

STATE OF COLORADO

Intelligent Transportation Systems
700 Kipling Street, Suite 2500
Lakewood, Colorado 80215
Phone (303) 512-5834
FAX (303) 239-0848



CTMS/CTIS INTEGRATION **Contract Routing No. 04 HAA 00063**

GUI Detailed Design

Version 1.8

Approved By

Frank Kinder
CDOT ITS Office

Signature: _____

Date: _____

John Williams
CDOT ITS Office

Signature: _____

Date: _____

Prepared By:



CTMS/CTIS	Version: 1.6
GUI Detailed Design	Date: 03-Dec-04

Revision History

Date	Version	Description	Author
21-Apr-04	1.0	First Draft	Trey Grubbs Sachin Saindane
26 – Apr – 04	1.1	Significant Updates	Trey Grubbs Sachin Saindane
10 – May – 04	1.2	Screen Shots, Other updates	Trey Grubbs Sachin Saindane
18 – May - 04	1.3	Added Manage User, Add User, Edit User and Sign status screens	Sachin Saindane
27 – May – 04	1.4	Added Exception Handling	Trey Grubbs
20 – Aug – 04	1.5	Updated to reflect implementation of construction one (C1)	Trey Grubbs
03 – Dec – 04	1.6	Updated to reflect implementation of construction two (C2)	Trey Grubbs Sachin Saindane
04 – Feb – 05	1.7	Updated to reflect implementation of construction three (C3)	Trey Grubbs
02-Mar-05	1.8	Updated to conform to standards: added tables.	Trey Grubbs

CTMS/CTIS	Version: 1.6
GUI Detailed Design	Date: 03-Dec-04

Table of Contents

I. EXECUTIVE SUMMARY	7
II. SUMMARY OF KEY TECHNOLOGIES.....	7
1. INTRODUCTION.....	8
1.1 PURPOSE	8
1.2 SCOPE	8
1.3 DESIGN TOOLS	8
1.4 REFERENCE.....	8
2. KEY DESIGN CONCEPTS	9
2.1 CONCEPTUAL DESIGN	9
2.2 CLIENT PROFILE.....	10
2.3 DEPLOYMENT CONSIDERATIONS.....	11
2.4 BUILDING BLOCKS.....	11
2.5 LOGGING.....	11
2.6 SPELL CHECKER.....	11
2.7 PACKAGES	11
3. DETAILED DESIGN.....	15
3.1 CORE FRAMEWORK.....	15
3.1.1 <i>Core Framework Class Diagram</i>	15
3.1.2 <i>Login Sequence Diagram</i>	17
3.1.3 <i>Logout Sequence Diagram</i>	19
3.1.4 <i>Asynchronous Instruction Framework</i>	19
3.2 PROXIES AND DELEGATES.....	21
3.2.1 <i>Delegate Package Class Diagram</i>	22
3.2.2 <i>Delegate Initialization</i>	23
3.3 UI ACTION FACTORY	23
3.3.1 <i>UI Action Factory Class Diagram</i>	24
3.3.2 <i>An Example Usage of the UI Action Factory</i>	26
3.4 MESSAGE QUEUE FRAMEWORK	27
3.4.1 <i>Message Queue Framework Class Diagram</i>	27
3.4.2 <i>Benefits of Message Queue Framework</i>	29
3.5 MAP NAVIGATOR.....	29
3.5.1 <i>Map Navigator Class Diagram</i>	29
3.5.2 <i>Map Navigator Sequence Diagram</i>	31
3.5.3 <i>Map Class Diagram</i>	31
3.5.4 <i>Map Sequence Diagram</i>	33
3.5.5 <i>Map Selection Sequence</i>	34
3.5.6 <i>Integration of GIS Data</i>	35
3.6 STATUS CONSOLE	35
3.6.1 <i>Status Console Class Diagram</i>	35
3.6.2 <i>Status Console Creation Sequence Diagram</i>	37
3.7 STATUS CONSOLE WINDOWS – ALARMS, LOGGING, AND THE INSTRUCTION QUEUE.....	38
3.7.1 <i>Alarms</i>	39
3.7.2 <i>Logging</i>	40
3.7.3 <i>Instruction Queue</i>	40
3.8 DMS SERVICES.....	40
3.8.1 <i>DMS Services Class Diagram</i>	40

CTMS/CTIS	Version: 1.6
GUI Detailed Design	Date: 03-Dec-04

3.8.2	Activate Message Sequence Diagram.....	40
3.8.3	WYSIWYG Message Editor Class Diagram	40
3.8.4	WYSIWYG Message Editor Sequence Diagram	40
3.9	DMS MANAGEMENT.....	40
3.9.1	Add DMS Implementation of Wizard Framework	40
3.9.2	Configure DMS Class Diagram	40
3.10	USER MANAGEMENT.....	40
3.10.1	User Management Class Diagram	40
3.10.2	User Management Sequence Diagram.....	40
3.10.3	Add User Sequence Diagram	40
3.11	COMMUNICATIONS MANAGEMENT	40
3.11.1	Communications Pool	40
3.11.2	Communications Port.....	40
3.12	CTMS HELP SYSTEM.....	40
3.12.1	Help System Class Diagrams	40
3.12.2	Help System Sequence Diagrams	40
3.13	UI UTILITIES	40
3.13.1	Generic UI Utilities Class Diagram.....	40
3.13.2	UI Component Factory Class Diagram.....	40
APPENDICES	40
APPENDIX A: DETAILED DESCRIPTION OF PRESENTATION		40
Login and Splash Screen.....		40
Map Navigator		40
Manage Users		40
Edit Users.....		40
Add Users.....		40
View DMS.....		40
Activate DMS Message.....		40
View Sign Status.....		40
View Errors.....		40
Adjust Brightness		40
Clear Sign		40
Generate Fax Log Report.....		40
Status Console.....		40
Alarm Window.....		40
Logging Window		40
Instruction Queue Window.....		40
Manage Communications Pool.....		40
Add New Pool.....		40
Edit Pool		40
Manage Communications Ports		40
Add New Port.....		40
Edit Port.....		40
Help Navigator.....		40
APPENDIX B: THE MENU XML DESCRIPTOR FILE		40
APPENDIX C: THE UI ACTION XML DESCRIPTOR FILE.....		40
APPENDIX D: THE MAP CONFIGURATION XML FILE.....		40
APPENDIX E: HELP SYSTEM EXAMPLE CODE.....		40
APPENDIX F: CLIENT DEPLOYMENT GUIDE.....		40
System Configuration.....		40
Installation Procedure.....		40
Update Procedure.....		40

Table of Figures

Figure 1: UI Conceptual Regions	9
Figure 2: UI Conceptual Diagram	10
Figure 3: Package Diagram	12
Figure 4: Package Relationships.....	14
Figure 5: Core Framework Class Diagram	16
Figure 6: Login Sequence.....	18
Figure 7: Logout Sequence.....	19
Figure 8: Class Diagram for Asynchronous Framework	20
Figure 9: Sequence Diagram for Asynchronous Framework.....	21
Figure 10: Delegate Package Class Diagram.....	22
Figure 11: Delegate Initialization	23
Figure 12: UI Action Factory Class Diagram.....	25
Figure 13: Usage of UI Action Factory	27
Figure 14: Message Queue Class Diagram.....	28
Figure 15: Map Navigator Class Diagram.....	30
Figure 16: Map Navigator Sequence Diagram	31
Figure 17: Map Class Diagram.....	32
Figure 18: Map Sequence Diagram	34
Figure 19: Selection Framework Sequence Diagram	35
Figure 20: Status Console Class Diagram	36
Figure 21: Status Console Sequence Diagram.....	38
Figure 22: Alarm Messaging Class Diagram.....	39
Figure 23: Alarm Sequence Diagram	40
Figure 24: Instruction Queue Class Diagram	40
Figure 25: Instruction Queue Sequence Diagram.....	40
Figure 26: DMS Services Class Diagram	40
Figure 27: Activate Message Sequence Diagram	40
Figure 28: Message Editor Class Diagram	40
Figure 29: Message Editor Sequence Diagram.....	40
Figure 30: Add DMS Class Diagram.....	40
Figure 31: Configure DMS Class Diagram	40
Figure 32: User Management Class Diagram.....	40
Figure 33: User Management Sequence Diagram	40
Figure 34: Add User Sequence Diagram	40
Figure 35: Comm Pool Class Diagram	40
Figure 36: Comm Port Class Diagram.....	40
Figure 37: Help System class diagram	40
Figure 38: IHelpConnector.....	40
Figure 39: HelpUtilities	40
Figure 40: Sequence Diagram for Help Action	40
Figure 41: Sequence Diagram for Context Sensitive Help	40
Figure 42: Generic UI Utilities Class Diagram	40
Figure 43: UI Component Factory Class Diagram	40
Figure 44: Splash Screen	40
Figure 45: Login Screen	40
Figure 46: Map Frame	40
Figure 47: Manage Users Screen.....	40
Figure 48: Edit Users Screen	40
Figure 49: Add New User Screen.....	40
Figure 50: View DMS Dialog	40
Figure 51: Activate DMS Message Dialog.....	40

CTMS/CTIS	Version: 1.6
GUI Detailed Design	Date: 03-Dec-04

Figure 52: Sign Status Dialog.....	40
Figure 53: View Details Window	40
Figure 54: Adjust Brightness.....	40
Figure 55: Clear Sign.....	40
Figure 56: Generate Report Fax	40
Figure 57: Status Console.....	40
Figure 58: Alarms Window	40
Figure 59: Logging Window	40
Figure 60: Instruction Queue Frame.....	40
Figure 61: Add DMS Wizard - Introduction	40
Figure 62: Add DMS Wizard - Category Page.....	40
Figure 63: Add DMS Wizard - Device Info Page.....	40
Figure 64: Add DMS Wizard - Communication Page for Dialup	40
Figure 65: Add DMS Wizard - Communication Page for Fiber.....	40
Figure 66: Add DMS Wizard - Communication Page for IP.....	40
Figure 67: Add DMS Wizard - Summary Page.....	40
Figure 68: Configure DMS - General Page	40
Figure 69: Configure DMS - Details Page.....	40
Figure 70: Edit Common Name.....	40
Figure 71: Configure DMS Dimensions Page	40
Figure 72: Configure DMS - Equipment Page	40
Figure 73: Configure DMS - Communication Page	40
Figure 74: Configure DMS Communication Page	40
Figure 75: Configure DMS - One Wire Page	40
Figure 76: Configure DMS - Multi Defaults Page.....	40
Figure 77: Manage Communications Pool	40
Figure 78: Add Pool	40
Figure 79: Edit Pool.....	40
Figure 80: Manage Communications Port.....	40
Figure 81: Add New Port.....	40
Figure 82: Edit Port	40
Figure 83: Help Navigator.....	40
Figure 84: Monitor Settings.....	40
Figure 85: Graphics Device Settings	40
Table 1: CTMS Client Package Description.....	13
Table 2: Menu XML Description	40
Table 3: UI Action XML Description	40
Table 4: Map Configuration XML	40
Table 5: The Help Action	40
Table 6: Example of Adding a Modal Help Topic	40
Table 7: Using The Help Dialog.....	40

Note: The screen shots and wire frames in the document are intended to communicate a possible appearance of the User Interface. The actual implementation may vary in size and appearance. Also, the document discusses the design and UI related to system Alarms. These Alarm features were delayed until CTMS Iteration 2. The discussions, prototypes, and framework were developed as part of Iteration 1, therefore, they are present in this document.

CTMS/CTIS	Version: 1.6
GUI Detailed Design	Date: 03-Dec-04

i. Executive Summary

This document describes the architecture and various technologies that form the CTMS/CTIS client application. A summary of the GUI's key development concepts, detailed architectural design, and set of screen mock-ups are presented. The first iteration of the project will include the foundation for GUI and full support for sign devices. The proposed architecture for the client application includes full support for continuous enhancement, maximum flexibility, and building-block components that will ensure a highly adaptable and robust system.

ii. Summary of Key Technologies

The development CTMS UI System will employ proven technologies, software methodologies, and best practices. Specifically, the Rational Unified Process will be followed in order to facilitate a thorough analysis and design of the complex system. The core technology will be built using XML, EJB, Object-Oriented concepts, and several design patterns including Factory, Delegate, and Model-View-Controller.

CTMS/CTIS	Version: 1.6
GUI Detailed Design	Date: 03-Dec-04

1. Introduction

This document describes the detailed design of the Graphical User Interface (GUI) for the Colorado Traffic Management System (CTMS).

1.1 Purpose

The purpose of this document is to detail the architecture and foundation concepts behind the CTMS GUI.

1.2 Scope

The scope of this document is the GUI of the CTMS/CTIS project. All other aspects of the system are covered in other architectural documents. Please refer to the references section for a complete list.

1.3 Design Tools

Several technologies were leveraged to reduce system design and development time as well as reduce testing costs. These tools include:

- Infragistics JSuite v7.0 – the Infragistics GUI toolkit provides an extensive set of enhanced java objects that offer complex, well tested behaviors for minimal cost.
- ESRI MapObjects v2.0 – the ESRI MapObjects toolkit provides industry leading tools and capabilities for producing GIS data solutions.
- Log4J – the Log4j framework offers a widely accepted logging capability.
- JDOM – the JDOM XML tools provide an efficient and effective solution for integrating XML into Java applications.
- Ant – the most impressive and powerful build and deployment solution available.
- Quartz – the scheduling framework that is widely accepted.
- OHJ – Oracle Help for Java API's for Java based Help System.

1.4 Reference

The following references were either used in the creation of this document or may be used to supplement the detailed descriptions of the CTMS UI.

- The server architecture document
- ESRI MapObjects v 2.0
- JSuite documentation
- GoF book
- J2EE patterns book
- Use Case documents

CTMS/CTIS	Version: 1.6
GUI Detailed Design	Date: 03-Dec-04

2. Key Design Concepts

The CTMS graphical user interface (GUI) is driven by several key concepts that greatly influence its design and development. These concepts include the general structure, GIS map, the use of client profiles, the method of deployment, and the creation of generic technologies, externalized label and messages, and the use of XML for action and menu construction.

2.1 Conceptual Design

The CTMS UI can be divided into four, mostly isolated regions: Framework, Delegates, Maps, and Tables. The Framework region will handle all the overall responsibilities, such as handling the login, initiating the JMS and EJB connections, creating the UI containers, and managing the client profile. The Delegate region will provide details for establishing the JMS specific messaging and EJB manipulations by providing generic services for use by the client framework. The Maps region will handle everything that is ESRI MapObjects specific. It will also handle the UI details for using map objects and provide a panel for use by a container in the client framework. The Tables region will handle all the guts of the tables that are present in the system. The following diagram details the four regions.

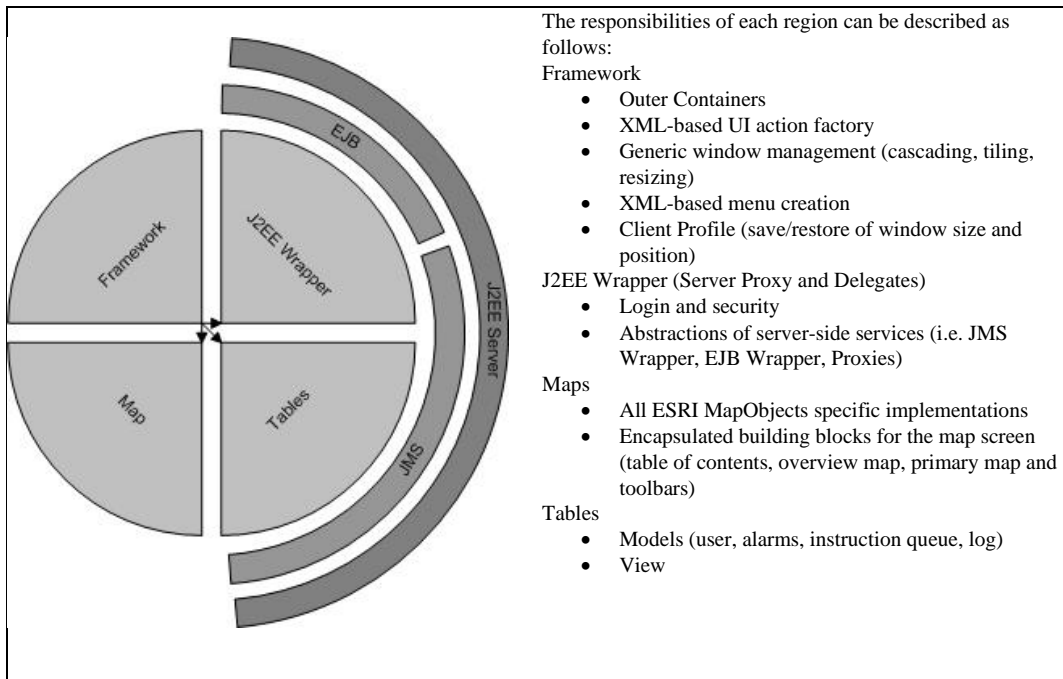


Figure 1: UI Conceptual Regions

As an extension to the high-level table that defines the system regions, the client architecture can be further conceptualized into a set of high-level components. Although these objects do not necessarily map directly into software constructs, they assist in breaking down and isolating the behavioral responsibilities. The following figure demonstrates this analysis.

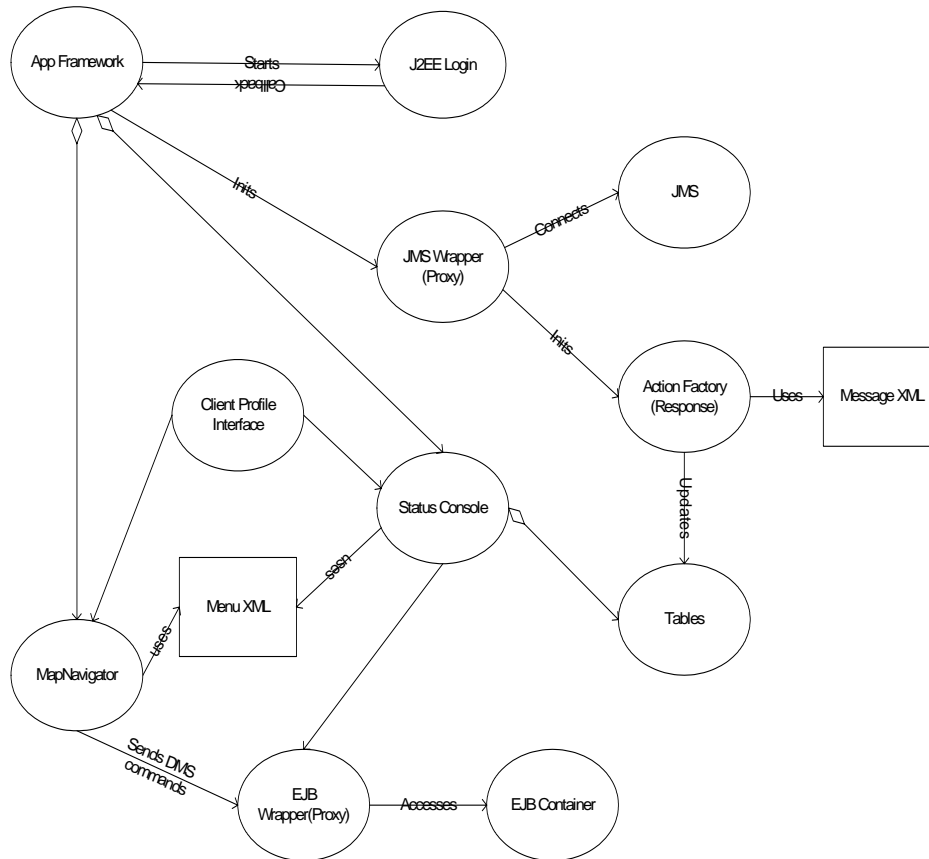


Figure 2: UI Conceptual Diagram

As demonstrated in the diagram, the application framework deals directly with the login and the initialization of the J2EE abstractions. It is also the owning link for the two UI screens, the Map Navigator and the Status Console. These screens will be instantiated and then configured based on the users last know state, or client/session profile. The Map Navigator will create its menu based on an XML file. It will also use a server proxy to issue requests for services dealing with the DMS'. The Status Console will orchestrate the presentation of status tables that are stimulated by updates received via the JMS messaging abstraction. Upon receipt of a JMS message, an action factory will initiate the appropriate action to a status table based on the incoming message type.

2.2 Client Profile

The CTMS UI will store the state of the application, or profile, on a per user bases when the user logs out. When the user logs back in to the CTMS client application on the same machine, the size and position of the windows will be restored based on the users client profile. The client profile will consist of the following items:

- Map Navigator
 - Windows size and position
 - The visibility selections (view table of contents, view overview map, view status console, etc...)

CTMS/CTIS	Version: 1.6
GUI Detailed Design	Date: 03-Dec-04

- The location of the divider between the contents/overview map pane and the main map pane.
- The location of the divider between the contents and overview map panes.
- Status Console
 - The visibility selections for the instruction, logging, and alarm windows
 - The frames window size and position
 - The size and position of the internal frames within the Status Console

The UI will present a checkable menu-item for turning the feature on and off. Furthermore, the state of the status console may be easily restorable to defaults by a checkable menu item.

2.3 Deployment Considerations

The CTMS UI will not require a large configuration during client installs. The client installation will have the following directory structure:

- CTMS/client/ - root for the application
 - bin/ - batch file(s) for launching the application
 - resources/ - the configuration files used by the system
 - lib/ - the collection of jar files used by the client application
 - classes/ - the jar files of the client application
 - jre/ - the official jre for the version of the CTMS client

The application will not require the installation of any additional software on the client machine. The Swing implementations within a JRE tend to change between even the minor revisions. In an effort to eliminate problems, the JRE will be included as part of the CTMS client. The resource directory will contain a text file that can be edited to include default server connection information, if desired. The installation will not require the addition or modification of any system registry settings. The configuration-XML (menu, UI actions, and JMS response actions) and image (window icons and splash screen) files will be jar'd into the client jar.

2.4 Building Blocks

The CTMS UI will be built with abstraction and encapsulation in the forefront. In an effort to isolate the client UI from potential changes in future java SDK releases, the majority of java UI components will be encapsulated into a UI component factory. This factory will allow for the objects to be created and configured in a controlled and expected way. If a future SDK version, modifies a default attribute of a component, the necessary configuration will occur in a single place. Additionally, a mature and well-designed set of UI components will be used that should improve quality tremendously. Also, the menu, JMS response, and user interaction construction will be handled via an XML (plain text) abstraction. This will greatly ease the construction of new menu items and user interactions in future releases.

2.5 Logging

An important aspect of the CTMS client application is that all requests must be logged. All commands dealing logging in/out, managing users and manipulating DMS' will be logged to a log file located on the server using Log4j. Additionally, all processed JMS messages, will be logged to Log4J.

2.6 Spell Checker

The CTMS Message Editor requires the use of spell checker. This will be performed using a third party tool. The identified tool will be wrapped in a Servlet architecture and deployed within the CTMS JBoss Server. This approach offers the most flexibility while ensuring security and limiting the burden on the CTMS Server

2.7 Packages

The software design is broken into several packages. Each package holds java classes that relate to the package name. Packaging helps separate identifiable components to enable reuse. The diagram below identifies all the packages used for the client design.

