging in
# MessageLibraryModule.MessageLibraryModule.java.
#
#  Log Messages controlled by the prop are prefixed with special strings
#  indicating the area of the test:
#
#  prefix   test
#  ------   ------------------
#  #CCI     MessageLibraryModule.initialize() runs a test of
#           getAllClipIdentifiers and confirmClipInterest().
#           confirmClipInterest displays its results each time called by
#           AudioClipManager.
#
#  #MLT     Traces actions on stored messages.
#
#
MessageLibraryModule.DoMessageLibraryTests=false
```

Notserv.props

```
# Directory for DB files
#ooc.notification.dbdir=<INSTALL_PATH>\bin\NotifyService\db
ooc.notification.dbdir=d:/test/chartii-r1b3.10/bin/NotifyService/db

# Generate references using IP address.
# - BOA flavor is for ORB 3.X (R1B3)
# - IIOP flavor is for ORB 4.X
ooc.boa.numeric=true
ooc.iiop.numeric=true

ooc.notification.dispatch_strategy=thread_pool
ooc.notification.dispatch_threads=50

#These are not verified to work... I am just playing with them.
ooc.notification.MaxRetries=5
ooc.notification.RetryMultiplier=1.0

#location of event channel factory
ooc.service.NotificationService=corbaloc:iiop:chart2-
svr5:8010/DefaultEventChannelFactory
#
# The following should contain a list of all the trading services that are needed to
be
# linked to each other. The format in which it should be written ahould be
#
"ooc.orb.service.TradingService(x)=corbaloc::<TRADER_HOST>:<TRADER_PORT>/TradingServic
e"
# where x should be 1,2,3....
#

ooc.orb.service.TradingService1=corbaloc::CHART2-SVR5:8002/TradingService
ooc.orb.service.TradingService2=corbaloc::CHART2-SVR3:9002/TradingService

#
# Each of the Trading services listed above can be given a "real" name which will
# be used as the link name for that trading service when linked to another
# trading service
#

TradingService1=CHART2-SVR5
TradingService2=CHART2-SVR3
```

TrafficEventService.props

```
#
# This is the properties file used by the TrafficEvent Service Application.
# All properties in this file are added to the system properties.
# Vendor code like the ORB and Oracle access properties from the
# system properties and thus including their properties in this
# file allows all properties needed by the application to reside
# in the same file.
#

#
# Name of this service.
#
DefaultServiceApplication.ServiceName=TrafficEventService

#
# Refresh Trader Offers flag.  Set this flag to true when you want
# all previous trader offers made from within this app to be withdrawn
# prior to starting service modules.  Also allows service modules
# to publish objects that are de-persisted from the database.
#
DefaultServiceApplication.RefreshOffers=true


#
# DB Connection String for Oracle Database
#
DefaultServiceApplication.DBConnectString=jdbc:oracle:thin:@chart2-svr3:1526:SOC3

#
# User name to be used to connect to the database
#
DefaultServiceApplication.DBUserName=TrafficEventService

#
# Password to be used to connect to the database
#
DefaultServiceApplication.DBPassword=chart2te

#
# Maximum number of connections for DBConnectionManager to allocate.
#
DefaultServiceApplication.DBMaxConnections=10

#
# Interval for DBConnectionManager to scan list of in-use connections
# to detect threads that have exited without releasing connection.
# (milliseconds)
#
DefaultServiceApplication.DBConnectionMonitorInterval=5000

#
# Interval for OperationsLog to write queued items to the database.
# (milliseconds)
#
DefaultServiceApplication.OperationLogInterval=5000

#
# Network connection site of the service
#
DefaultServiceApplication.NetConnectionSite=chart2-svr5

#
# Name of file where trader offers are stored so that they
# can be cleaned up later using the RefreshOffers flag.
```

```
#
DefaultServiceApplication.OfferFilename=TrafficEventOffers.txt

#
# Name to be prepended to the date to name the log file
#
DefaultServiceApplication.LogFileName=TrafficEventService_

#
# Logging level. Valid values are production or debug
#
DefaultServiceApplication.LogFileLevel=production

#
# Number of days to keep old log files
#
DefaultServiceApplication.LogFileKeepDays=7

#
# Number of tries to make when resolving references for initial services. (Default 1)
#
DefaultServiceApplication.ResolveInitialReferenceTries=2