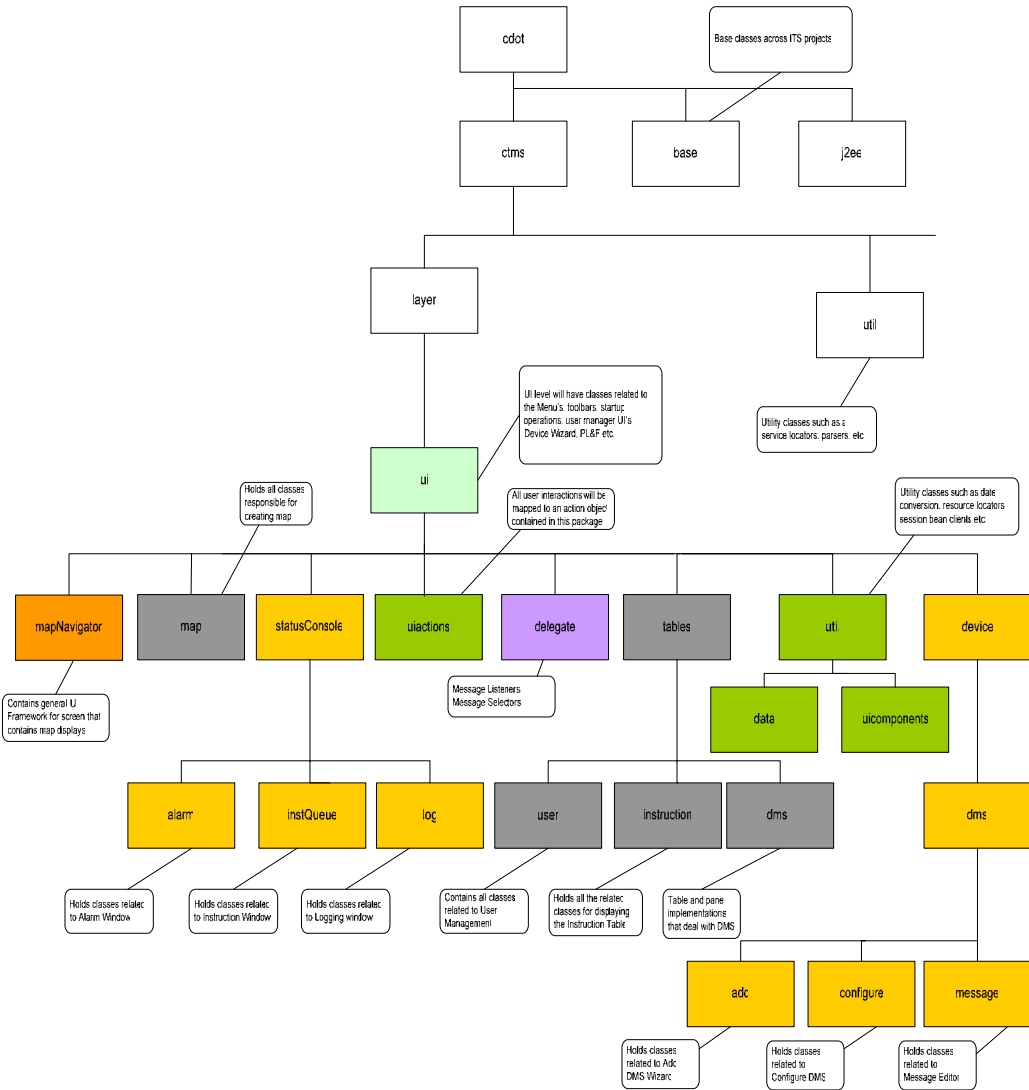


Figure 3: Package Diagram

Package may be read in the following fashion: Cdot.ctms.layer.ui.tables.log. The various packages have been described in the table below:

Package Names	Description
---------------	-------------

CTMS/CTIS	Version: 1.6
GUI Detailed Design	Date: 03-Dec-04

cdot.ctms.util	This package contains utility classes used by the GUI and servers
cdot.ctms.layer.ui	This is the top level package that holds main classes responsible for the loading of UI properties, startup operations and server connections
cdot.ctms.layer.ui.mapnavigator	This package contains UI map framework classes for screen that displays the map
cdot.ctms.layer.ui.map	This package holds map related classes such as layers, overviewmaps, toc, rubberband actions, other toolbar actions
cdot.ctms.layer.ui.statusconsole	This package holds containers and framework classes related to screen 2. Screen 2 displays alarms, instructions and system logs
cdot.ctms.layer.ui.statusconsole.alarm	This package holds classes related to the window displaying system alarms.
cdot.ctms.layer.ui.statusconsole.instQueue	This package holds classes related to the window displaying system queues
cdot.ctms.layer.ui.statusconsole.log	This package holds classes related to the Logging window
cdot.ctms.layer.ui.uiactions	All user interactions will be mapped to an action object contained in this package
cdot.ctms.layer.ui.delegate	This package will hold jms related classes such as message listeners, message selectors etc
cdot.ctms.layer.ui.tables	This package serves as the top level package for all tables related to status console
cdot.ctms.layer.ui.tables.user	Contains table and panel implementations that deal with user management
cdot.ctms.layer.ui.tables.dms	Holds classes related to table and panel implementations that deal with DMS
cdot.ctms.layer.ui.tables.instruction	This package holds all classes pertaining to display of instruction queue table, instruction queue listeners, instruction queue table model
cdot.ctms.layer.ui.util	This package holds utility classes used in the client project, ex: resource locators, session bean clients, date conversions etc
cdot.ctms.layer.ui.util.data	Holds hashtable implementations used for Add/ Configure DMS
cdot.ctms.layer.ui.util.uicomponents	Holds all ui components used throughout the product.
cdot.ctms.layer.ui.device	This is a high level package to hold device related classes.
cdot.ctms.layer.ui.device.dms	Holds all UI classes related to DMS screens - Poll DMS, Activate DMS message, Configure DMS etc
cdot.ctms.layer.ui.device.dms.add	Holds classes related to the Add DMS wizard
cdot.ctms.layer.ui.device.dms.configure	Holds classes related to Configure DMS wizard
cdot.ctms.layer.ui.device.dms.message	Holds classes related to Message Editor.

Table 1: CTMS Client Package Description

The ui package holds the main classes required to startup the client GUI and load all application properties. Framework classes for map related screen can be found within the mapnavigator class, while framework classes for the status console screen is within the statusconsole package. Tables related to each screen are under the 'tables' package. Table is the top level package for user, dms, and instruction package. Class related to a particular table is isolated and contained within a separate

CTMS/CTIS	Version: 1.6
GUI Detailed Design	Date: 03-Dec-04

package. For example, tables related to user administration are contained with the tables.user package. Most table related packages do not depend upon another package at the same level. This helps developer maintain distinction between tables used throughout the client GUI. The util package holds all utility classes used by most other packages. Classes within util are self contained and have no references to classes outside of the util package. The following diagram shows the interaction/interdependency of the packages. The delegate package is the only package that is responsible for dealing with the EJB and JMS connections. The cdot.ctms.layer.ui.statusConsole package is the top level package for alarm window, instruction window and logging window. The package device.dms.add holds all classes related to the generation/ population and validation of the Add DMS Wizard. The package device.dms.configure holds all classes related to the generation/population of Configure DMS. The package device.dms.message holds all classes related to the Message Editor.

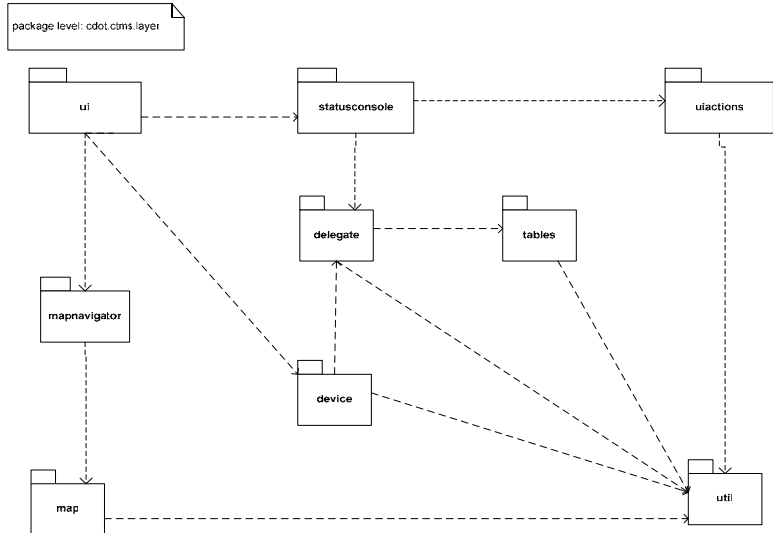


Figure 4: Package Relationships

Each package will handle the responsibilities of exception handling. These exceptions will often result in a user notification that will be handled in the package where the parent dialog/frame resides. These exceptions will not ever be forwarded to another package.

CTMS/CTIS	Version: 1.6
GUI Detailed Design	Date: 03-Dec-04

3. Detailed Design

The CTMS client UI system has been thoroughly analyzed and designed using a validated software analysis paradigm, the Rational Unified Process (RUP). The artifacts of RUP consist of a collection of diagrams that demonstrate the system components, their responsibilities and behaviors, and the interrelationships given certain events. The CTMS UI will be described in nine sections including Core Framework, UI Action Factory, Map Navigator, Status Console, Delegate and Proxies, DMS Services, User Management and Utilities.

3.1 Core Framework

The entire UI system uses a collection of classes that define the core framework of the application. This framework is responsible for providing a generic layer of behavior and constructs that facilitate the addition of more specialized constructs of the system. All the visual UI screens use components of the core framework in the client system. Specifically, the core framework will provide the loading of system properties, the coordination of the login, the base containers for the two the primary screens (Map Navigator and Status Console), the fundamental behaviors of these two screens (resizing, client profile).

3.1.1 Core Framework Class Diagram

This section will present the structure for the core framework of the UI system. It is primarily located in the top-level UI package, but it will orchestrate the construction and initialization of objects that reside in sub-packages. The sub-packages will use the generic constructs in the parent UI package that will provide the foundation for saving/restoring the ClientProfile and window management. The core framework consists of the CTMSApp, CTMSLoginHandler, J2EEClientWrapper, ApplicationProperties, AbstractCTMSFrame, JMS Response ActionFactory, and the CTMSSplashThread.

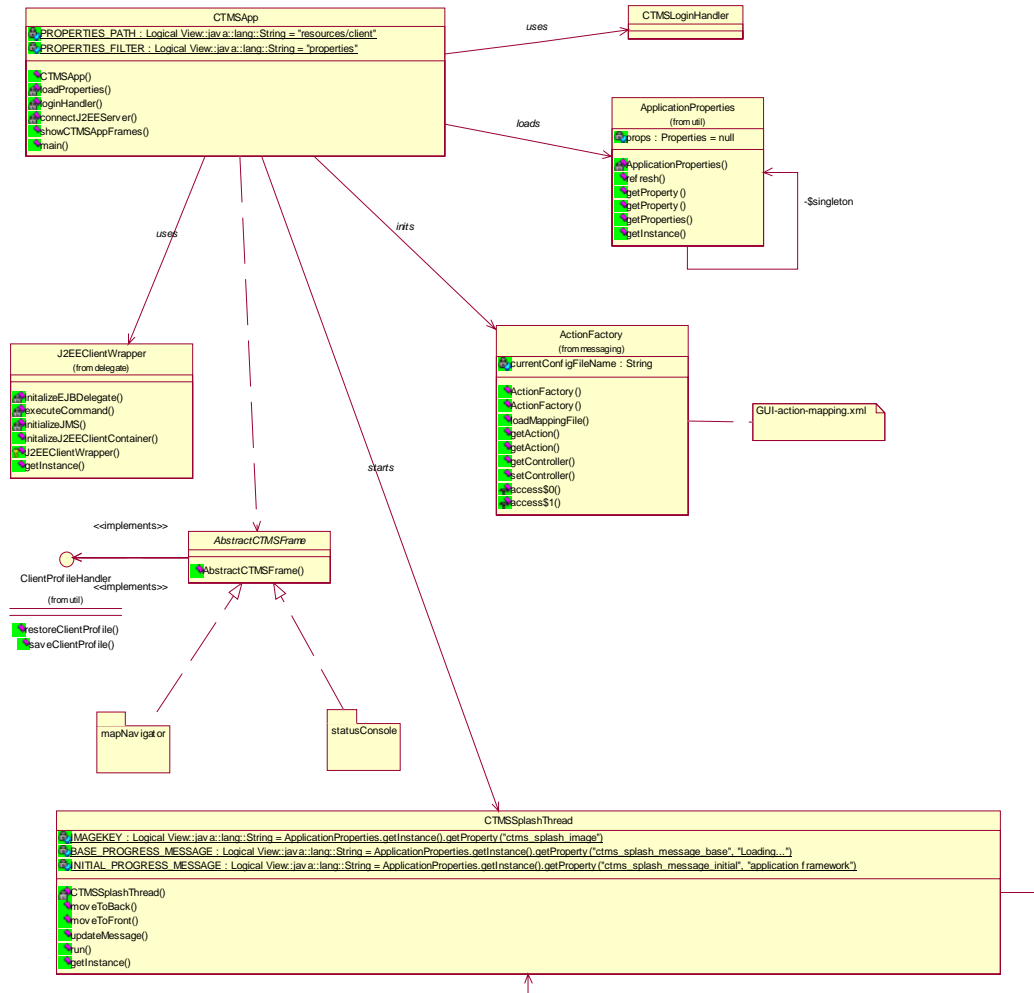


Figure 5: Core Framework Class Diagram

3.1.1.1 CTMSApp

The CTMSApp defines the primary class of the UI system. It is the users entry point in the client application. This class is responsible for loading the system properties, managing the client login, initializing the J2EE server connection to JMS and the EJB container, instantiating the action factory, and launching the main UI screens. This class will also manage the system splash screen that will show during the client startup.

CTMS/CTIS	Version: 1.6
GUI Detailed Design	Date: 03-Dec-04

3.1.1.2 CTMSLoginHandler

The login handler will coordinate the login to the J2EE server and manage the security settings. If successful, the CTMSApp will proceed with the system startup.

3.1.1.3 J2EEClientWrapper

The J2EEClientWrapper forms a client-side abstraction of the J2EE server details. The J2EE services are captured into an object that follows the Proxy pattern. The J2EEClientWrapper will further abstract and will work closely with the server proxies to ensure proper connection of the JMS generators and listeners and EJB initialization.

3.1.1.4 ApplicationProperties

The system properties include all text that is present to the user, all system configuration settings, and additional resource references. During the development of a user interface, it is very important and common practice to capture ALL text and images into a collection of resource files. The primary benefit is the significant ease in modifying the displayed items. For instance, the product could be almost totally converted to Hungarian just by swapping out the client resources. The properties object, ApplicationProperties, will use the GOF Singleton pattern. As a result, any object may easily access the collection of properties in the client system. Also, by ensuring a single set of properties, we can ensure that all objects are referring to the same property values. The loading of the properties will be the first task during the client startup. The properties will not be periodically reloaded or saved.

3.1.1.5 AbstractCTMSFrame

The abstract frame will handle common behaviors of the two UI screens, Map Navigator and Status Console. The details of each screen will be contained in the MapNavigator and StatusConsole packages. The common behaviors will be the handling of some of the client profile behavior and other windowing issues.

3.1.1.6 JMS Response ActionFactory

The CTMS UI uses the GOF Factory pattern for two different action factories. One action factory is responsible for creating and initializing the actions that occur upon receipt of a JMS message. This action factory is represented on the preceding class diagram. The second action factory is responsible for modeling the actions taken by a user, such as printing the map or exiting the system. The user inter-action factory, will be described in later section. Both action factories are fully defined by an XML file.

3.1.1.7 CTMSSplashThread

The CTMS client application will use a splash screen. This screen will display a product identifier and will also communicate the status of the client initialization. The CTMSSplashThread object will be responsible for the splash screen. The object is a Singleton, so other objects can obtain and update their status to it as it is being displayed. The thread will be started as soon as the application is launched and will be stopped as soon as the application is loaded.

3.1.2 Login Sequence Diagram

This section will describe the sequence of events that occur when the user starts the CTMS client. The entry point for the client is the CTMSApp object. The first step is to immediately load the application properties by instantiating the ApplicationProperties object with a call to getInstance(). All subsequent steps can only be performed after this has completed. After the ApplicationProperties object is loaded, it is assumed that the properties will not change throughout the client session. The next step is to initiate the users session with the server via a login handler. The handler will be instantiated and the login procedure initiated behind the scenes with the first call to the the server Proxy that is accessed via the J2EEClientWrapper. The login handler will perform all the functions to ensure a secure and valid login to the CTMS system. This includes the handling of the presentation of the login dialog, the messages resulting from a failed login, etc... If the login is unsuccessful, the client application will not continue with the login sequence. It will exit. Once a user has successfully logged into the system, the client application will obtain the users credentials from the handler. The next steps in the sequence will handle the initialization of the client-side EJB and JMS connections with the J2EE server. Based on the users login privileges, the user will have access to all or a subset of actions available with

on the UI. The J2EE Wrapper will provide a security manager that will detail which actions should be disabled based on the users role. The initialization events for the J2EE server may be performed in a separate thread in order to start loading the map components as soon as possible. However, this initialization must be complete prior to instantiating the elements of the StatusConsole. The next step is the loading of the client profile. It will be read from the client machine from .\resource\profile\username.cp. This profile will then be used to restore the last state of the map navigator and status console. Next, the MapNavigator will be started in its own thread in order to ensure efficient loading. The MapNavigator object is only dependent of the ApplicationProperties being initialized. The separate threads will only be used if performance is poor enough to justify it.

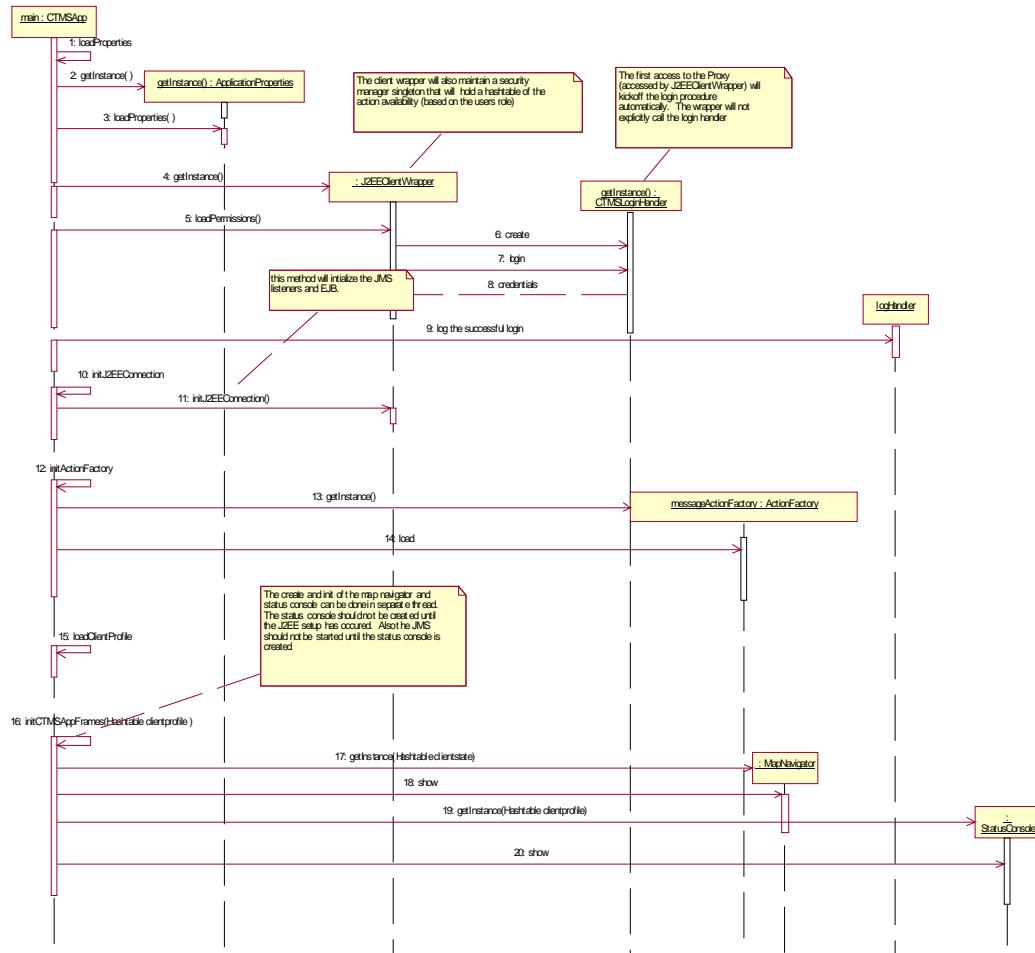


Figure 6: Login Sequence

3.1.3 Logout Sequence Diagram

This section will detail the sequence of events that occur when a user logs out of the system. The exit event will always be routed to the exitUIAction. This action will obtain the frameworks base class, CTMSApp, and ensure that the client profile is saved. The client profile will be saved in the resources directory (i.e. .resources/profile/username.cp). The client profile will then be used when the user logs back into the system. The CTMSApp.saveClientProfile() will delegate the save duties to its aggregate components. The logout action will be logged by the framework. The last step is to close the server connections.

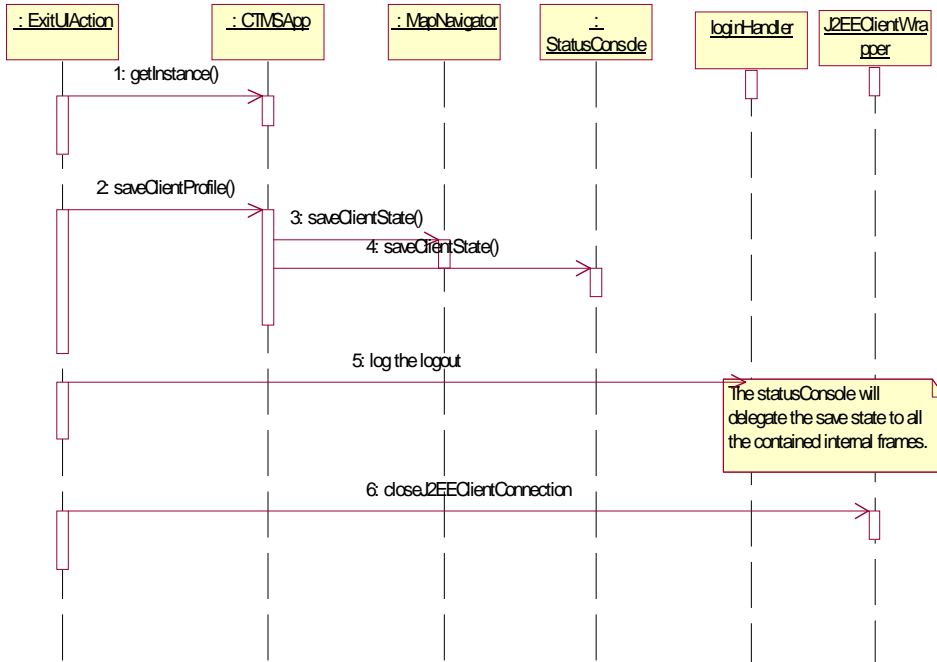


Figure 7: Logout Sequence

3.1.4 Asynchronous Instruction Framework

The UI offers several actions (accessible via menu-items, buttons, and key strokes) that require communication with devices. The actions communicate with the delegate layer of the server which in-turn generates one or more instructions that are handled asynchronously by the COMM Server. The COMM Server handles communications between our server and the devices. This poses an interesting dilemma for the CTMS client where some user initiated actions are expected to be synchronous. For instance, when you hit a button you expect immediate results. As a result of the server side asynchronous philosophy, we must somehow acknowledge the users request, proceed in some manner (don't hang the UI), and display the instruction response when it is ready. To meet these responsibilities, the client system will provide a reusable framework for generically handling these situations. The general approach will be:

- Notify the user that the instruction request (Poll Sign, Read from Sign, Perform Pixel Test) has been received and that it may take some time to obtain the results.
- Put the UI in a "waiting" state. The definition of "waiting" state differs according to the instruction/action, but it NEVER means blocking. The following paragraph will define the details of this state.
- Listen to the Notification Topic for completion of the instruction.

3.1.4.2 Sequence Diagram for Asynchronous Instruction Framework

The following sequence diagram describes how the framework will be used. A user initiates an action which causes the common component, AsyncSignInstructionOptionPane, to be instantiated. This component will add the caller to the set of listeners, subscribe to the notification topic, and notify the user that the request has been received. The UI will go into the pending state as described earlier. Once the notification is received, the listener framework is used to notify the appropriate dialog that the data has been received. The user will be prompted for display of the data and the dialog will be repopulated.

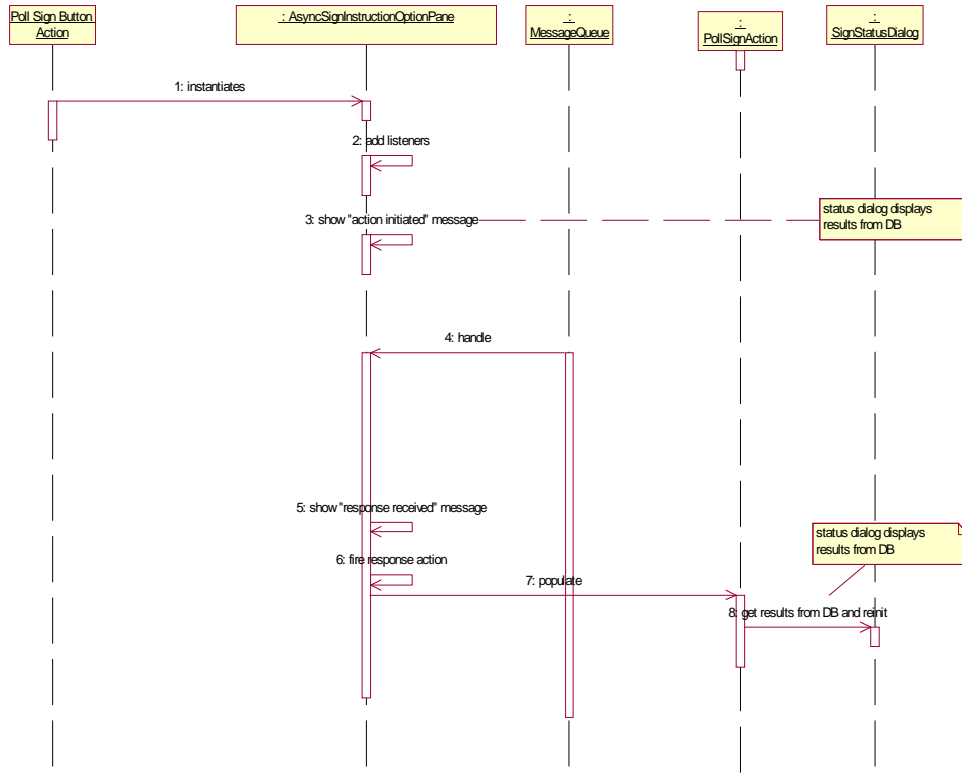


Figure 9: Sequence Diagram for Asynchronous Framework

3.2 Proxies and Delegates

The business services provided by the server are encapsulated into a class that serves as a proxy. This object, defined as RemoteControllerProxy, will serve to hide some of the implementation details that are specific to the J2EE container in use by the CTMS system. This proxy will assist in handling the requests for the business delegates of the server. As an additional layer, the client will use an additional proxy itself, called J2EEClientWrapper. This object will provide a single access point for all things that are dependent on the services provided by the server including the EJB and JMS accesses. The use of this proxy will keep business services out of the presentation layer, isolate the business service details into a single access point, reduce coupling between the presentation layer and the services, add will facilitate a centralized control point for implementing client-side caching mechanisms (if needed.) The additional layer will further isolate the client system from the details of the server communication that are encapsulated by the

CTMS/CTIS	Version: 1.6
GUI Detailed Design	Date: 03-Dec-04

RemoteControllerProxy. This section will describe the static structure of the delegate package and will describe how the listeners are created. The details involving what happens when a JMS message is received is described in a later section, called Status Console Windows.

3.2.1 Delegate Package Class Diagram

This section will detail the design for the delegate package of the CTMS client. The delegate package will hold the class responsible for initializing server connections and coordinating with the remote server proxy. Additionally, the package will contain the listeners, or JMS message handlers.

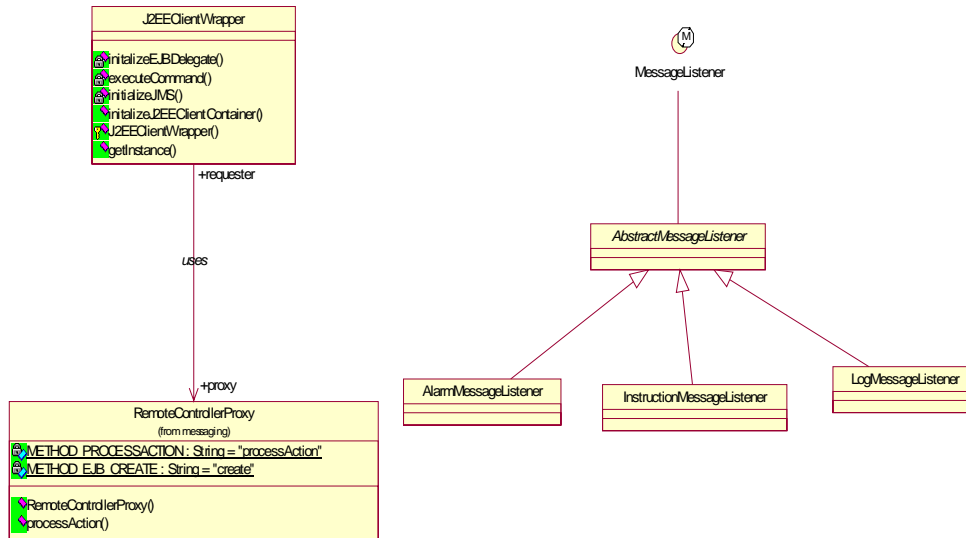


Figure 10: Delegate Package Class Diagram

3.2.1.1 J2EEClientWrapper

The J2EEClientWrapper will serve to encapsulate all the responsibilities surrounding the initialization and interaction with the J2EE server. It will extensively use the RemoteControllerProxy to perform commands dealing with the EJBs. Specifically, the J2EEClientWrapper will initialize the J2EE client connection to the container. This will include initializing the EJB delegates, performing the necessary naming lookups, and attaching the JMS listeners. This object serves as proxy to the RemoteControllerProxy. The entire client system will refer to this object, not the RemoteControllerProxy.

3.2.1.2 RemoteControllerProxy

The proxy object will assist in encapsulating the knowledge about and responsibility of the server implementation. This object will contain the details about the EJBs and will issue any commands to the server.

3.2.1.3 AbstractMessageListener

This object is the generic parent to the specialized JMS listeners that listen for JMS alarm, instruction, and log messages.

3.2.2 Delegate Initialization

This sequence diagram simply shows that the J2EEClientWrapper will create and add the specialized message listeners.

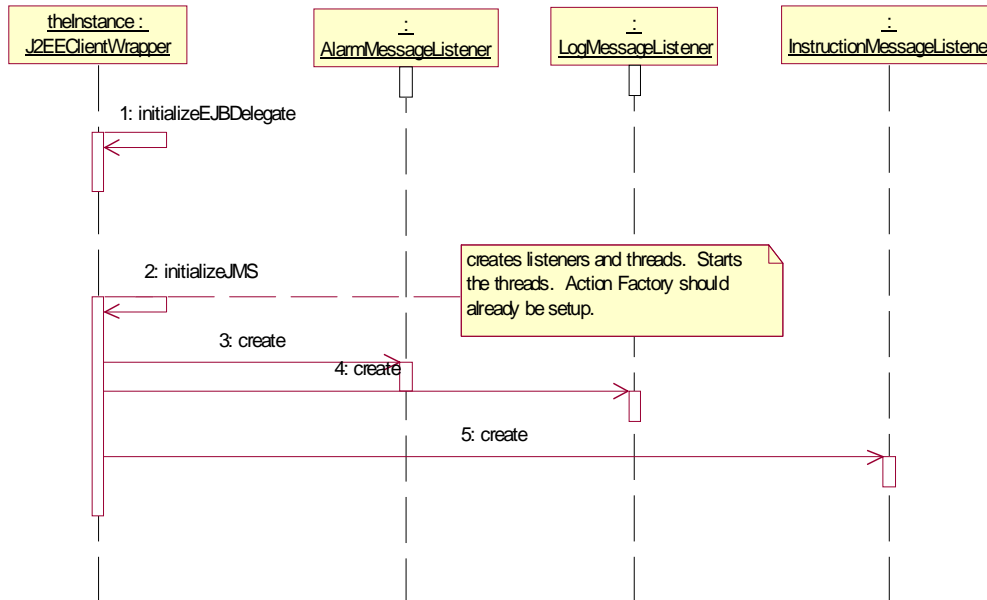


Figure 11: Delegate Initialization

3.3 UI Action Factory

A previous section described the action factory used to coordinate the response to JMS messages. This section will detail the UI Action Factory. This factory consists of a Singleton object that may be used to obtain all the user actions on the system. These actions include:

- Map Navigator General Actions
 - Exiting
 - Printing the map
 - Refreshing the view
 - Showing/hiding the overview map
 - Showing/hiding the table of contents
 - Showing/hiding the status console
- Map Actions
 - Select a DMS
 - Zoom out
 - Zoom in
 - Select multiple DMS'
 - Pan map
- Status Console Actions
 - Exiting
 - Tiling

CTMS/CTIS	Version: 1.6
GUI Detailed Design	Date: 03-Dec-04

- Cascading
- Showing/hiding the logging window
- Showing/hiding the instruction queue window
- Showing/hiding the alarms window
- Restoring the console windows to default size and position
- Canceling an instruction
- Acknowledging an alarm

- Managing Users
 - Showing the Manage Users dialog
 - Adding users
 - Editing users
- DMS Actions
 - Checking sign status
 - Polling the sign status
 - Displaying the device status dialog
 - Adding a sign
 - Removing a sign
 - Configuring a sign
 - Clearing a sign
 - Activating a sign message
 - Adjusting the brightness of a sign
 - Displaying the sign selection table

These actions will be loaded from an XML descriptor file and initialized upon first usage of the object. The actions that extend from the java AbstractAction object can then be attached to the UI buttons and/or menu items. In some cases, an action may only be accessible from one entry point. For instance, the add user action is only accessible from the manage users dialog. Other actions, such as polling the status of a sign, are accessible from a menu item and the sign status dialog. In all cases, the actions will be “routed” through an action in the action factory. This may in some cases appear to be a needless “round-trip”, but it improves quality and maintainability by applying the pattern consistently throughout the product.

3.3.1 UI Action Factory Class Diagram

The following diagram represents the UIActionFactory framework. The framework consists of a collection of AbstractActions that represent specific actions the user may perform while using the client application. All the actions are fully defined in an XML file that is defined in appendices. Notice that the diagram only represents a subset of the entire collection of actions available in the system. The UIActionFactory, the AbstractAction, the CTMSUIAction, and all the specialized CTMSUIActions represent the UI Action Factory.

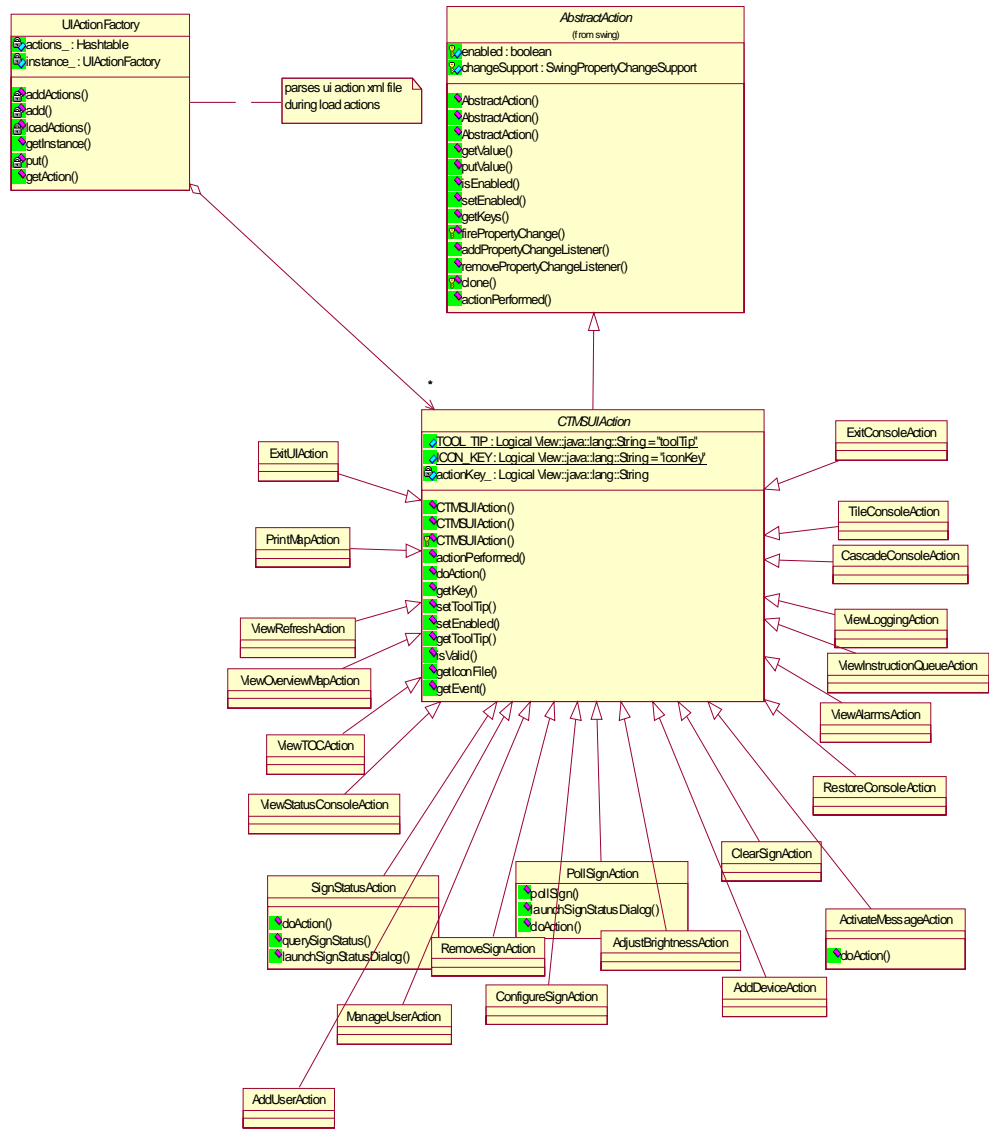


Figure 12: UI Action Factory Class Diagram

3.3.1.1 UIActionFactory

The UIActionFactory implements the GoF Singleton pattern. It maintains a set of static finals that represent the keys used to lookup the specific action in the contained hashtable of specialized CTMSUIActions. This class will provide

CTMS/CTIS	Version: 1.6
GUI Detailed Design	Date: 03-Dec-04

methods for retrieving the action given its unique key.

3.3.1.2 CTMSUIAction

This class is the base for all actions in the system. It will handle all the common getters and setters, but will not perform any of the actual action implementation. The doAction() method will be an abstract in the class.

3.3.2 *An Example Usage of the UI Action Factory*

This section fully describes the usage of one of the actions within the UIActionFactory. All other actions will operate very similarly, so only one such flow is described in great detail. The sequence starts with a user issues some command by committing some interaction, usually by selecting a menu item via the primary menu. In the case of Poll Sign, this action may arise through the primary menu, the pop-up menu, or a button on one of several DMS dialogs. The specialized action, PollSignAction will receive the action performed message and will launch the appropriate dialog, if it is not already launched (i.e. visible). The action will delegate the polling command to the J2EEClientWrapper in order to obtain the status. The details of the interaction with the J2EEClientWrapper will be discussed in a later section. This status will then be updated and made visible on the SignStatusDMSDialog. The SignStatusDMSDialog is modal, meaning that no other windows are available until the dialog is closed. Although not shown on the diagram, the PollSignAction will obtain the status of the sign by referencing an EJB via the J2EEClientWrapper. It is this status that will be used to update the dialog. Any errors will be logged by the J2EEClientWrapper and an exception will be handled in the actual action.

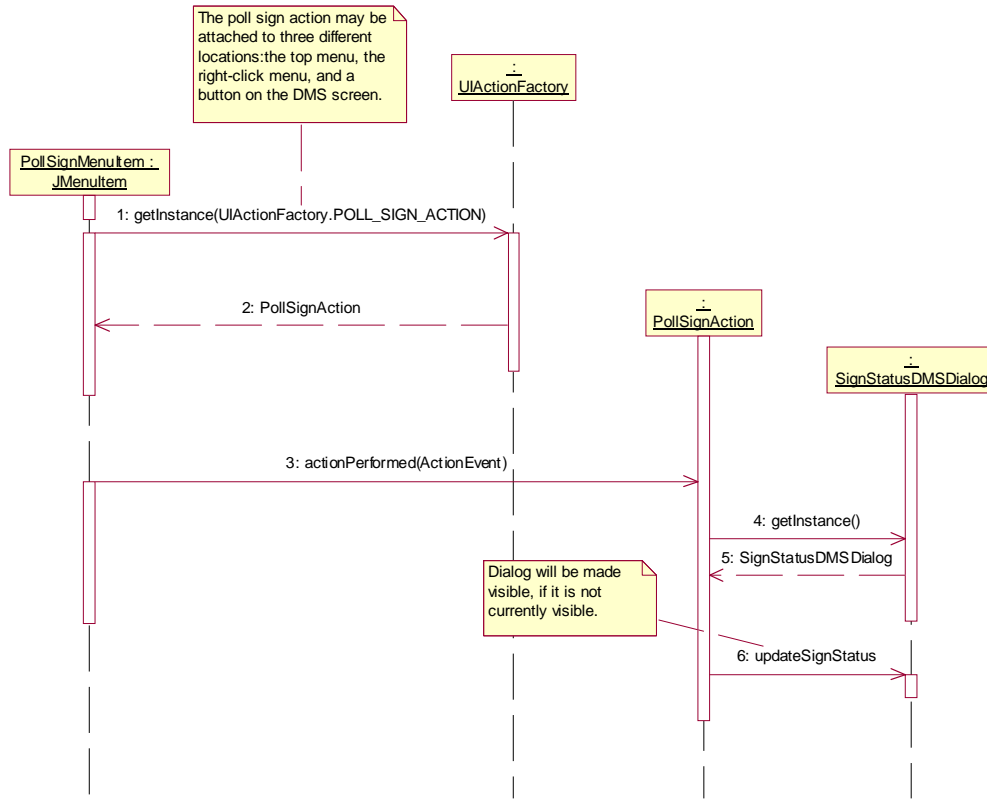


Figure 13: Usage of UI Action Factory

3.4 Message Queue Framework

The client system requires the monitoring of a JMS subsystem for obtaining system status information, state changes, alerts, and feedback from asynchronous requests. For performance reasons, the JMS subsystem should not be blocked for a significant amount of time while the incoming messages are being serviced. Instead, the incoming messages should be quickly handled and the JMS subsystem freed to handle additional messages. This is achieved by minimizing the JMS onMessage() method. The framework described here discusses the use of an internal queue that will hold the incoming JMS messages until the UI is ready to handle them.

3.4.1 Message Queue Framework Class Diagram

This section presents the class diagram that details the common architecture for the Message Queue Framework. This framework is used in all components in which a UI element (i.e. table) is dependent on JMS stimulus. Specifically, all Status Console windows will use the Message Queue Framework



Figure 14: Message Queue Class Diagram

3.4.1.1 JMSHandler

The JMS Handler is instantiated by the J2EEClientWrapper. It is responsible for initializing the connection to the appropriate topic, instantiating the queue, and starting the monitoring of the queue.

CTMS/CTIS	Version: 1.6
GUI Detailed Design	Date: 03-Dec-04

3.4.1.2 MessageQueue

The MessageQueue is the base class that handles the core responsibilities of managing a queue. This class provides the JMS onMessage() method that is called by the JMS subsystem. This method is optimized so that it quickly adds the message to the queue and returns. This class offers a method for making a thread that will monitor the queue until new messages are ready to be serviced. This thread will sleep until it is notified. Once notified, it will fire off the messages to all interested parties. To show interest, an object will add itself as a MessageListener..

3.4.2 Benefits of Message Queue Framework

The framework minimizes the amount of time the client denotes to capturing incoming messages. This is done by minimizing the onMessage() call. The result is that more messages may be handled at a higher rate. Without the Message Queue approach, an incoming JMS message would essentially block the client until the UI completed its update. Any incoming messages would have to wait for this action to complete.

3.5 Map Navigator

This section will detail the design for two packages in the CTMS client system; mapNavigator and map. MapNavigator is responsible for the outer container and framework that is required to display the primary map screen. This package will coordinate with the map package. The map package contains ALL the MapObjects specific definitions and implementations. After discussing the modeling for the architecture, the details pertaining to the integration of GIS data will be discussed.

← Formatted: Bullets and Numbering

3.5.1 Map Navigator Class Diagram

This particular section describes classes that are responsible for the generation of the primary map screen, or screen 1. Screen 1 is the GIS interface that gives user a Map based representation of devices. This map container, defined by MapNavigatorFrame, provides navigation to other related screens such as User administration screen, Poll DMS screen, configure DMS screen, etc. These options are available through the menu system that is configurable with an external XML file.

← Formatted: Bullets and Numbering

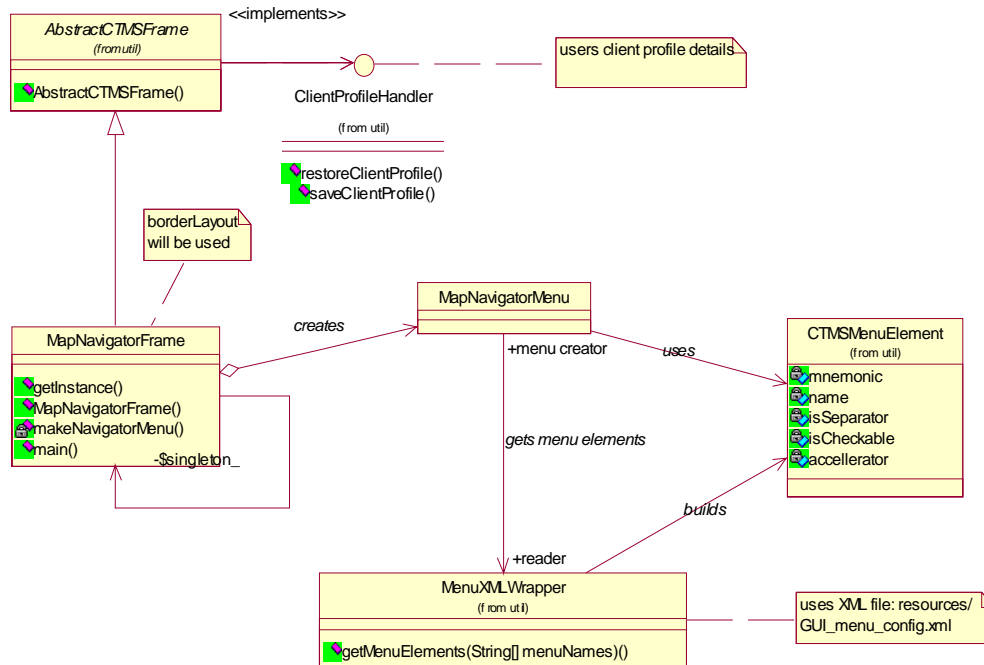


Figure 15: Map Navigator Class Diagram

3.5.1.1 MapNavigatorFrame

This particular class is responsible for the creation of the frame for the Map Navigator screen. Since the client application will only contain one map, the MapNavigatorFrame will follow the GoF Singleton pattern. This will provide a global access point that is especially useful for controlling the menu. Although this release may not need advanced menu control, future releases may require enabling and disabling of menu items based on user selections. The global access point provides a quick way to update the menu set based on interdependencies. The actual construction of the UI elements of the frame will be divided into several methods that assist in componentizing the UI elements. This means the Composite pattern is applied at a method, or wholly contained object level, as well as the aggregate object level. For instance, the method makeNavigatorMenu() will assist in creating main menu for the this particular frame. The MapNavigatorMenu is a composite part of the MapNavigatorFrame.