#
# Amount of time to wait for establishing a connection for requests to other services
# specified in milliseconds.
#
DefaultServiceApplication.ORBConnectTimeoutMillis=5000

#
# Amount of time to wait for requests to other services to complete
# specified in milliseconds.
#
DefaultServiceApplication.ORBRequestTimeoutMillis=30000

#
# Class names of ServiceApplicationModules to be instantiated
# and run by this service.
#
DefaultServiceApplication.ModuleClassName0=CHART2.CommLogModule.CommLogModule
DefaultServiceApplication.ModuleClassName1=CHART2.TrafficEventModule.TrafficEventModul
e

#
# This property specifies how long to wait, in minutes,
# before declaring an iterator to be disused, i.e., no longer active or needed.
#
CommLogModule.LogIteratorDisuseTimeoutMin=15

#
# This property specifies how often to check, in minutes, to see if the
# iterator should be considered as disused.
#
CommLogModule.LogIteratorDisuseCheckIntervalMin=30

#
# Interval used by TrafficEventFactory to determine how frequently to poll
# for devices that are being controlled but whose controlling op center has
# no users logged in in seconds.
#
TrafficEventModule.SharedResMonIntSecs= 300

#
# Interval used by TrafficEventFactory to determine how long a closed traffic event
# is to kept active before taking it offline in hours.
#
TrafficEventModule.OfflineThresholdHours = 12
```

```
#
# Interval used by TrafficEventFactory to determine how frequently to poll
# for responses from response plan items if a traffic event is executed
# and to send history change updates for the traffic events in seconds.
#
TrafficEventModule.TrafficEventResponseMonitorInterval = 1

#
# Interval used by TrafficEventFactory to determine how frequently to poll
# the trader for EORS permits in minutes.
#
TrafficEventModule.EORSPermitLookupInterval = 60

#
# Property key that specifies the interval at which response
# plan item status are updated in seconds.
#
TrafficEventModule.TrafficEventRPIStatusMonitorInterval=60

#
# This property specifies how long to wait, in minutes,
# before declaring an iterator to be disused, i.e., no longer active or needed.
#
TrafficEventModule.LogIteratorDisuseTimeoutMin=15

#
# This property specifies how often to check, in minutes, to see if the
# iterator should be considered as disused.
#
TrafficEventModule.LogIteratorDisuseCheckIntervalMin=15

#
# Oracle db driver class
#
jdbc.drivers=oracle.jdbc.driver.OracleDriver

#
# Settings for the Orbacus ORB.  Refer to the ORB documentation
# for usage.
#
#ooc.orb.trace.connections=10
ooc.orb.conc_model=threaded
ooc.orb.oa.conc_model=thread_pool
ooc.orb.oa.thread_pool=10
ooc.iiop.port=8006
ooc.orb.service.EventChannelFactory=corbaloc::chart2-
svr5:8001/DefaultEventChannelFactory
ooc.orb.service.TradingService=corbaloc::chart2-svr5:8002/TradingService
```

TSSService.props

```
#
# TSSService properties pre-configured to use the DEV database
# and the trader and event service on chart2-svr3.
# These properties are for use during testing from your ClearCase view.
#
#
# This is the properties file used by the TSS Service Application.
# All properties in this file are added to the system properties.
# Vendor code like the ORB and Oracle access properties from the
# system properties and thus including their properties in this
# file allows all properties needed by the application to reside
# in the same file.
#