Formatted: Bullets and Numbering

3.5.1.2 MenuXMLWrapper

All menu items for the frames are specified in an external xml file called – GUI_menu_config.xml (see Appendix). The MenuXMLWrapper class uses the JDOM (open source) XML parser to parse the xml file and create a java collection, ArrayList, for each menu list. The ArrayList is returned to the MapnavigatorMenu class for the actual creation of the menu and menuitems.

Formatted: Bullets and Numbering

3.5.1.3 MapNavigatorMenu

Formatted: Bullets and Numbering

MapNavigatorMenu is responsible for the creation of the menubar, menus and menu items. This class gets menu elements from the MenuXMLWrapper and uses CTMSMenuElements.

3.5.1.4 CTMSMenuElement

Formatted: Bullets and Numbering

This class is an abstraction of the details of the menu element. This class is instantiated by the MenuXMLWrapper class and given to the MapNavigatorMenu class for the actual creation of the menu.

3.5.2 Map Navigator Sequence Diagram

Formatted: Bullets and Numbering

This particular section details the sequence of events that occur while generating the MapNavigator frame. The first step is to get properties for the frame such as the frame title, imageicon etc. To get the frame title we get an instance of the ApplicationProperties object and call its getProperty() method. ImageIcon is fetched using helper method of the CDOTUIUtilities class. After the main frame is ready, the next step is to create a MapNavigator Menu . The MapNavigatorMenu creates an instance of the UIActionFactory to get all actions. getSecurityManager() method of the J2EEClientWrapper class handles a securitymanager object. This object determines whether a particular menu item should be enabled or disabled for the specific user that has logged into the system. All menu items are created regardless of their state. addUIActions() method creates the menu with the menu items and submenus.

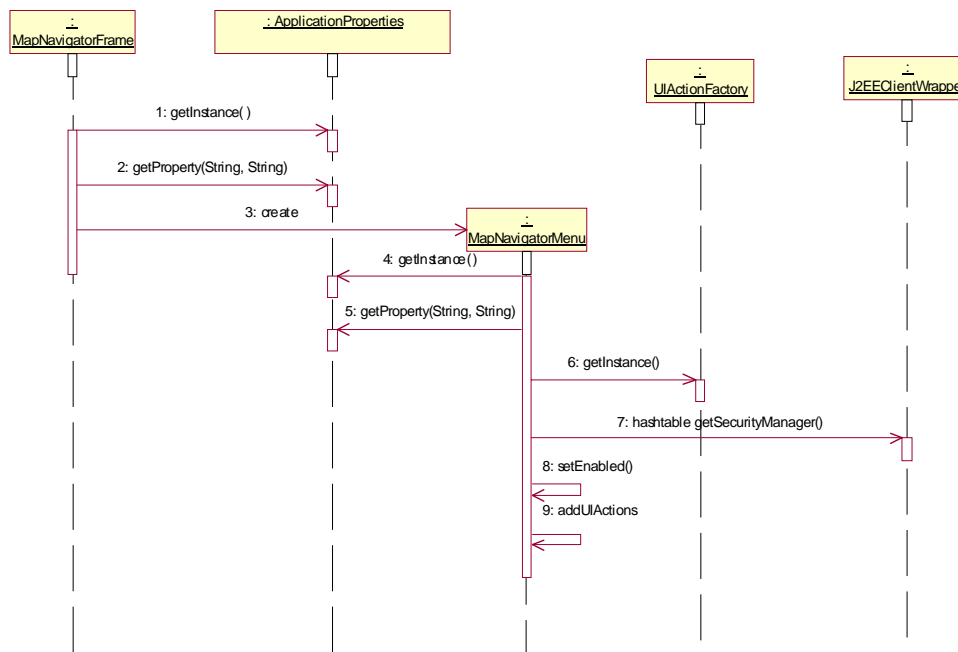


Figure 16: Map Navigator Sequence Diagram

3.5.3 Map Class Diagram

Formatted: Bullets and Numbering

This particular section describes the classes that generate the components associated with the Map such as the table of contents, toolbar, overview map, scale bar and others. The following classes share responsibility for generating

individual components. CTMSMapPanel will use the Composite pattern and will coordinate the construction of the aggregate components.

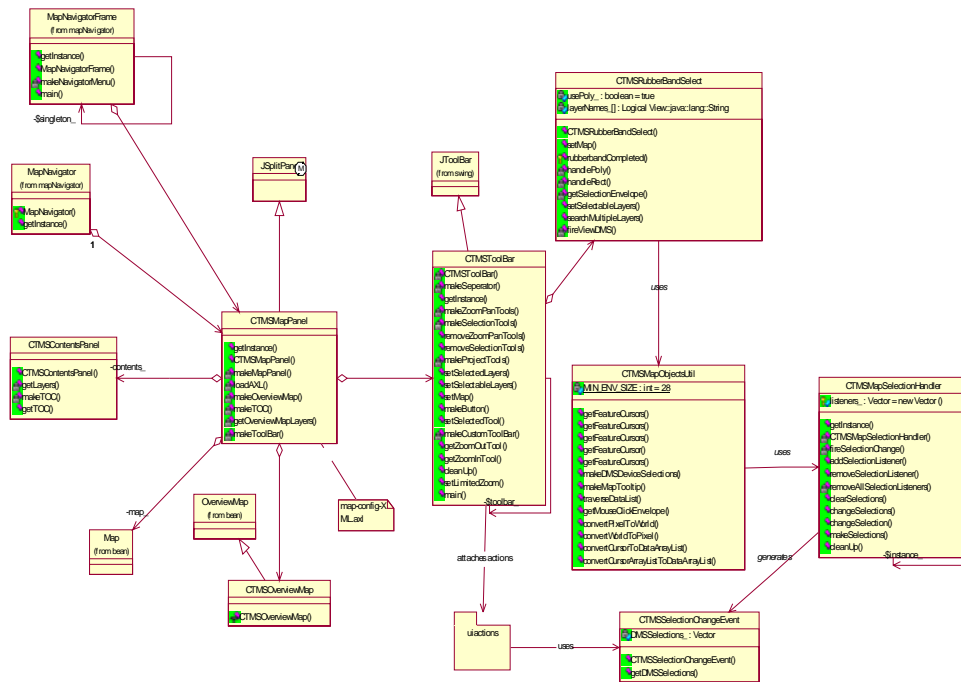


Figure 17: Map Class Diagram

3.5.3.1 CTMSMapPanel

This is the delegate class that is responsible for rendering the map interface. All layer objects attached to the map component are specified in an ArcIMS proprietary AXL version 1.1 file. Map-Config-XML.xml file. Each layer object is specified with the renderers (ex. raster image). Layers such as highways specify the shields if any. CTMSMapPanel class parses the xml file and attaches all layer objects to the specified map component. This class holds the main reference to the map component. The reference is passed to secondary classes that generate individual components for the main map. The CTMSMapPanel will extend the JSplitPane object. A BorderLayout will be used by CTMSMapPanel. Once the delegated construction of the table of contents and the overview map and complete, these objects will be placed into another JSplitPane and a BorderLayout will be used.

Formatted: Bullets and Numbering

3.5.3.2 CTMSContentsPanel

This class is responsible for generating the table of contents (toc) for the layer objects. Each layer attached to the map is displayed as a legend node object in the toc. The toc gives control to make the layers visible on the map component. CTMSContentsPanel makes two toc objects for the static and dynamic layers respectively. Static layers include highways, statelines etc whereas the dynamic layers include layers related to the device ex. DMS layer.

Formatted: Bullets and Numbering

CTMS/CTIS	Version: 1.6
GUI Detailed Design	Date: 03-Dec-04

3.5.3.3 CTMSOverviewMap

This class is responsible for generating the overview map component for the specified map. Layers selected for the overview map component are specified in this class. The outer and inner fill for the overview map panel are also specified here. Visibility of the overview map will be user specific and will be saved as a part of the client profile.

Formatted: Bullets and Numbering

3.5.3.4 CTMSMapToolBar

A singleton instance of this class extends from the swing JToolBar class. This class is responsible for all the interactions available to the user. Interactions may include some predefined MapObjects actions such as zoom in, zoom out, find, print map. Custom actions include refresh and custom rubber band. All actions are defined within the cdot.ctms.ui.uiactions package.

Formatted: Bullets and Numbering

3.5.3.5 CTMSRubberBandSelect

A class that defines the behavior associated with the custom single and multi-select rectangle and the multi-select polygon. All device selections performed by the user are handled by an instance of this class. This class heavily uses the CTMSMapObjectUtil class.

Formatted: Bullets and Numbering

3.5.3.6 CTMSMapObjectsUtil

This class contains several helper utilities for interchanging data between CTMS and MapObjects formats and querying the MapObjects collection of DB objects. This class will route selection changes to the selection handler which will fire the change events to the actions that are dependent upon selections (ActivateMessage, ClearSign, Poll, View Status, etc...)

Formatted: Bullets and Numbering

3.5.4 Map Sequence Diagram

This diagram shows the sequence of events that complete the generation of the main Map Navigator Frame. The MapNavigator class creates the outer container for the 'map' object. The MapPanel class defines the behavior and attributes needed for creating the map panel. It is composed of several classes that represent the individual map objects components. The initial steps require creation of map panel and loading the axl project. The MapNavigator class acts as a delegate class that handles the main map reference to the aggregate components for their own construction and later add them to the main map panel. makeTocPanel() method creates and renders a 'mapobjects' specific treetoc object and returns it to the CTMSMapPanel. makeOverviewMap() method creates and renders 'mapobjects' specific overviewmap object and returns it to the CTMSMapPanel. makeToolBarPanel() method creates and renders a JToolBar with various interactions for GUI specific operations. Actions for the interactions are specified in UIActions package. Finally, MapPanel class re-renders all components including Map, TreeToc, OverviewMap and toolbar in the refreshClient() method to create a unified updated view of the MapNavigator Frame.

Formatted: Bullets and Numbering

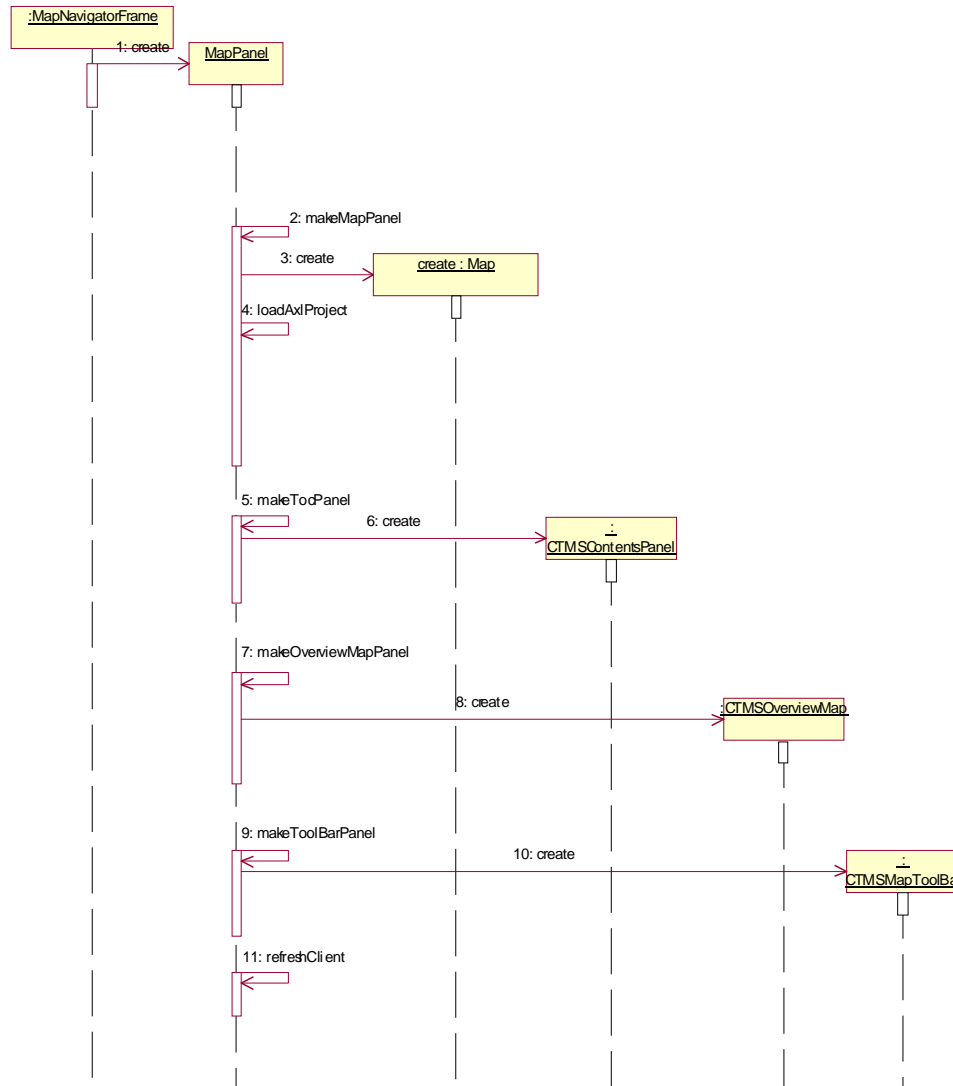


Figure 18: Map Sequence Diagram

3.5.5 Map Selection Sequence

This diagram details the flow of events when a user makes a selection on the map. The tool is an instance of CTMSRubberBandSelect in this case. When the mouse clicks are completed, the utility class is used to obtain the selected features. During this process, the selection handler will be notified that the selections need to be performed.

Formatted: Bullets and Numbering

This method will clear the selections, update the ui selected objects and notify all the listener actions that a change in selections has been performed.

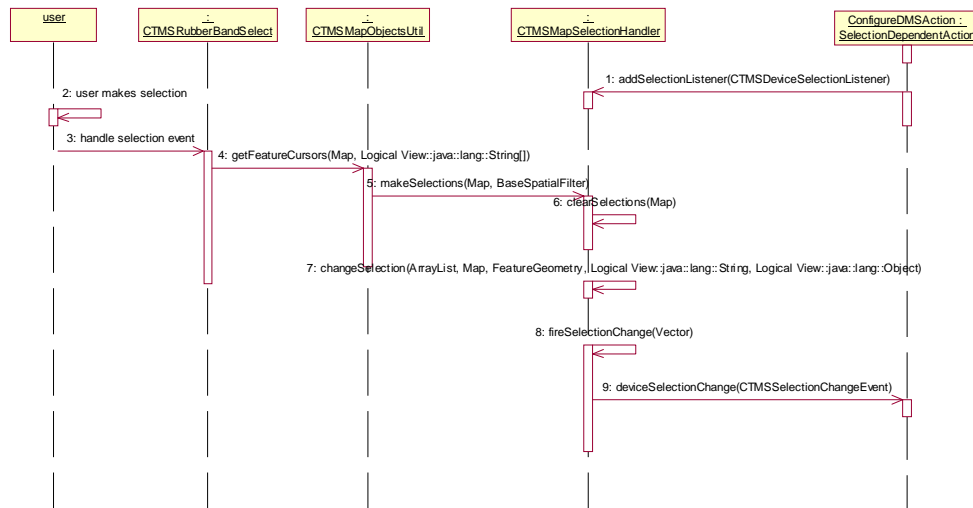


Figure 19: Selection Framework Sequence Diagram

3.5.6 Integration of GIS Data

This section will describe how the GIS data will be used by the map package. As previously stated the map package will be tightly coupled with the ESRI MapObjects API.

Formatted: Bullets and Numbering

3.6 Status Console

This section describes the classes responsible for the generation of the Status Console, or screen 2. This screen will present the user will a large frame that contains free-floating windows that are movable, resizable, and minimize/maximize-able. The free-floating windows will contain details on alarms, pending/running instructions, and logged events. These windows will consist of tables that relay the details of the inner workings of the CTMS system. The updates for these views are provided via a JMS subsystem. For a discussion on the client framework used to monitor the JMS subsystem, please refer to the section titled [Message Queue Framework](#).

Formatted: Bullets and Numbering

3.6.1 Status Console Class Diagram

The following diagram provides the details of the Status Console Framework. The statusConsole package will contain all the framework for supporting the free floating frames. It will also contain the panel details used for each of these internal frames. The actual table details, such as the renderers and models, will reside in the specialized table package.

Formatted: Bullets and Numbering

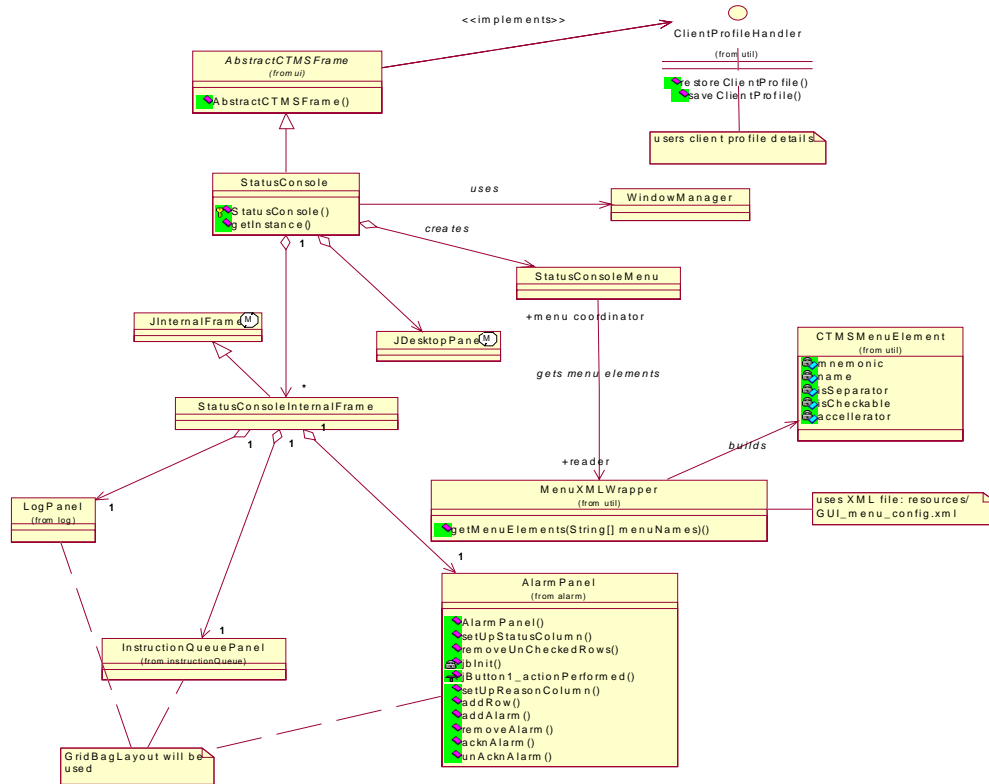


Figure 20: Status Console Class Diagram

3.6.1.1 StatusConsole

This class generates the parent frame for screen 2. It maintains a singleton instance of the class at all times. This class handles attributes and behaviors of the main frame. Furthermore, it delegates to aggregate components for their own construction and adds them to its main frame. It also manages behavior of the child panels.

Formatted: Bullets and Numbering

3.6.1.2 StatusConsoleMenu

This class is responsible for the creation of the menubar, menus and menu items. This class gets menu elements from the MenuXMLWrapper and uses CTMSMenuElements.

Formatted: Bullets and Numbering

3.6.1.3 StatusConsoleInternalFrame

This class extends a JInternalFrame and establishes a generic frame that is used by LogPanel, InstructionQueuePanel and AlarmPanel. Having a generic frame helps control common behaviors such as default window closing or maximizing behavior etc.

Formatted: Bullets and Numbering

CTMS/CTIS	Version: 1.6
GUI Detailed Design	Date: 03-Dec-04

3.6.1.4 LogPanel

This class extends the StatusConsoleInternalFrame and is responsible for creating the log panel. It is also responsible for handling the display of the logs.

Formatted: Bullets and Numbering

3.6.1.5 InstructionQueuePanel

This class extends the StatusConsoleInternalFrame and is responsible for creating the Instruction panel. It is also responsible for handling the display of the logs.

Formatted: Bullets and Numbering

3.6.1.6 AlarmPanel

This class is responsible for creating the alarm panel, alarm table and managing operations on the alarm objects. It subscribes to incoming alarms and displays the alarms in a tabular fashion. Unacknowledged alarms are rendered differently than acknowledged alarms. When the user acknowledges an alarm, this class is responsible for broadcasting the user event to other users.

Formatted: Bullets and Numbering

3.6.1.7 MenuXMLWrapper

The MenuXMLWrapper (of the util package) has been fully discussed in the MapNavigator description. The class will be reused in the StatusConsole package.

Formatted: Bullets and Numbering

3.6.1.8 CTMSMenuElement

The CTMSMenuElement (of the util package) has been fully discussed in the MapNavigator description. The class will be reused in the StatusConsole package.

Formatted: Bullets and Numbering

3.6.2 Status Console Creation Sequence Diagram

This particular section details the sequence of events that occur while generating the Status Console frame. The first step is to get properties for the frame such as the frame title, imageicon etc. To get the frame title we get an instance of the ApplicationProperties object and call its getProperty() method. ImageIcon is fetched using helper method of the CDOTUIUtilities class. After the main frame is ready, the next step is to create a StatusConsole Menu. The StatusConsoleMenu creates an instance of the UIActionFactory to get all actions. getSecurityManager() method of the J2EEClientWrapper class handles a securitymanager object. This object determines whether a particular menu item should be enabled or disabled for the specific user that has logged into the system. All menu items are created regardless of their state. addUIActions() method creates the menu with the menu items and submenus. After the creation of menu, the internal frames are created. StatusConsoleInternalFrame is the generic frame class that delegate creation of individual panels such as log panel etc.

Formatted: Bullets and Numbering

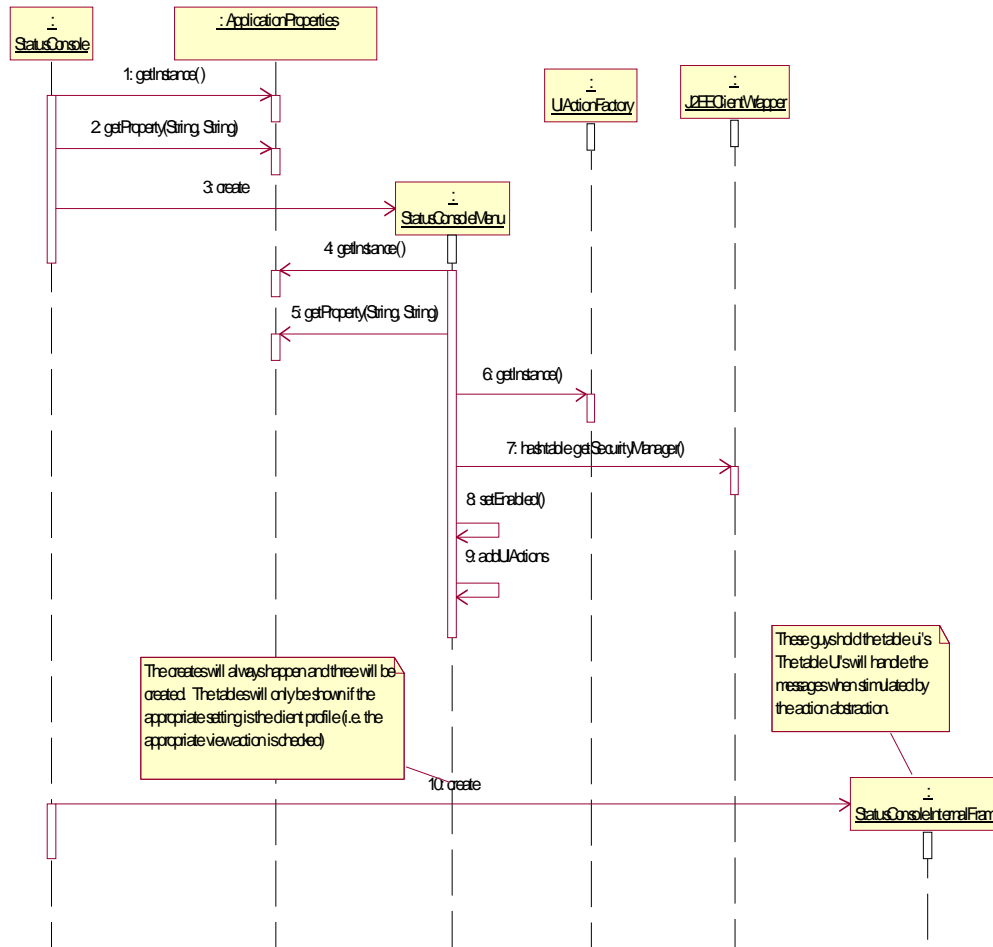


Figure 21: Status Console Sequence Diagram

3.7 Status Console Windows – Alarms, Logging, and the Instruction Queue

Formatted: Bullets and Numbering

The Status Console component of the CTMS client will be responsible for managing a collection of UI windows that allow for viewing the status of the system. The three windows will be composed tables and, in some cases, buttons that present a view of the alarms, log, and pending instructions. Although the information between these windows is very different, the framework for the presentation of the information, the tie into the model for manipulating the presented data, and the control logic for catching JMS updates and driving the presentation are the same. This section will present a brief description of what is handled by each table and will provide extensive details for the alarm table in order to communicate the framework. The components discussed in this section reside within the alarms, logging, and instructionqueue packages of table. The support for the internal frames that hold the panels consisting of the tables and other ui components resides in the statusconsole package.

3.7.1 Alarms

A user will be able to view any unacknowledged alarms. Alarms represent any problems that have occurred during the execution of a DMS command issued by any user. This section will detail the static structure of the alarm component and will describe the framework employed to handle an incoming alarm JMS message.

Formatted: Bullets and Numbering

3.7.1.1 Alarm Messaging Class Diagram

The following class diagram details the infrastructure behind the alarms component of the client system. The focus of this diagram is on the details of the event handling mechanism. The action/event related components, the UI elements, and the alarm message interfaces will be in the tables.alarms package. A major aspect of the alarm presentation is the alarm table, which resides in the tables.alarms package. The alarm table will be an aggregate component of the AlarmPanel. It is not shown on the diagram for the sake of simplicity. All renderers and table models used in the construction of the table will be placed in the tables.alarms package. This diagram does not discuss the listeners, the JMS system, or the action controller. A top-level discussion of the listeners is available in the previous section titled Proxies and Delegates. The ActionController is fully discussed in the server architecture document.

Formatted: Bullets and Numbering

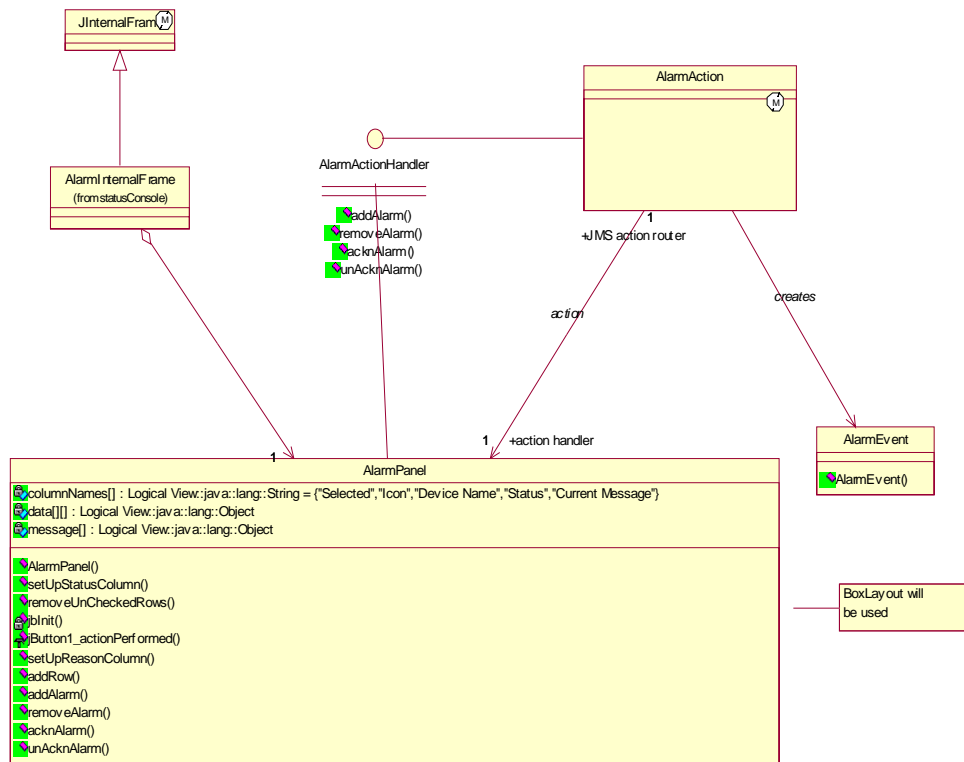


Figure 22: Alarm Messaging Class Diagram

CTMS/CTIS	Version: 1.6
GUI Detailed Design	Date: 03-Dec-04

3.7.1.2 AlarmPanel

The alarm panel will be placed into a specialized JInternalFrame. The AlarmInternalFrame and AlarmPanel will reside in the package of the status console, statusconsole. The AlarmInternalFrame will handle generic instantiation, sizing, and client profile details while delegating the specific of the panel to AlarmPanel. The specifics of the table that are on the alarm panel will be delegated to the AlarmTable which resides in the alarm subpackage of the table package. This panel will use a BorderLayout. The addRow() will delegate the table specifics to the AlarmTable (not shown). The panel will implement the AlarmActionHandler interface.

Formatted: Bullets and Numbering

3.7.1.3 AlarmAction

The AlarmAction is contained by the ActionFactory (not shown). Upon receipt of a JMS message the action controller will retrieve the appropriate action and invoke the appropriate calls on it. The alarmAction will create the AlarmEvent and pass it along to the alarm handler.

Formatted: Bullets and Numbering

3.7.1.4 Alarm Sequence Diagram

This sequence diagram details the flow of events that occur upon receipt of a JMS message. As the message arrives the appropriate listener is notified. In this particular case, the alarmMessageListener is notified. The listener will create the alarm event and will delegate the processing of the event to the action controller. The controller will retrieve the appropriate action and invoke the methods on the object assigned to the particular action. In this case, the AlarmAction is used. The alarm action delegate the work of the actions to the AlarmPanel itself. Although not detailed on the document below, the JMS System will not "block" with the onMessage() call. The framework will include a queue mechanism (queue and queue monitoring thread) that will free the JMS system for further processing of messages. The queue thread will take the messages off the queue and hand them over to the actions for processing. This should significantly reduce wait times and message build-ups, thus improving performance.

Formatted: Bullets and Numbering

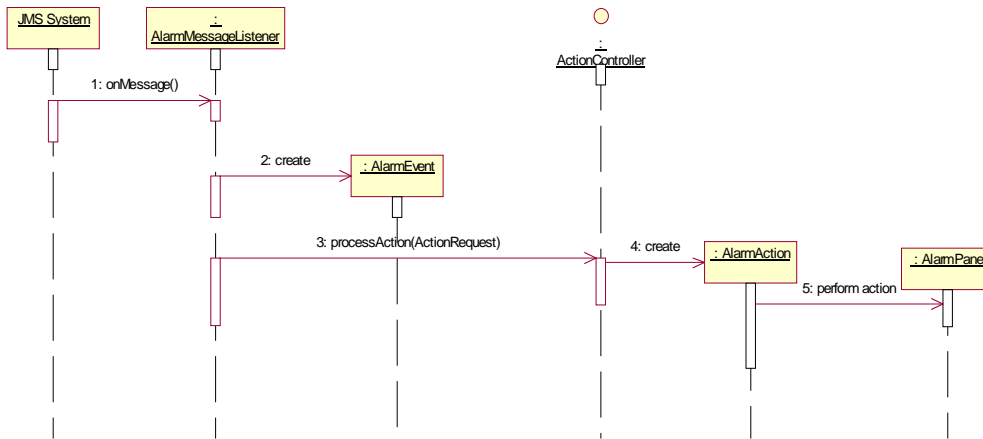


Figure 23: Alarm Sequence Diagram

3.7.2 Logging

A user will be able to view a log of any issues that have arisen during the processing of the system.

Formatted: Bullets and Numbering

3.7.3 Instruction Queue

A user will be able to view any pending or running DMS instructions. The table will quickly communicate the device identifier, the instruction identifier, the brief instruction description, the start time, the priority, the status, and an owner. In addition to the table the panel will also include a button for canceling one or more selected instructions. An

Formatted: Bullets and Numbering

instruction must be selected in order for the button to be enabled. The server will issue an instruction status JMS message when a request is made and when the state changes. Also, the client system will be able to request for the status of all instructions. When the instruction queue window is “launched” by starting the client or by selecting View Instruction Queue menu item, the model for the Instruction Queue will request the active set of messages and their status. This information will then be used to populate the table.

3.7.3.1 Instruction Queue Class Diagram

Formatted: Bullets and Numbering

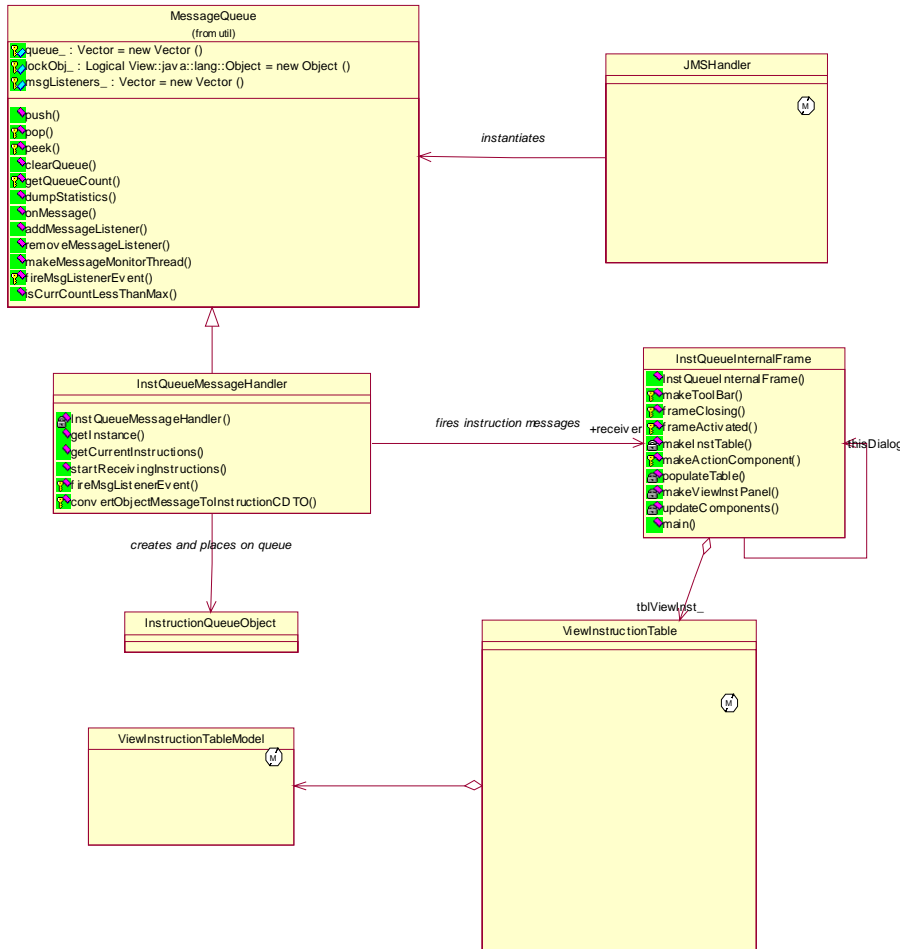


Figure 24: Instruction Queue Class Diagram

Formatted: Bullets and Numbering

3.7.3.1.1 Message Queue

The message queue object is used during JMS processing to assist in relieving the bottleneck that can occur during the

CTMS/CTIS	Version: 1.6
GUI Detailed Design	Date: 03-Dec-04

receipt of JMS messages. The queue is added as the JMS message handler and will quickly add the new message onto the queue and return from the onMessage callback. All queue service threads that are waiting for objects to be present on the queue will be woken, so that they may perform whatever operations are needed. The MessageQueue object contains an inner class for tracking statistics. Since the incoming messages are already organized by topics, there is no need to combine all the messages are resort, therefore, a message queue is instantiated and assigned to each JMS topic.

3.7.3.1.2 InstQueueMessageHandler

The specialized MessageQueue that will handle messages for the instruction queue topic. This object is a singleton, since it is needed by the JMSHandler (to connect it to the topic) and by the InstQueueInternalFrame (for UI updates).

← Formatted: Bullets and Numbering

3.7.3.2 JMS Handler

The JMS Handler resides in the delegate package. This class defines the behavior for connecting to the CTMS JMS topics.

← Formatted: Bullets and Numbering

3.7.3.2.1 InstQueueInternalFrame

The internal frame that is responsible for showing the instruction queue. This frame handles all the sizing and routes the table updates that are driven by the receipt of JMS messages.

← Formatted: Bullets and Numbering

3.7.3.2.2 ViewInstructionTable

This class defines the table that holds the instructions.

← Formatted: Bullets and Numbering

3.7.3.2.3 ViewInstructionTableModel

The class defines the model that holds the data shown in the instruction queue table.

← Formatted: Bullets and Numbering

3.7.3.3 Instruction Queue Sequence Diagram

This sequence diagram details the flow involved in connecting to the JMS instruction topic and handling the messages. The JMS Handler will instantiate the specialized MessageQueue, InstQueueMessageHandler. Once the InstQueueHandler is instantiated, all the instructions will be retrieved and updated in the UI. As messages arrive via the onMessage, the queue will be populated and all sleeping "monitor" threads will be woken so that the collection of new messages can be serviced by the UI components.

← Formatted: Bullets and Numbering

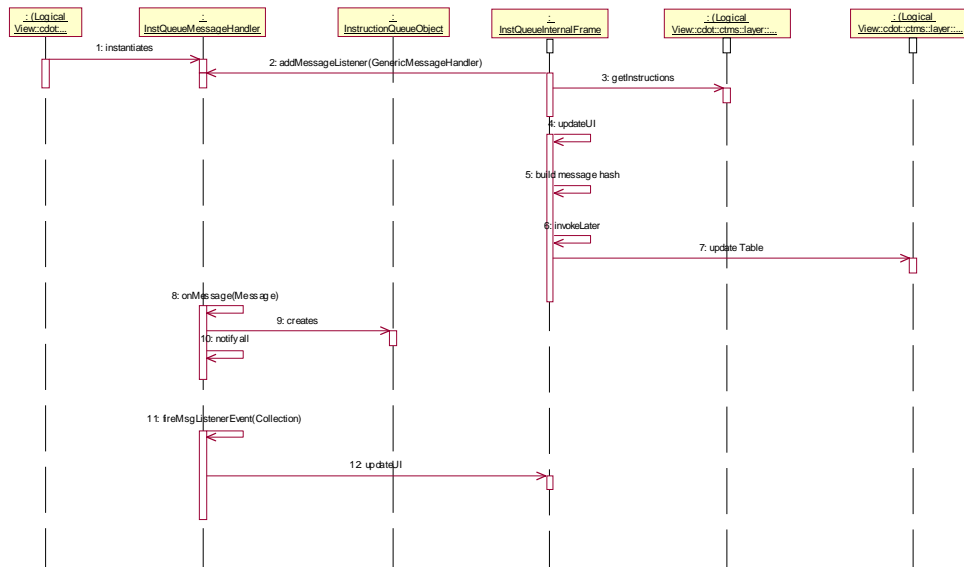


Figure 25: Instruction Queue Sequence Diagram

3.8 DMS Services

The CTMS client will provide a set of services for dealing with DMS devices. The services include viewing device details, activating a message, checking the device status, adjusting the brightness, and clearing the sign.

Formatted: Bullets and Numbering

3.8.1 DMS Services Class Diagram

This section will present the overall class diagram for the usage of DMS services. All requests will be generated by a user interaction with the system either by selecting a menu item or clicking on a button. The specialized action will then delegate the actual “device” interaction to the J2EEClientWrapper which encapsulates the RemoteControllerProxy. The RemoteControllerProxy will issue the command to the server. For the sake of simplicity, the J2EEClientWrapper is not included in the diagram. Only the end result, usage of the RemoteControllerProxy, is shown. The following class diagram only shows a few of the actions and the corresponding dialogs, since all other dms service dialogs will follow the same exact pattern.

Formatted: Bullets and Numbering

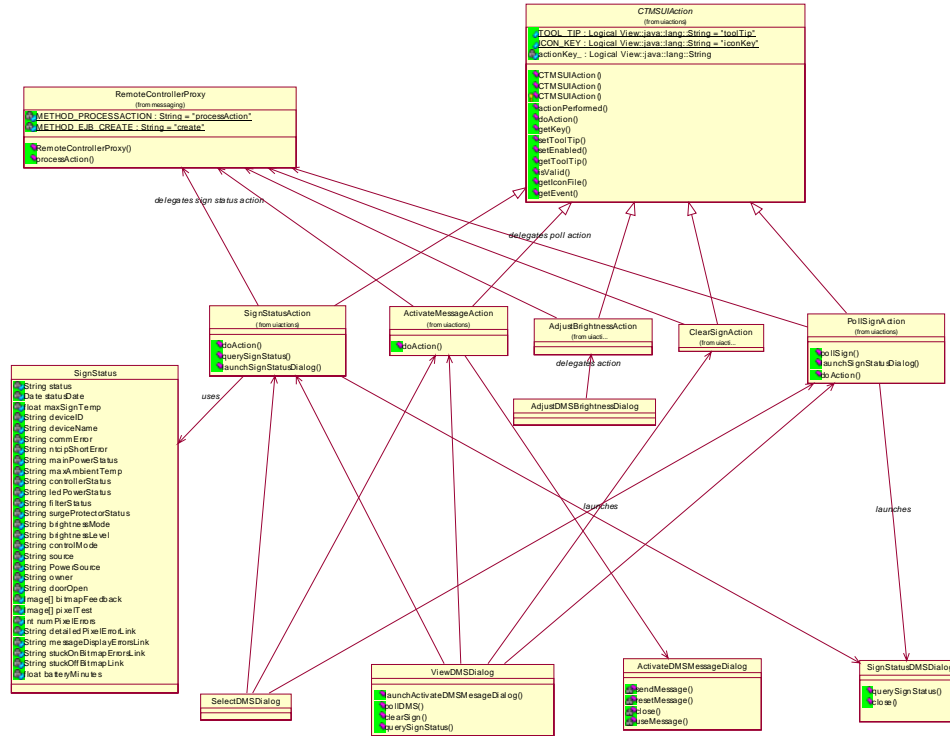


Figure 26: DMS Services Class Diagram

3.8.1.1 RemoteControllerProxy and J2EEClientWrapper

These objects manage the requests from the client and route the command to the server. They also will coordinate the reply.

Formatted: Bullets and Numbering

3.8.1.2 SignStatusAction

This action will use a SignStatus object to relay the status of the sign to the calling component. The action may be initiated from the selectDMSDialog, and the ViewDMSDialog. This action will launch a SignStatusDMSDialog in order to present the results.

Formatted: Bullets and Numbering

3.8.1.3 ActivateMessageAction

This action may be initiated by the selection of menu item or a button from the SelectDMSDialog or the ViewDMSDialog. It will launch the ActivateDMSMessageDialog.

Formatted: Bullets and Numbering

3.8.1.4 AdjustBrightnessAction

The selection of a menu item will launch the AdjustBrightNessDialog that will be used to adjust the brightness of the sign.

Formatted: Bullets and Numbering

CTMS/CTIS	Version: 1.6
GUI Detailed Design	Date: 03-Dec-04

3.8.1.5 ClearSignAction

From the menu or from the ViewDMSDialog, a user will be able to clear a sign.

Formatted: Bullets and Numbering

3.8.1.6 PollSignAction

This action will be accessible from the SelectDMSDialog or the ViewDMSDialog. The action may also be initiated by selecting a menu item. This action will launch the SignStatusDialog (if it is not already being displayed) and update the status of the device.

Formatted: Bullets and Numbering

3.8.2 *Activate Message Sequence Diagram*

This section will detail the sequence of events that occur when a user activates a message. The action is performed given some user interaction. The action will create the activateMessaegDialog, add itself as a listener, and initialize the dialog. The initialization will consist of setting the list of banned words and the library of previously defined messages. These items will come from cached copies, if available. Otherwise, the objects will be retrieved from the server. Once the user enters the message and hits send, the message will be checked for spelling errors and then will be checked against a set of banned words. Once everything is validated, the activateMessage event will be fired and processed by the action. The action will work with the J2EEClientWrapper to send the request to the server via an EJB.