#
# Name of this service.
#
DefaultServiceApplication.ServiceName=TSSService

#
# Refresh Trader Offers flag.  Set this flag to true when you want
# all previous trader offers made from within this app to be withdrawn
# prior to starting service modules.  Also allows service modules
# to publish objects that are de-persisted from the database.
#
DefaultServiceApplication.RefreshOffers=true


#
# DB Connection String for Oracle Database
#
DefaultServiceApplication.DBConnectString=jdbc:oracle:thin:@chart2-svr3:1526:SOC3

#
# User name to be used to connect to the database
#
DefaultServiceApplication.DBUserName=TSSService

#
# Password to be used to connect to the database
#
DefaultServiceApplication.DBPassword=chart2tss

#
# Maximum number of connections for DBConnectionManager to allocate.
#
DefaultServiceApplication.DBMaxConnections=10

#
# Interval for DBConnectionManager to scan list of in-use connections
# to detect threads that have exited without releasing connection.
# (milliseconds)
#
DefaultServiceApplication.DBConnectionMonitorInterval=5000

#
# Interval for OperationsLog to write queued items to the database.
# (milliseconds)
#
DefaultServiceApplication.OperationLogInterval=5000

#
# Network connection site of the service
#
```

```
DefaultServiceApplication.NetConnectionSite=chart2-svr5

#
# Name of file where trader offers are stored so that they
# can be cleaned up later using the RefreshOffers flag.
#
DefaultServiceApplication.OfferFilename=TSSOffersDev.txt

#
# Name to be prepended to the date to name the log file
#
DefaultServiceApplication.LogFileName=TSSServiceDev_

#
# Logging level. Valid values are production or debug
#
DefaultServiceApplication.LogFileLevel=production

#
# Number of days to keep old log files
#
DefaultServiceApplication.LogFileKeepDays=7

#
# Amount of time to wait for establishing a connection for requests to other services
# specified in milliseconds.
#
DefaultServiceApplication.ORBConnectTimeoutMillis=5000

#
# Number of tries to make when resolving references for initial services. (Default 1)
#
DefaultServiceApplication.ResolveInitialReferenceTries=2

#
# Class names of ServiceApplicationModules to be instantiated
# and run by this service.
#
DefaultServiceApplication.ModuleClassName0=CHART2.TSSManagementModule.TSSManagementMod
ule
#

#
# Full path + name of the raw data log file for TSS data.
#
TSSManagementModule.RawDataFilename=.\\RawData\\RawData_

#
# Directory where debug logs are to be placed.
#
TSSManagementModule.DebugFileDir=.\\DebugLogs

#
# Interval at which TSS data for all TSS objects in this module is to be
# collected and pushed on a CORBA event channel.
#
TSSManagementModule.StatusPushIntervalSecs=60



#
# Oracle db driver class
#
jdbc.drivers=oracle.jdbc.driver.OracleDriver

#
# Settings for the Orbacus ORB.  Refer to the ORB documentation
```

```
# for usage.
#
ooc.orb.trace.connections=10
ooc.orb.conc_model=threaded
ooc.orb.oa.conc_model=thread_pool
ooc.orb.oa.thread_pool=10
ooc.orb.oa.port=8008
ooc.orb.service.EventChannelFactory=corbaloc::chart2-
svr5:8001/DefaultEventChannelFactory
ooc.orb.service.TradingService=corbaloc::chart2-svr5:8002/TradingService
```

UMService.props

```
#
# This is the properties file used by the User Management Service Application.
# All properties in this file are added to the system properties.
# Vendor code like the ORB and Oracle access properties from the
# system properties and thus including their properties in this
# file allows all properties needed by the application to reside
# in the same file.
#

#
# Name of this service.
#
DefaultServiceApplication.ServiceName=UserManagementService

#
# Refresh Trader Offers flag.  Set this flag to true when you want
# all previous trader offers made from within this app to be withdrawn
# prior to starting service modules.  Also allows service modules
# to publish objects that are de-persisted from the database.
#
DefaultServiceApplication.RefreshOffers=true