Formatted: Bullets and Numbering

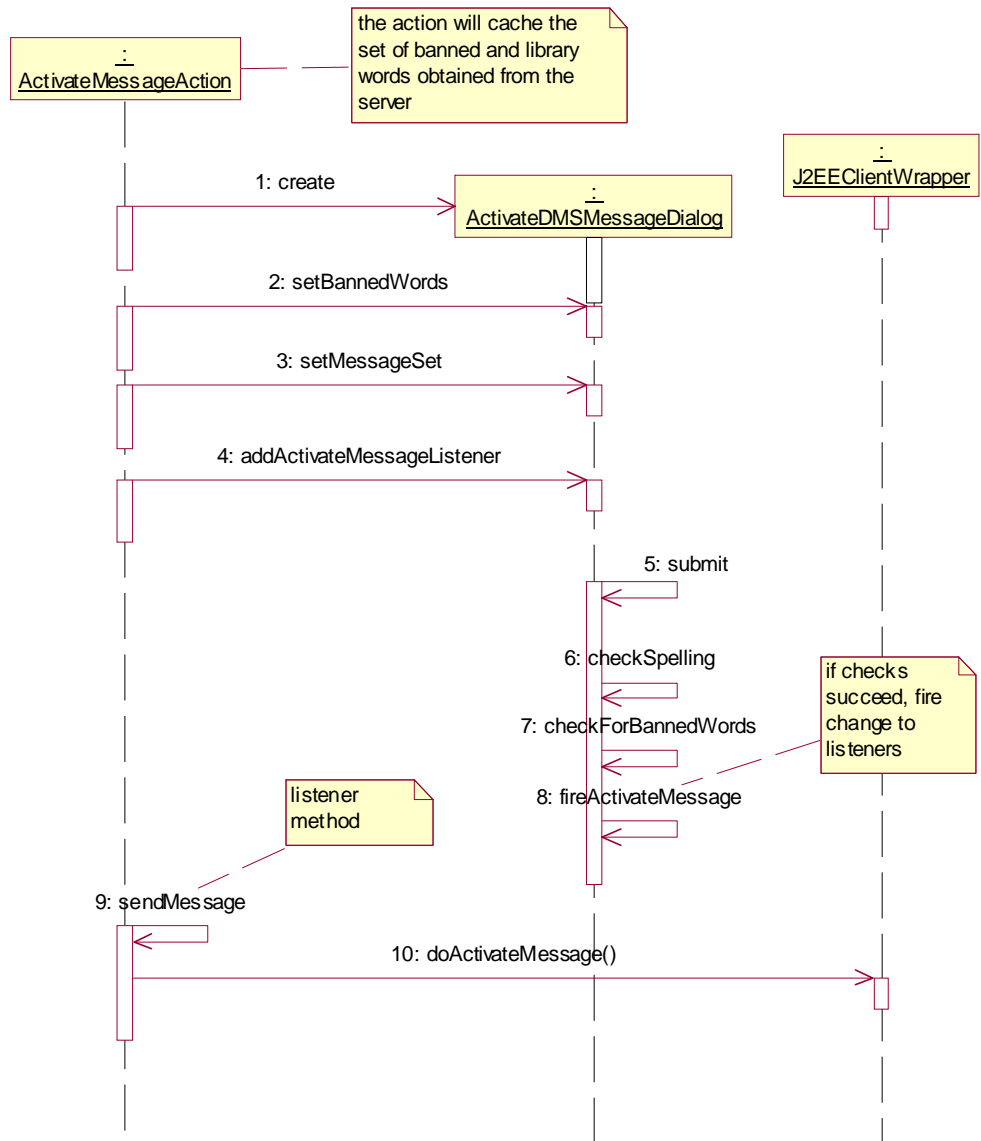


Figure 27: Activate Message Sequence Diagram

3.8.3 WYSIWYG Message Editor Class Diagram

One of the most important features of the CTMS client is the ability to view and edit sign messages as they appear on the physical sign. During the analysis phase, a lot of research was performed in order to find solutions to this problem. Unfortunately, nothing else exists for solving. The original idea was to create a custom font and then use basic Java Swing components to render this font, but this proved to be inadequate. Specifically, a mechanism for controlling spacing around the fonts was not complete enough in the Swing tool set. As a result, the WYSIWYG (What You See is What You Get) viewer/editor progressed into an image processing problem. The solution catches user keystrokes, constructs a small image on-the-fly, then replaces a small region of the sign image.

This section details the overall class diagram for the WYSIWYG Message Editor. The Editor is used by the Poll Sign Dialog, View DMS Sign Dialog and Activate Message Dialog. All three views use the same core framework, however, the message is only editable in the Activate Message usage. A factory pattern is applied in order to create the editable and non editable hierarchy.

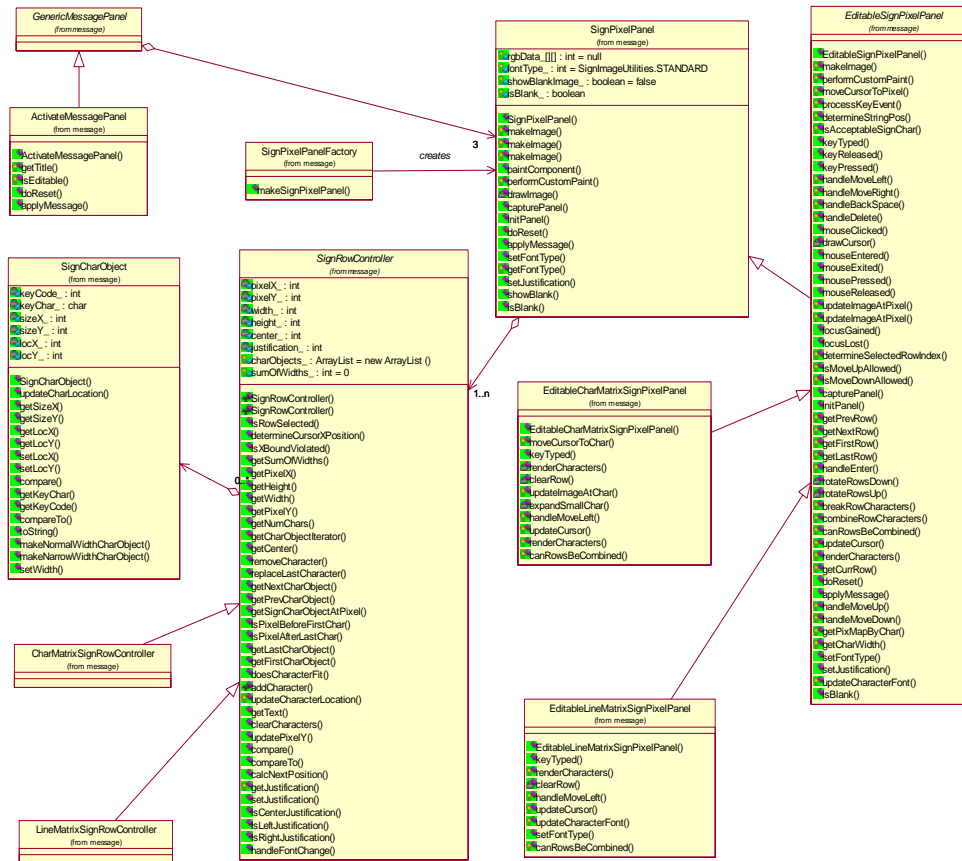


Figure 28: Message Editor Class Diagram

CTMS/CTIS	Version: 1.6
GUI Detailed Design	Date: 03-Dec-04

3.8.3.1 GenericMessagePanel

The basic container for the image presentation is the GenericMessagePanel. This panel presents common options and manages the sign image. This panel contains three SignPixelPanels; one for each sign page.

3.8.3.2 ActivateMessagePanel

A specialized version of the generic message panel that adds additional UI components to the base panel.

3.8.3.3 SignPixelPanel

The primary image panel. This class delegates the font and justification requests and handles the rendering of the image into the background of panel. This is done in a custom paint method. The appropriate SignPixelPanel (SignPixelPanel – noneditable or EditableSignPixelPanel – editable) is created through the use of the SignPixelPanelFactory. This class contains a collection of SignRowControllers; one for each row on the page.

3.8.3.4 EditableSignPixelPanel

This class specializes the SignPixelPanel in order to provide the base user interaction functionality for key presses, key handling, etc... This class is responsible for creating the character image and updating the whole image.

3.8.3.5 EditableCharMatrixSignPixelPanel

The EditableCharMatrixSignPixelPanel specializes the set of core user interactions that are specific to character matrix signs.

3.8.3.6 EditableLineMatrixSignPixelPanel

The EditableLineMatrixSignPixelPanel specializes the set of core user interactions that are specific to line matrix signs.

3.8.3.7 SignRowController

The base row controller handles the character inputs for a row of the image. The rows maintain a set of SignCharObjects, but do not provide any image processing capabilities. The row controller is responsible for determining if a character will fit on the row and will update character positions as they are added and deleted. The controller knows about and coordinates all characters in the row. It does not know about the other rows.

3.8.3.8 CharMatrixSignRowController

This class extends the base controller to specialize the behavior for character matrix signs.

3.8.3.9 LineMatrixSignRowController

This class extends the base controller to specialize the behavior for line matrix signs.

3.8.3.10 SignCharObject

The SignCharObject is the fundamental building block of the Message Editor/Viewer. A sign char object does not know anything about other characters on the sign, but it supplies everything needed by the image processing engine in order for character images to be rendered.

3.8.4 WYSIWYG Message Editor Sequence Diagram

This section details the sequence of events that occur from pressing a Key from the keyboard to actually rendering the character on the message editor. When a user presses a key several events are fired within the Swing framework. The EditableSignPixelPanel listens for these events. Once a key event is received, the key is checked to make sure it is an acceptable character. If it is then the character is created and given to the SignRowController. The controller checks to see if the character will fit, checks the justification, and adds the character in the proper place. This may require that all other characters held by the row controller have their positions updated. Once the character is added, a subregion of the image is cleared and the characters for the row are rendered.



Figure 29: Message Editor Sequence Diagram

3.9 DMS Management

Formatted: Bullets and Numbering

The CTMS system must facilitate the management (add, remove, configure, etc...) of devices. For phase one development the devices will only include DMS's. This section will discuss the designs of the management component of the CMTS client system. The following class diagram details the classes involved in managing DMS devices. The following diagrams detail the collection of classes related to the UI elements used to facilitate the addition, configuration, and removal of DMS devices. The management collection of classes uses a significant number of shared components.

3.9.1 Add DMS Implementation of Wizard Framework

Formatted: Bullets and Numbering

The client system offers a generic framework for wizard construction. This generic framework is implemented for the Add DMS component of the system. The generic aspects include the UI elements and actions for of the Back, Next, and Finish buttons, the title and description sections, and the coordination of the populating of the UI fields as the wizard is traversed. The logical connection between the panels, or pages, of the wizard is controlled by the individual components.

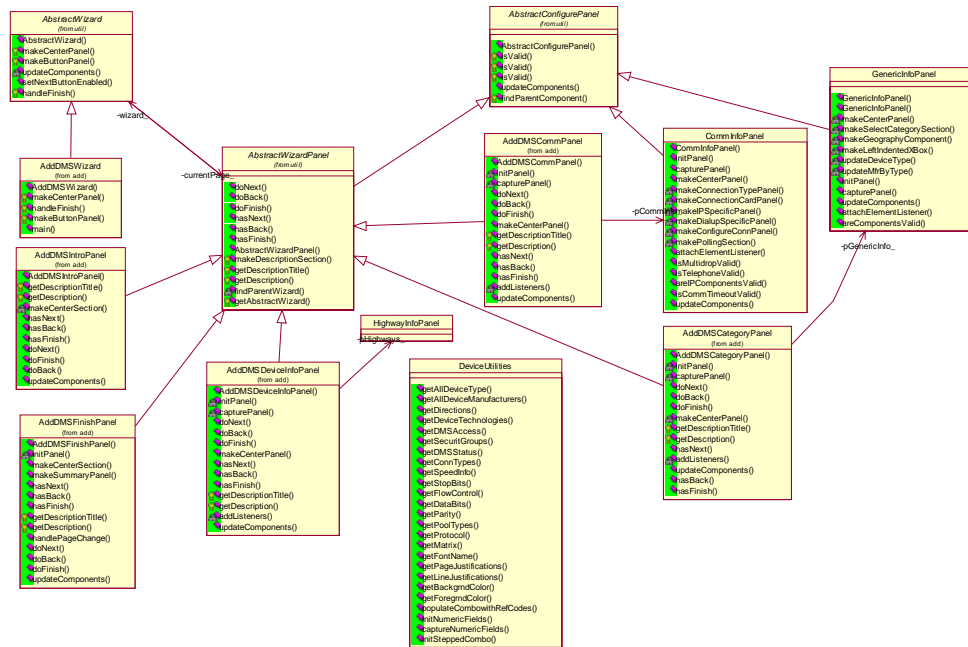


Figure 30: Add DMS Class Diagram

3.9.1.1 AbstractWizard

Formatted: Bullets and Numbering

The AbstractWizard class defines the core foundation of the generic wizard. It offers methods for constructing the various components of the wizard and controls the button actions that facilitate traversal through the wizard.

3.9.1.2 AddDMSWizard

Formatted: Bullets and Numbering

The AddDMSWizard class defines the specialized behavior the Add DMS wizard.

CTMS/CTIS	Version: 1.6
GUI Detailed Design	Date: 03-Dec-04

3.9.1.3 AbstractWizardPanel

This abstract class provides the abstract methods that facilitate the traversal through each page of the wizard. Each concrete class of this abstract class is responsible for establishing the traversal order. For instance, the AddDMSIntroPanel will implement the method for doNext(). This implementation will return the object for the next panel. Since this page does not have a finish or previous panel, these methods will return null.

Formatted: Bullets and Numbering

3.9.1.4 AddDMSIntroPanel

The introduction panel of the wizard. This panel will not have a previous button action.

3.9.1.5 AddDMSFinishPanel

The final panel of the wizard. This panel will not have next button action.

3.9.1.6 AddDMSDeviceInfoPanel

This panel will hold generic device details that the user can fill out. One of its primary components is the shared HighwayInfoPanel.

3.9.1.7 AbstractConfigurePanel

The AbstractConfigurePanel is used to share some of the non-specific details of UI element validation.

3.9.1.8 AddDMSCommPanel

The panel that holds the communications details

3.9.1.9 CommInfoPanel

The shared component that is responsible for the UI components needed by the communications aspects of the sign.

3.9.1.10 AddDMSCategoryPanel

The panel that holds the generic category information.

3.9.1.11 GenericInfoPanel

A shared component that holds the category details.

3.9.1.12 DeviceUtilities

A utility class that handles the behaviors of the set of UI screens.

3.9.2 Configure DMS Class Diagram

The following diagram details the ConfigureDMS class architecture. It shares the same common framework offered by the Add DMS architecture. Specifically, the HighInfoPanel, DeviceUtilities, AbstractConfigurePanel, and a few others. Note that all these classes are not repeated on the configure DMS diagram. For a detailed object description, refer to the add DMS discussion.

Formatted: Bullets and Numbering

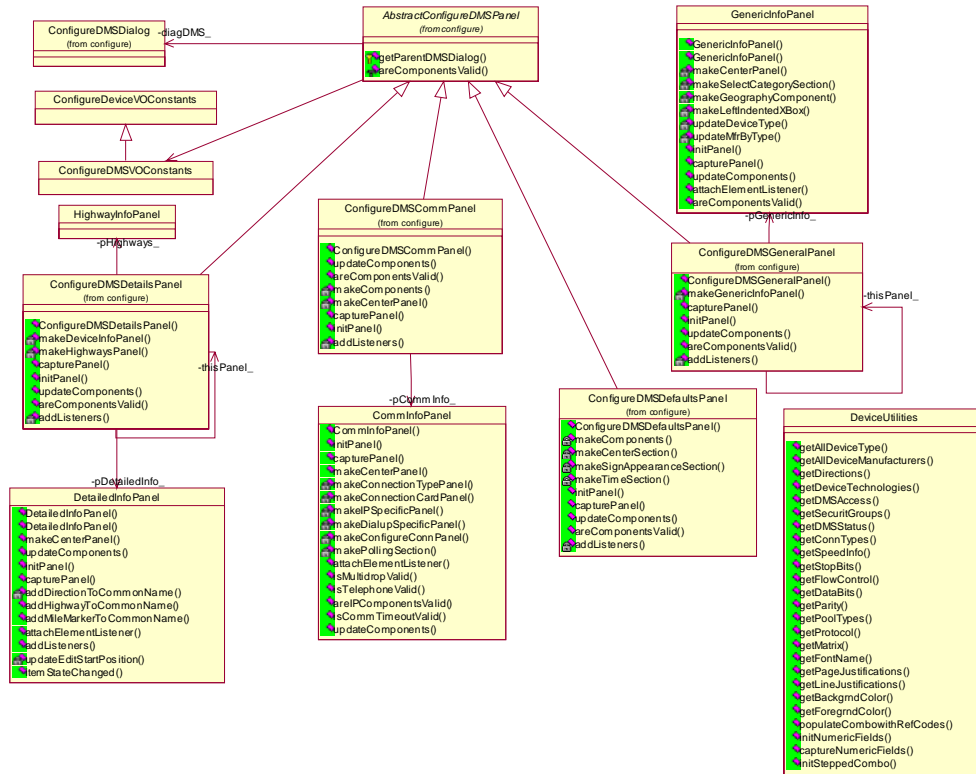


Figure 31: Configure DMS Class Diagram

3.10 User Management

An important component of the CTMS system is the ability to manage users. The client UI must facilitate the adding, viewing, and editing of users of the system.

3.10.1 User Management Class Diagram

The following class diagram details the classes involved in managing the system users. This collection of classes is responsible for managing all the dialogs and other presentation elements that relate to adding, editing, and viewing the set of the users for the system.

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

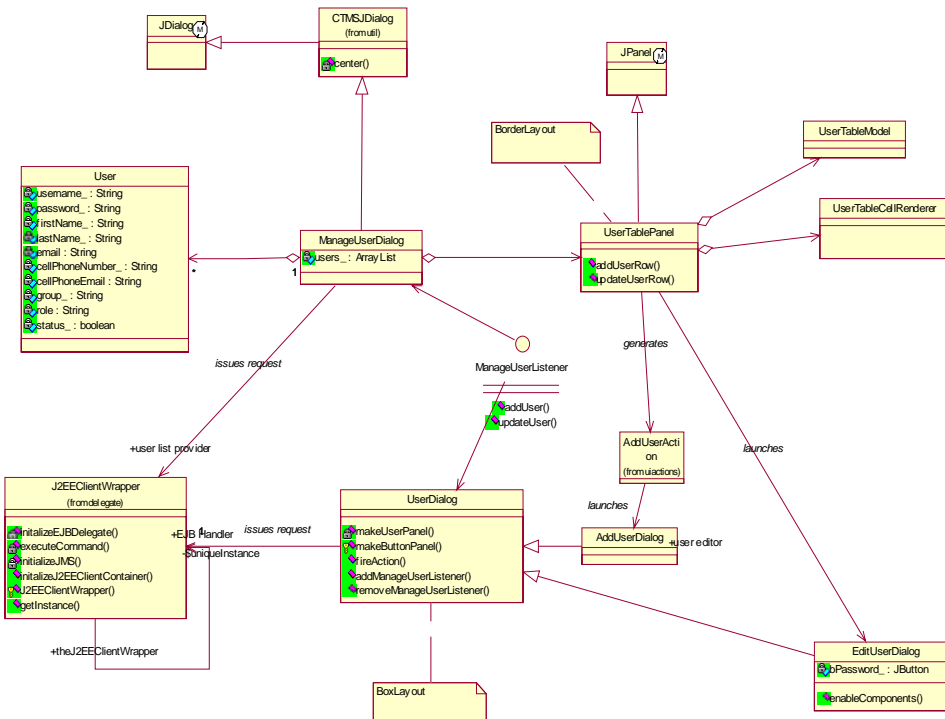


Figure 32: User Management Class Diagram

3.10.1.1 ManageUserDialog

The primary class for the user management is the ManageUserDialog. This dialog is launched via the ManageUserAction (not shown) that is only available through the menu of the MapNavigator. The dialog is composed of a users table panel, UserTablePanel and a collection users. The dialog will use the J2EEClientWrapper to obtain a set of users for the system. It will also access it to add and update users. The dialog is specialized version of a custom JDialog that will capture common behaviors across all dialogs of the client UI.

Formatted: Bullets and Numbering

3.10.1.2 User

This class defines a user of the system. Its values are used to populate the table.

Formatted: Bullets and Numbering

3.10.1.3 UserTablePanel

The UserTablePanel will be composed of the table, the table models, and the table renderers.

Formatted: Bullets and Numbering

3.10.1.4 UserDialog

The user dialog will be base for the specialized dialogs for adding and editing users. This dialog will facilitate the addition and removal of ManageUserListeners so that user manipulations can be fired to all the listeners. The system will be implemented with one listener, ManageUserDialog.

Formatted: Bullets and Numbering

3.10.2 User Management Sequence Diagram

Formatted: Bullets and Numbering

The following diagram details the sequence of events that occur when a manage user action occurs. The manageUserAction will create and launch the ManageUserDialog that will contact the J2EEClientWrapper that is the proxy to the RemoteControllerProxy. The proxy will provide the set of users.

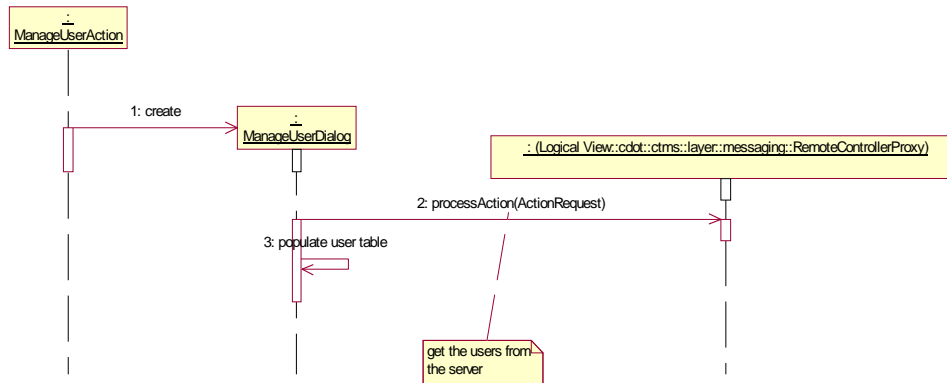


Figure 33: User Management Sequence Diagram

3.10.3 Add User Sequence Diagram

Formatted: Bullets and Numbering

The following diagram details the sequence of events that occur when an add user action occurs. The add user action will create and launch another AddUserDialog. It will also attach itself as a listener to the changes of the Add User Dialog. A cancel action on the AddUserDialog will merely close the Dialog. A submit action will encapsulate all the information entered by the user in a user object. The information is send to the ManageUserDialog to update the table model and a copy of the object is sent to the RemoteControllerProxy to be saved in the database.

The sequence of events for an edit user action is very similar. The only difference is that the textfields of the EditUserDialog will be pre-populated with the details of the user chosen for editing. The textfields for the username and the password will be uneditable.

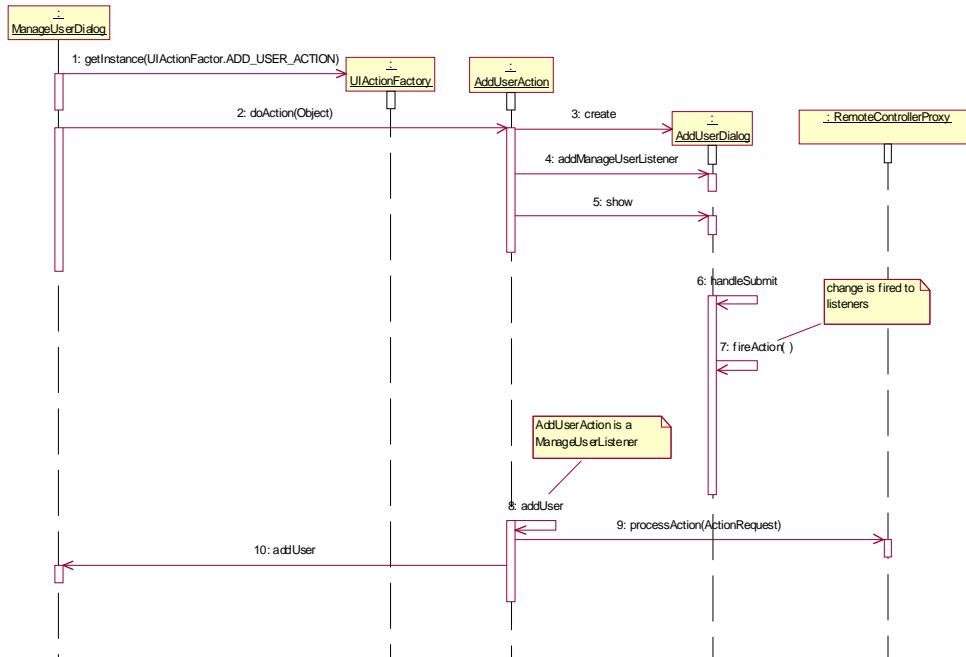


Figure 34: Add User Sequence Diagram

3.11 Communications Management

An important component of the CTMS application is the management of communication. Communication parameters are associated to a logical port which maps to a physical port. Pool is a collection of ports. Managing ports and pools include activities such as adding, editing and/or deleting ports/pools. The framework for communications management is similar to the framework for User management. The list of ports/pools is obtained from the server side. A listener framework orchestrates the launching of Add/Edit Dialog. Finally, Add/Edit actions are responsible for prompting the server to add/edit the new values. Sections 3.10.1 and 3.10.2 describe in detail the class diagram for Managing communications pool and managing communications port.