#
# DB Connection String for Oracle Database
#
DefaultServiceApplication.DBConnectString=jdbc:oracle:thin:@chart2-svr3:1526:SOC3

#
# User name to be used to connect to the database
#
DefaultServiceApplication.DBUserName=UserManagementService

#
# Password to be used to connect to the database
#
DefaultServiceApplication.DBPassword=chart2um

#
# Maximum number of connections for DBConnectionManager to allocate.
#
DefaultServiceApplication.DBMaxConnections=10

#
# Interval for DBConnectionManager to scan list of in-use connections
# to detect threads that have exited without releasing connection.
# (milliseconds)
#
DefaultServiceApplication.DBConnectionMonitorInterval=5000

#
# Interval for OperationsLog to write queued items to the database.
# (milliseconds)
#
DefaultServiceApplication.OperationLogInterval=5000

#
# Network connection site of the service
#
DefaultServiceApplication.NetConnectionSite=chart2-svr5

#
# Name of file where trader offers are stored so that they
# can be cleaned up later using the RefreshOffers flag.
```

```
#
DefaultServiceApplication.OfferFilename=UMServiceOffers.txt

#
# Name to be prepended to the date to name the log file
#
DefaultServiceApplication.LogFileName=UMService_

#
# Logging level. Valid values are production or debug
#
DefaultServiceApplication.LogFileLevel=production

#
# Number of days to keep old log files
#
DefaultServiceApplication.LogFileKeepDays=7

#
# Number of tries to make when resolving references for initial services. (Default 1)
#
DefaultServiceApplication.ResolveInitialReferenceTries=2

#
# Amount of time to wait for establishing a connection for requests to other services
# specified in milliseconds.
#
DefaultServiceApplication.ORBConnectTimeoutMillis=5000

#
# Amount of time to wait for requests to other services to complete
# specified in milliseconds.
#
DefaultServiceApplication.ORBRequestTimeoutMillis=30000


#
# Class names of ServiceApplicationModules to be instantiated
# and run by this service.
#
DefaultServiceApplication.ModuleClassName0=CHART2.UserManagementModule.UserManagementM
odule
DefaultServiceApplication.ModuleClassName1=CHART2.ResourcesModule.ResourcesModule
#

#
# Interval (in minutes )used by the resources module to determine how frequently
# to have each operations center object cleanup invalid login sessions.
# The operations center will have to ping each login session in order
# to perform this task.  This value will default to five minutes if unspecified
# or if the specified value is not an integer.
#
ResourcesModule.LoginSessionCleanupInterval=5

# Interval (in minutes) that should pass after the last
# successful ping before a logged in user session will be considered invalid.
#
# The value of this property should be at least three times greater
# than the cleanup interval; otherwise it will be ignored and a user
# session will be disconnected if it cannot be successfully pinged for
# a period of time equal to three cleanup intervals.
#
# If this value  is not specified, it will default to 30 minutes.
#
ResourcesModule.LoginSessionDisconnectTime=30

# Timeout (in seconds) for the resource check when the last user
```

```
# logs out of an operations center.  If the system cannot
# determine whether there are shared resources controlled by
# the operations center within this time period, the user
# will be allowed to log out.  The resource watchdogs in the
# SharedResourceManager implementations will then be responsible
# for alerting the appropriate people that there are unhandled resources.
ResourcesModule.LogoutResourceCheckTimeoutSec=60

#
# Oracle db driver class
#
jdbc.drivers=oracle.jdbc.driver.OracleDriver

#
# Settings for the Orbacus ORB.  Refer to the ORB documentation
# for usage.
#
#ooc.orb.trace.connections=10
ooc.orb.conc_model=threaded
ooc.orb.oa.conc_model=thread_pool
ooc.orb.oa.thread_pool=10
ooc.orb.oa.port=8003
ooc.orb.service.EventChannelFactory=corbaloc::chart2-
svr5:8001/DefaultEventChannelFactory
ooc.orb.service.TradingService=corbaloc::chart2-svr5:8002/TradingService
```