3.11.1 Communications Pool

This section details the class architecture for Managing Communications Pool. All classes involved are described below the class diagram.

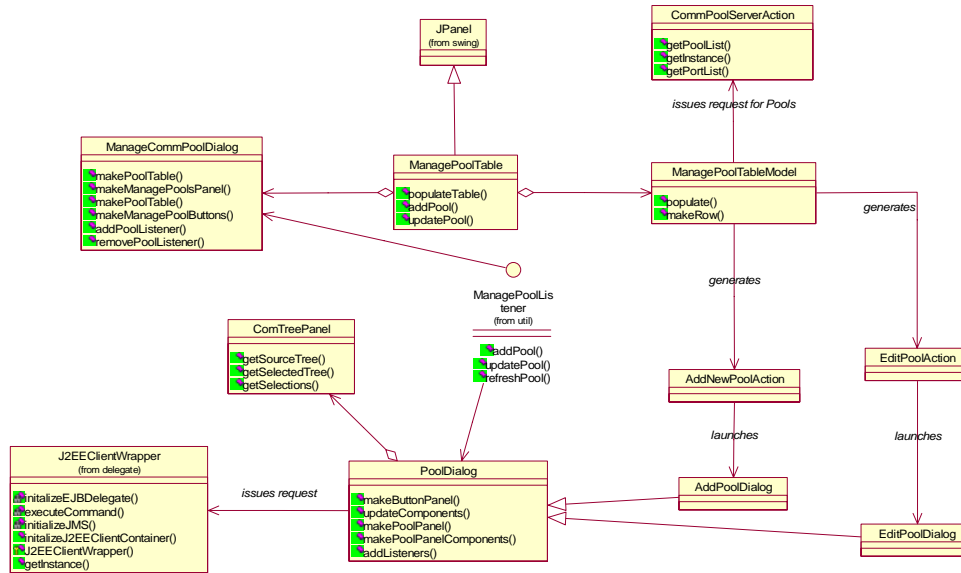


Figure 35: Comm Pool Class Diagram

3.11.1.1 ManageCommPoolDialog

This is the primary class for comm. Pools. This dialog displays all existing pools and is launched via menu items “Communication Pool” of Map Navigator frame. The dialog comprises of table, tablemodel for cell rendering and buttons for executing the add pool, edit pool and cancel actions.

3.11.1.2 ManagePoolTable

This class generates the table that displays the pools. The table is responsible for populating itself each time the Manage Comm Pool dialog is launched. The list of pools are obtained from CommPoolServerAction each time the table is created.

3.11.1.3 PoolDialog

PoolDialog is the base class that is extended by AddPool Dialog and EditPool Dialog. The dialog is responsible for creating all common UI components (textfields, combo boxes, table etc). Pool dialog uses a reusable common component generated by CommTreePanel. This panel handles two trees that display ports within each connection type. The component provided buttons for adding/removing ports from a pool.

3.11.1.4 AddPoolDialog

AddPool Dialog is responsible for creating of UI components required to capture pool related information. Each time a user executes the add new pool action from the ManagePools Dialog, the AddPoolDialog is launched.

3.11.1.5 EditPoolDialog

EditPoolDialog is responsible for creating UI components unique to edit pool dialog screen. EditPool dialog can be launched only from ManagePoolDialog. Double clicking a row from ManagePoolDialog launches the edit pool dialog with information related to the selected pool.

CTMS/CTIS	Version: 1.6
GUI Detailed Design	Date: 03-Dec-04

3.11.1.6 CommPoolServerAction

CommPoolServer action interacts with the RemoteControllerProxy of the J2EEClientWrapper to obtain a list of all exiting pools.

3.11.1.7 ManagePoolListener

ManagePoolListener interface provides methods to add pool, edit pool and refresh pools.

3.11.2 Communications Port

This section details the class architecture for managing communications port. The user management framework is reused here.

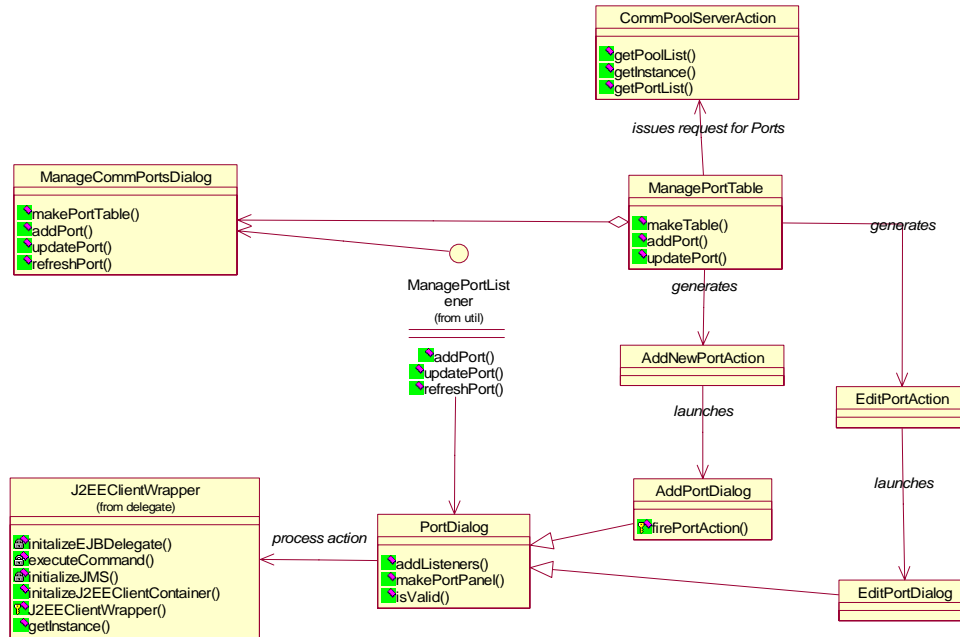


Figure 36: Comm Port Class Diagram

3.11.2.1 ManageCommPortsDialog

This is the main class for managing comm. Ports. This class generates a dialog with table (for displaying all ports) and buttons to execute the add and edit port actions. ManagePortTable is responsible for obtaining all existing ports through the CommPoolServerAction and further populating the table.

3.11.2.2 PortDialog

PortDialog is the base class for AddPortDialog and EditPortDialog. PortDialog is responsible for generating components (textfields, combo boxes etc) that are common to AddPortDialog and EditPortDialog.

CTMS/CTIS	Version: 1.6
GUI Detailed Design	Date: 03-Dec-04

3.11.2.3 AddPortDialog

AddPortDialog is responsible for creating the dialog and UI components required for capturing the new port information. Information entered is validated against the common validation framework that includes check for unique constraint, valid characters and length.

3.11.2.4 EditPortDialog

EditPortDialog is responsible for creating the dialog and UI components required for displaying/capturing the port information. Any information entered is validated using the validation framework containing rules specified in the Use Case.

3.12 CTMS Help System

This section describes the architecture for the comprehensive Help System for CTMS. There are various Help Authoring tools available ex. RoboHelp, MS Help, JavaHelp and others. After careful analysis, the most cost-effective and comprehensive package was determined to be Oracle Help for Java (OHJ). The OHJ package offers a set of components from Oracle corp. built upon the technology of Java Help. OHJ API's are free for use and distribution. CTMS will use OHJ 4.2.7 (released May 2004) as its primary choice for a Help System.

OHJ offers a set of UI components (Search Window, Navigator Window, docking framework etc) and API's to leverage/customize the components. The CTMS Help System uses three main OHJ objects – Help(a default implementation), HelpSet(a robust implementation of the Book interface, this object recognizes the xml files containing Help information) and CSHManager(object used to handle context sensitive help for UI components).

The Help architecture adheres to the following OHJ specific coding standards.

1. Help should be created in a standalone mode, such that when Help System exits the JVM, the entire application does not shutdown.
2. A singleton instance of HelpSystem (CSHManager and Help object) should be maintained.
3. Programmatically displaying a Help topic should involve disposing any old Help objects and creation of new Help object with specified Help Set.
4. Using Help from a modal window should at first register the Window with the Help System and later unregister when the dialog is closed. This eliminates any unwanted references and the window can be garbage collected.
5. All Help topics should be stored as constants.

3.12.1 Help System Class Diagrams

This section will detail the class architecture for the help system.

3.12.1.1 HelpSystem.

The protected methods of interest are:

1. getInstance(): returns a singleton instance of the Help System
2. registerModalWindowtoHelp(Window): registers the Modal window to the Help system
3. unregisterModalWindowtoHelp(Window): unregisters the Modal window from Help System. This method should be called only after the window is closed
4. addComponenttoHelp(component,topic): this method registers the specified component to the Help System. The topic string passed in is the topic associated with the HTML. This method enables “Help” popup on the component that display the help.
5. displayHelpTopic(topic): displays the Help System with the specified topic.

CTMS/CTIS	Version: 1.6
GUI Detailed Design	Date: 03-Dec-04

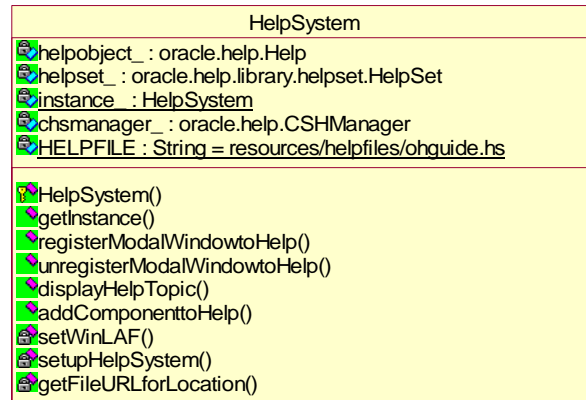


Figure 37: Help System class diagram

3.12.1.2 IHelpConnector

IHelpConnector is the interface that every component (dialog,frame) requiring Help should implement. It has the getHelpKey() method that returns the Help Key (string) specific to the component. This approach is different from the constants approach and is useful in decoupling the Help object from the keys required to use Help.



Figure 38: IHelpConnector

3.12.1.3 HelpUtilities

HelpUtilities provides a layer of abstraction for the CTMS application. Thus, any part of the product using help does not need to know anything about HelpSystem or the core implementation. The public methods of interest are as follows:

1. displayHelpforTopic(String topic): displays the Help Navigator with the html file corresponding to the Help topic passed in.
2. addTopic(IHelpConnector): add component to the CSHManager.
3. addTopictoComponent(IHelpConnector ,Component): This method is used to add a component (such as button) to the Help system.
4. addModalTopic(Window,IHelpConnector): This method is used to register the modal window to help, attaché the specified window for context help and finally unregister window from Help.

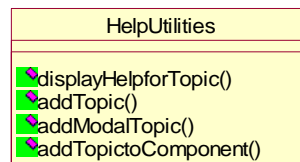


Figure 39: HelpUtilities

5.

3.12.2 Help System Sequence Diagrams

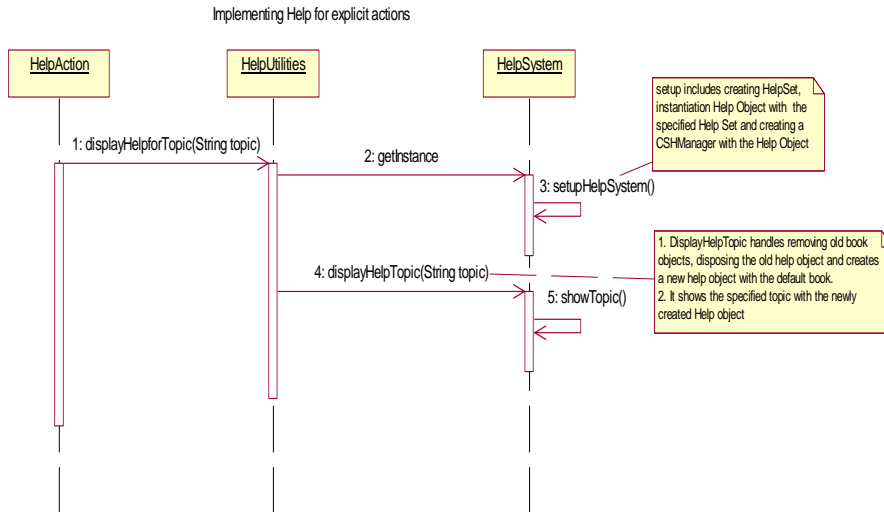


Figure 40: Sequence Diagram for Help Action

This diagram elaborates the sequence of steps required to implement Help for an explicit action (ex. Help button). The only action required from the application standpoint is to call `displayHelpforTopic()` method of `HelpUtilities` passing in the topic (string).

Behind the scenes Help gets an instance of `HelpSystem` by using `getInstance()`. The constructor of the Help System is responsible for setting up the Help object with default specifications (ex. Help Sets, HTML files to be used).

`HelpUtilities` calls `displayHelpTopic(String helptopic)`. This method displays the actual Help Navigator with the specified topic. ***Topic keys will be stored in resource file and available throughout the system at all times.***

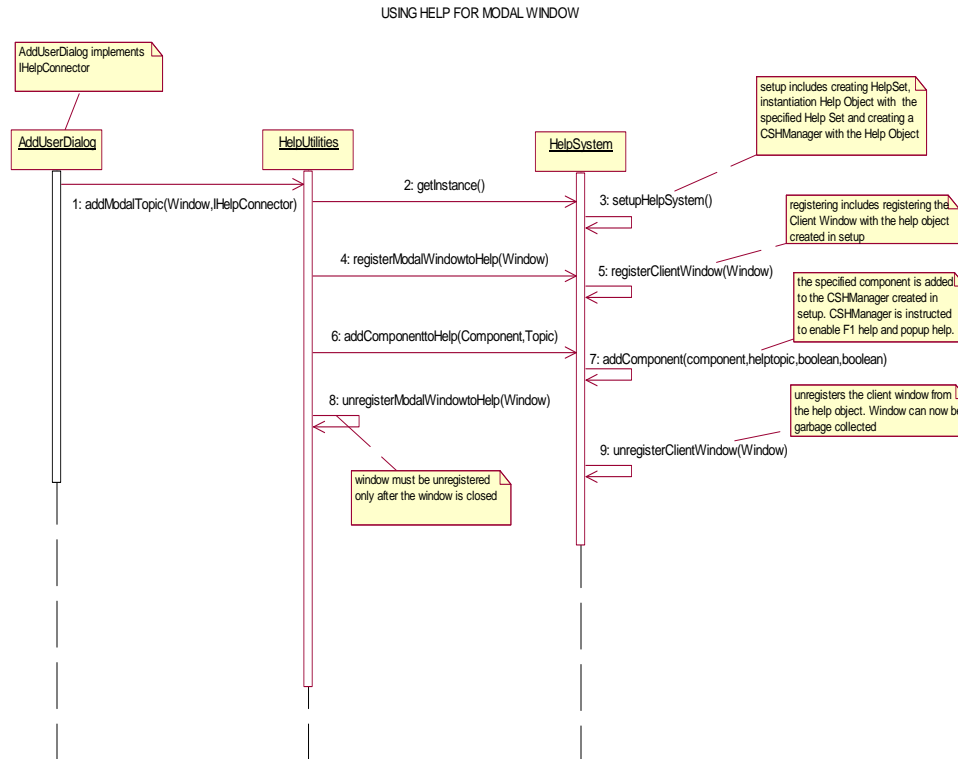


Figure 41: Sequence Diagram for Context Sensitive Help

This diagram elaborates the use of Help System for a modal window. The dialog adds a topic to the help system and the help system creates all the dependent components.

3.13 UI Utilities

The client will be composed of UI elements that will use a variety of UI utilities. These utilities will reside in the `cdot.ctms.client.layer.ui.util` package. Behaviors will include centering, mouse cursor management, image manipulation, icon management, and layout configuration. The classes defined in the util package may not refer to objects residing in other packages; however, utility classes may be used by one or more external packages. The utilities will be described in two class diagrams; one describing the generic classes and another describing the UI component factory.

3.13.1 Generic UI Utilities Class Diagram

The generic utilities will be composed of two primary classes, `CDOTUIUtilities` and `CTMSUIUtilities`. These utilities will offer a collection of static methods that form the building blocks for creating a consistent UI.

CTMS/CTIS	Version: 1.6
GUI Detailed Design	Date: 03-Dec-04

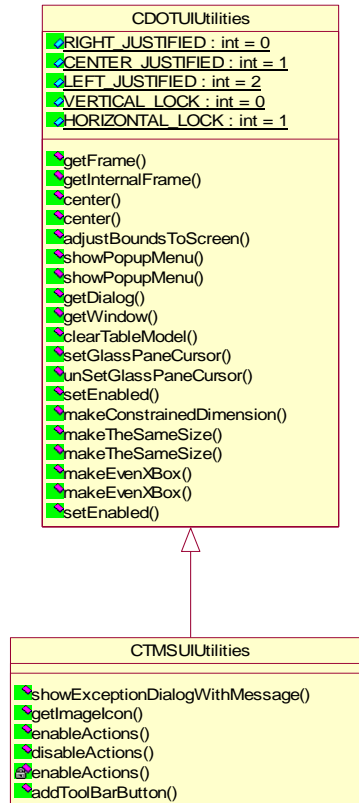


Figure 42: Generic UI Utilities Class Diagram

3.13.1.1 CDOTUIUtilities

This class defines attributes and behaviors that are common to UI's, but do not specifically relate to the CTMS system. These utilities may be used by other projects. Specific behaviors include making components the same size, making even layout managers, enabling a group of components, centering, and locating parents given a child component.

3.13.1.2 CTMSUIUtilities

This class defines attributes and behaviors that specialize the CDOTUIUtilities to make utilities that are more specific to the CTMS system.

3.13.2 UI Component Factory Class Diagram

This particular section describes the use of a factory pattern to coordinate the construction of UI components used for various GUI frameworks. Using a factory pattern helps establish a standard way of generating UI components. Further, it

also abstracts the details of each component while exposing all the required methods for control over look and feel of an individual component. The following classes share the main class as part of this framework is the CDOTUIFactory.

3.13.2.1 CDOTUIFactory

The CDOTUIFactory class delegates responsibility for construction of individual UI component. This class will provide a set of static methods for returning the requested components. Ex. makeJTextArea() returns a JTextArea component. Actual construction of the component is the responsibility of CDOTJTextArea class. Having such a framework helps isolate the UI components. Also, any construction related details for any component can be controlled from one single location.

3.13.2.2 UI Components

This section describes the general behavior of the classes that construct the UI component. Depending upon the standards set for the component, they may either extend a generic swing component or a custom component offered by JSuite. Actions for each component will be handled by the UI action object.

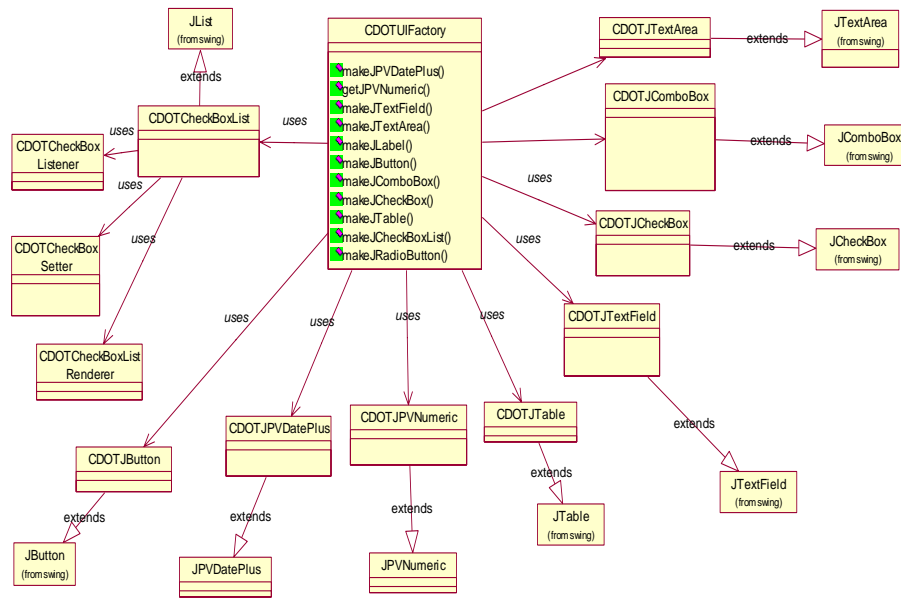


Figure 43: UI Component Factory Class Diagram

CTMS/CTIS	Version: 1.6
GUI Detailed Design	Date: 03-Dec-04

Appendices

The following appendices will provide a more in-depth look at some of the design elements that were referred to in the main sections of this document.

Appendix A: Detailed Description of Presentation

The CTMS client UI will be composed of several Swing-based interfaces that are presented to the user. The collection of views may be menus, frames (large, closable windows), dialogs (modal screens that are children of a parent frame), or internal frames (free floating frames within a parent frame.) These include Login, Splash Screen, Map Navigator frame, Map Navigator window, Manage Users dialog, Edit Users dialog, Add Users dialog, Change Password dialog, Select DMS dialog, View DMS dialog, Activate Message dialog, Status Console frame, Status Console Menu, Logging table, Instruction Queue table, and alarms table. This section will present the description for each of the screens in the CTMS client UI. In most cases, the descriptions are accompanied by rough mock-ups of the actual screens. These wireframes are intended to provide a general idea of the finished screen not the actual finished screen.

Login and Splash Screen

The following figures represents the opening splash screen and the login screen. An image will be displayed in the window with a status updating along the bottom. The login window will be a minimal window that allows the user to login. The Splash Screen will use a BorderLayout. The login window is supplied by JBoss.

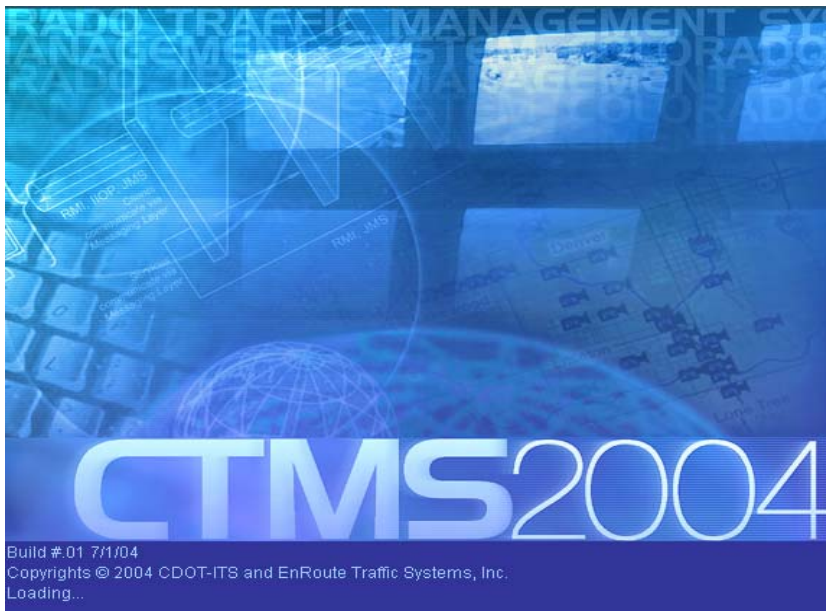


Figure 44: Splash Screen

CTMS/CTIS	Version: 1.6
GUI Detailed Design	Date: 03-Dec-04

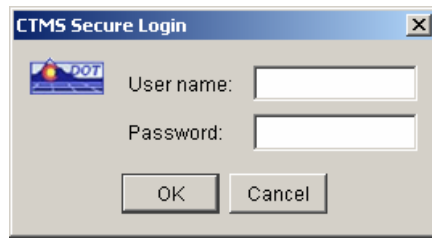


Figure 45: Login Screen

Map Navigator

The following screen details the sketch for the primary map screen that is in the CTMS product. The window is a split pane and will use a BorderLayout. The user will be able to scroll within the two table of contents windows in either direction. The user will also be able to adjust the screen size and split panel location between the map and the left components and between each item on the left section.

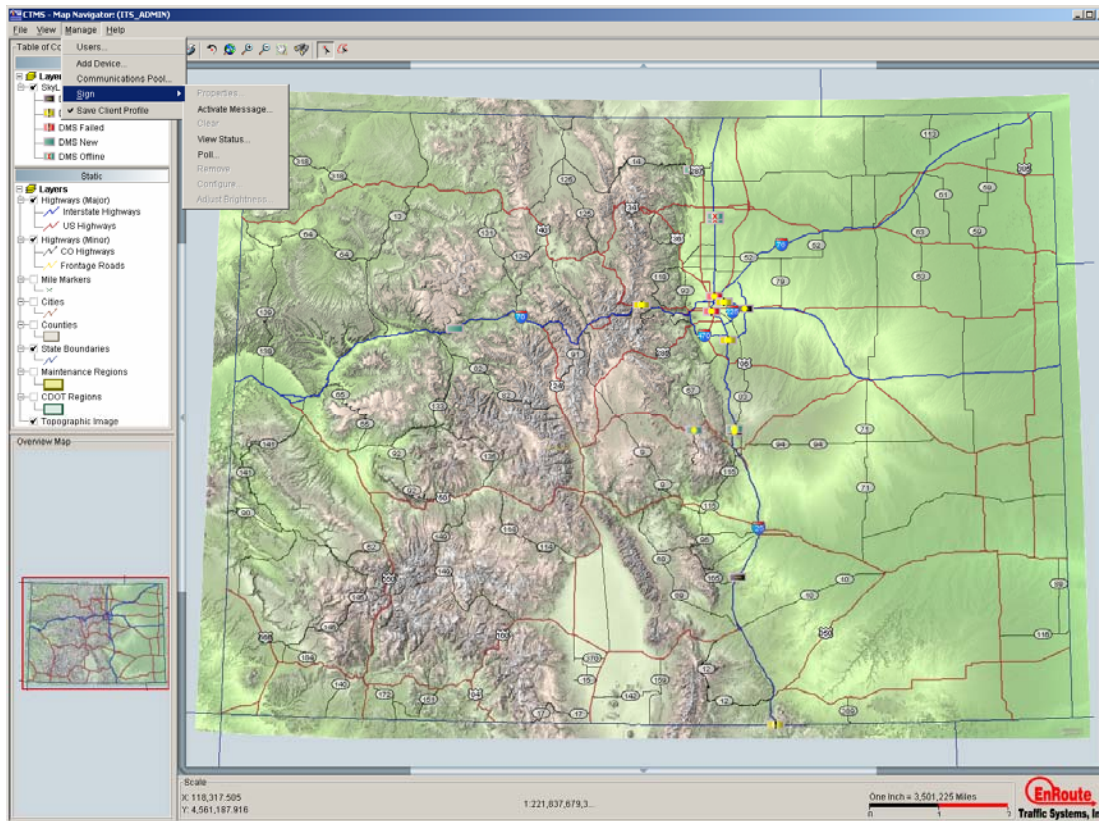


Figure 46: Map Frame

CTMS/CTIS	Version: 1.6
GUI Detailed Design	Date: 03-Dec-04

Manage Users

The following screen provides a sketch of the user management window. The dialog will consist of the BorderLayout panel with a table in the center and buttons in the south window region. The buttons will include Add User, Edit User, and Cancel.

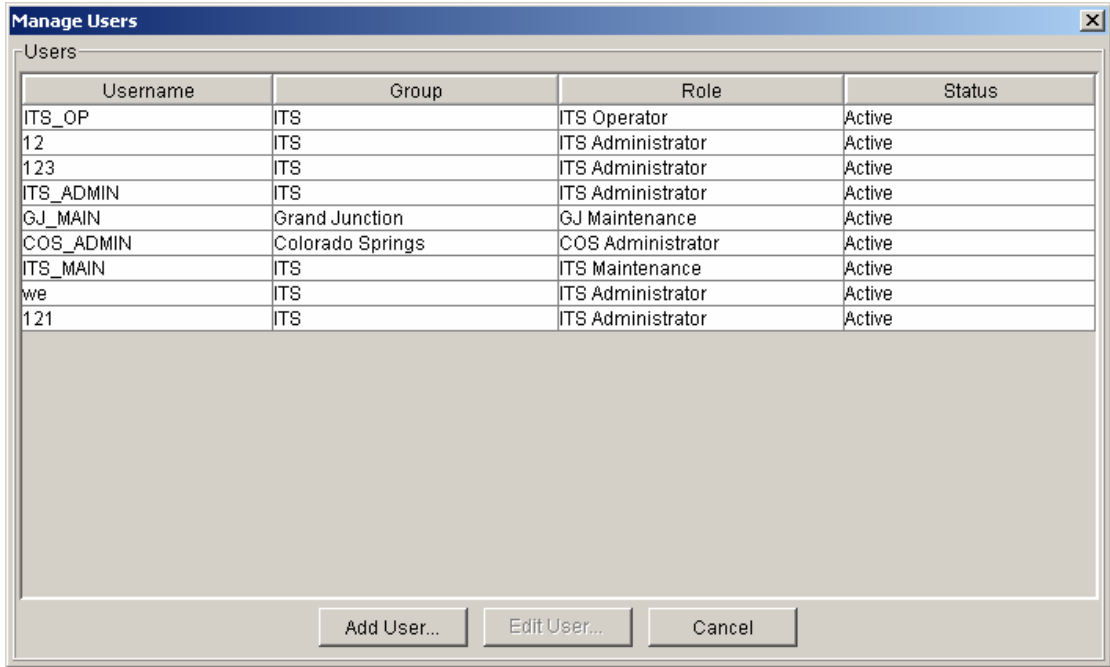


Figure 47: Manage Users Screen

Edit Users

Some users will be able to edit users. A screen will be presented that allows the user to modify everything about the user except for the users name. This screen will use a BoxLayout.

Edit User

User Description

Username: 121

Last Login:

Password: [masked] ...

Status: Active

* First Name: 12

Middle Name:

* Last Name: 12

* Email Address: 12@12.1@

Cell Phone: () -

Cell Phone Email:

* Organization: CDOT

Permissions

Group	Role
ITS	ITS Administrator

* Add... Remove Edit

OK Cancel

Figure 48: Edit Users Screen

CTMS/CTIS	Version: 1.6
GUI Detailed Design	Date: 03-Dec-04

Add Users

The add users screen is essentially the same screen as the edit users screen, but with the user name available for edit. This screen will be shared with the Edit Users Dialog.

Figure 49: Add New User Screen

View DMS

The Map Navigator will support the selection of multiple DMS'. Upon releasing the zoom rubberband, the following dialog will pop-up. This dialog will use a BorderLayout.

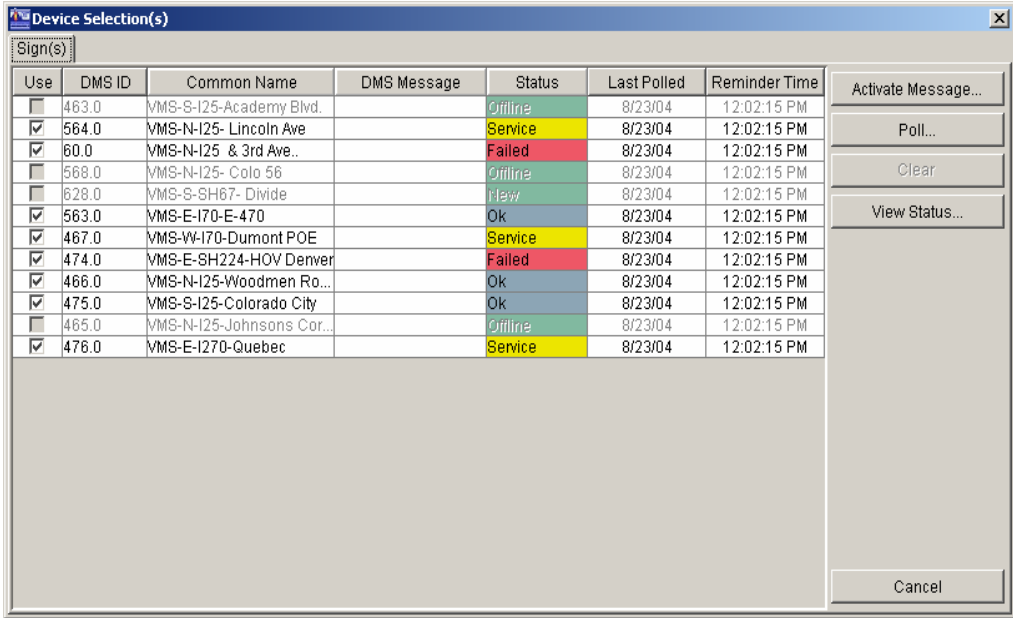


Figure 50: View DMS Dialog

CTMS/CTIS	Version: 1.6
GUI Detailed Design	Date: 03-Dec-04

Activate DMS Message

The following diagram provides an idea of how the activate message dialog will look. This dialog will use a GridBagLayout..

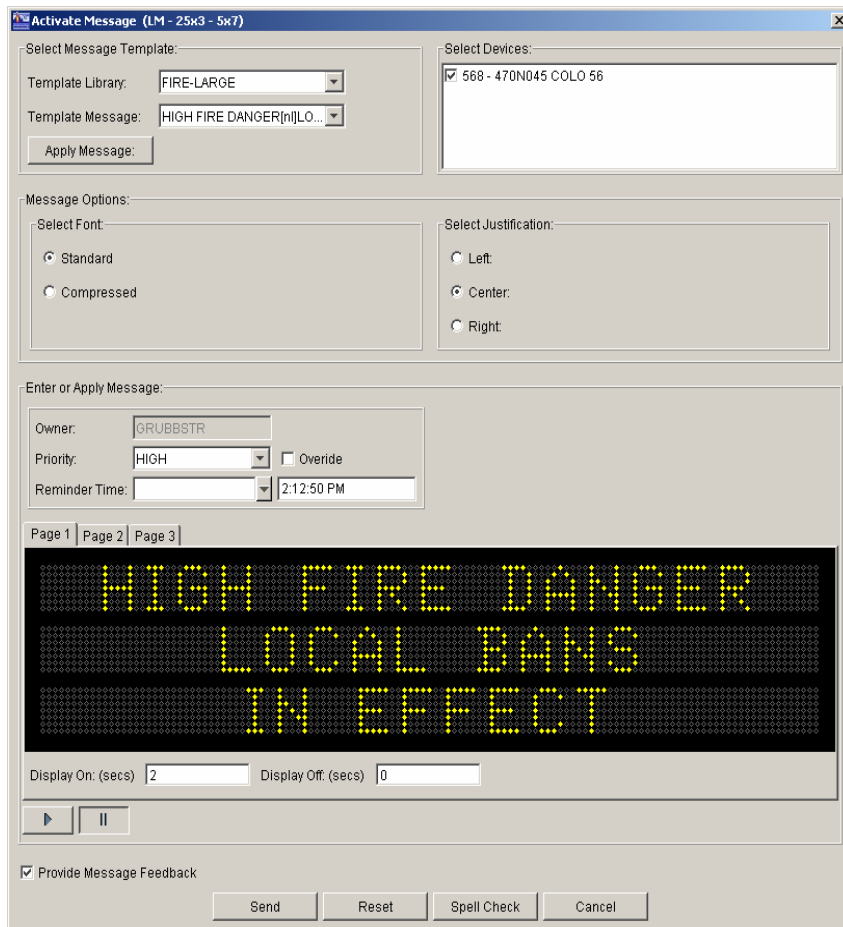


Figure 51: Activate DMS Message Dialog

CTMS/CTIS	Version: 1.6
GUI Detailed Design	Date: 03-Dec-04

View Sign Status

This dialog will be shown when the user selects to view a DMS or polls the status. This dialog will use a GridBagLayout.

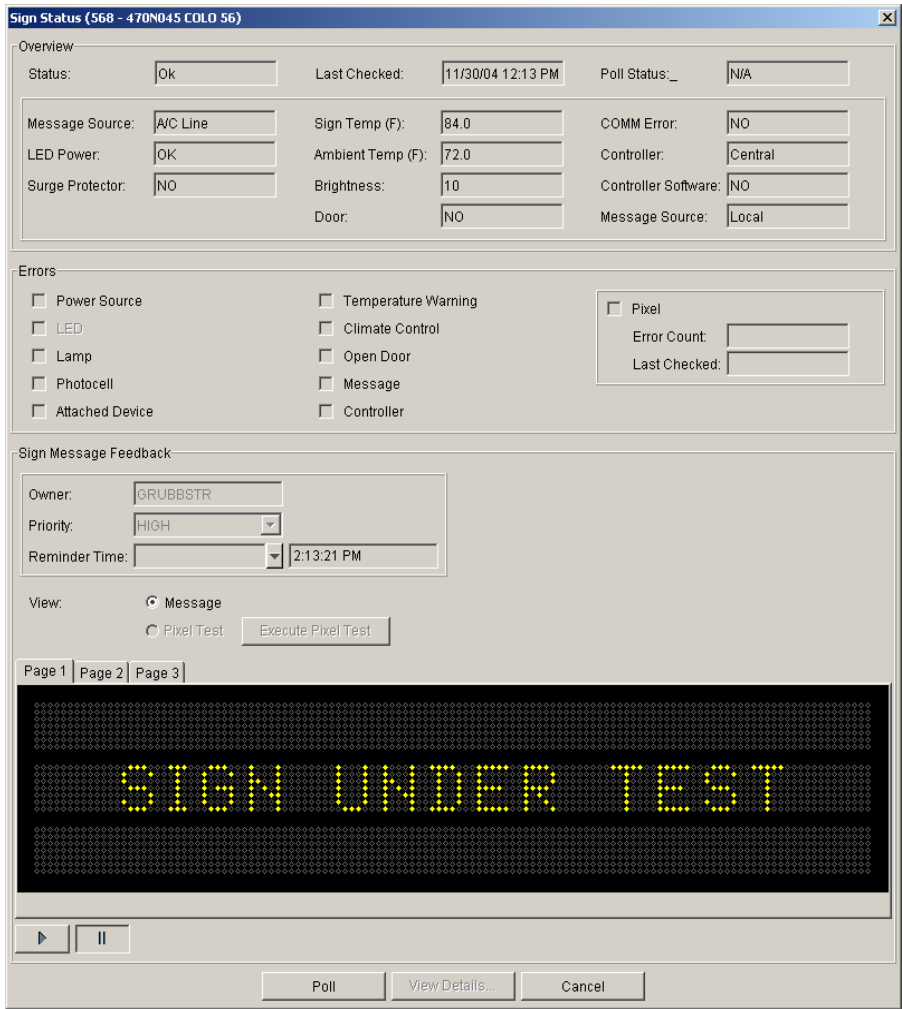


Figure 52: Sign Status Dialog

CTMS/CTIS	Version: 1.6
GUI Detailed Design	Date: 03-Dec-04

View Errors

The View Details button of the Sign Status Window displays all errors occurred during a poll operation.

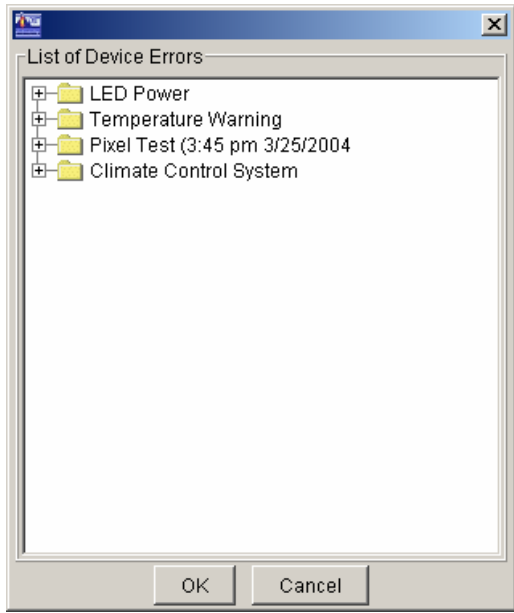


Figure 53: View Details Window

Adjust Brightness

Users can view/ adjust brightness of a particular DMS through the Adjust Brightness window.

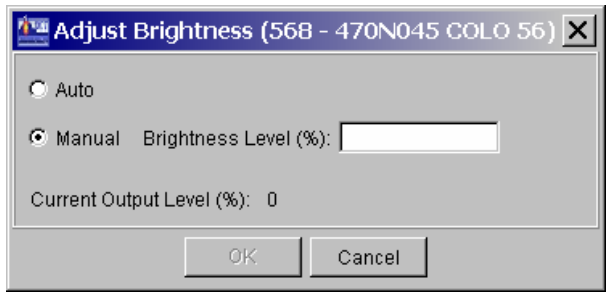


Figure 54: Adjust Brightness

CTMS/CTIS	Version: 1.6
GUI Detailed Design	Date: 03-Dec-04

Clear Sign

Users can clear the sign on DMS(s) through the Clear Sign Dialog. This same dialog displays single or multiple selected DMS.

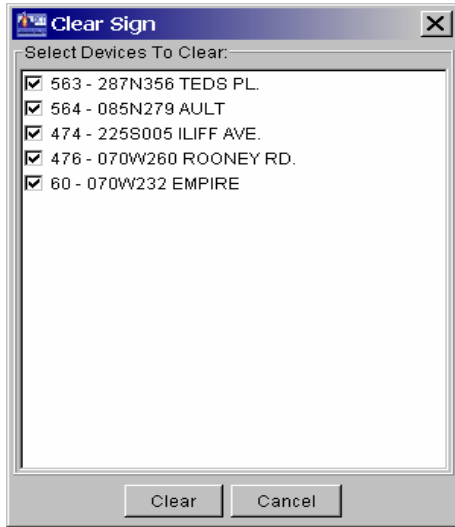


Figure 55: Clear Sign

Generate Fax Log Report

Users can generate one or more fax log reports through the Generate Fax Log Dialog. This same dialog displays single or multiple reports.

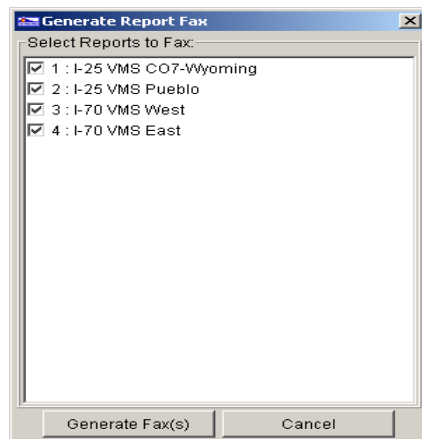


Figure 56: Generate Report Fax

Status Console

The status console is primary frame container that will consist of menu and an open desktop. The desktop will contain up to three free- floating frames; Alarms, Logging, and Instruction Queue.

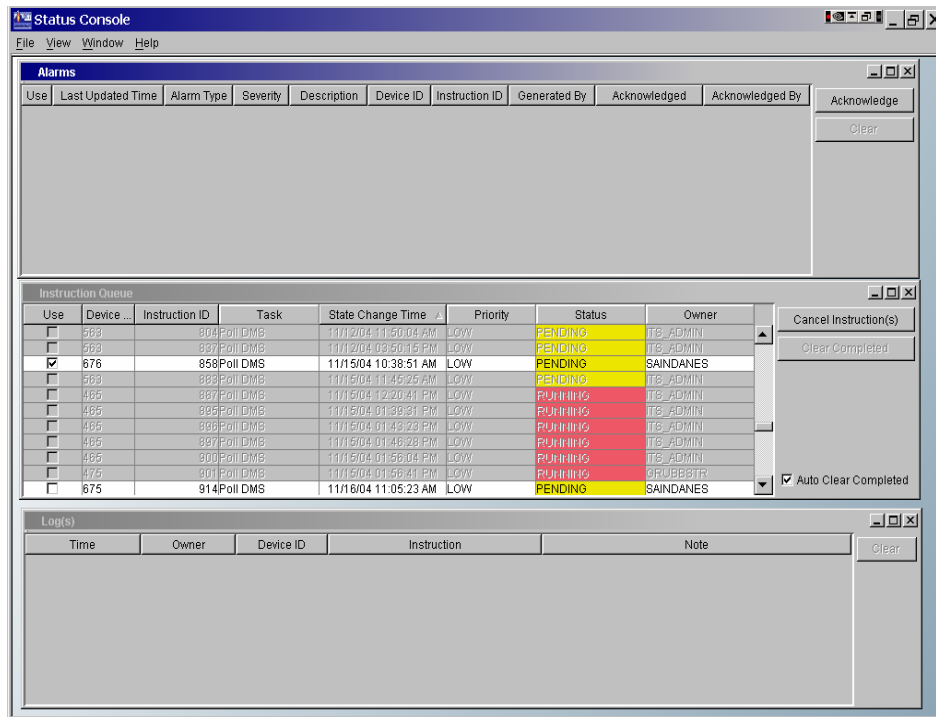


Figure 57: Status Console

Alarm Window

The alarms window will display a set of critical errors (unacknowledged) that have occurred in the system. A BorderLayout will be used. NOTE: Alarms are no longer a part of Iteration One.

CTMS/CTIS	Version: 1.6
GUI Detailed Design	Date: 03-Dec-04

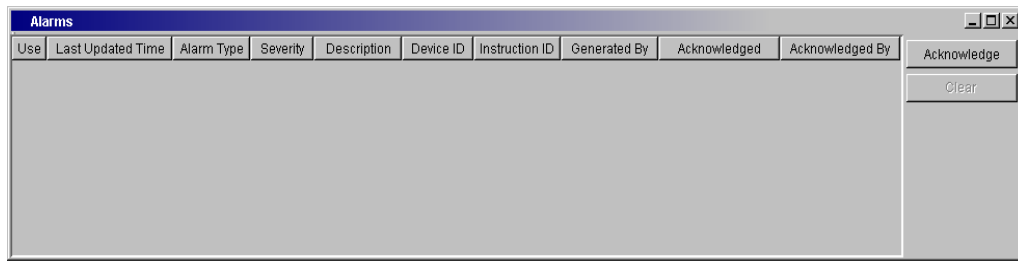


Figure 58: Alarms Window

Logging Window

The logging window of the status console will present a list of log entries in a table that pertain to errors that occurred during the current client session. A BorderLayout will be used.

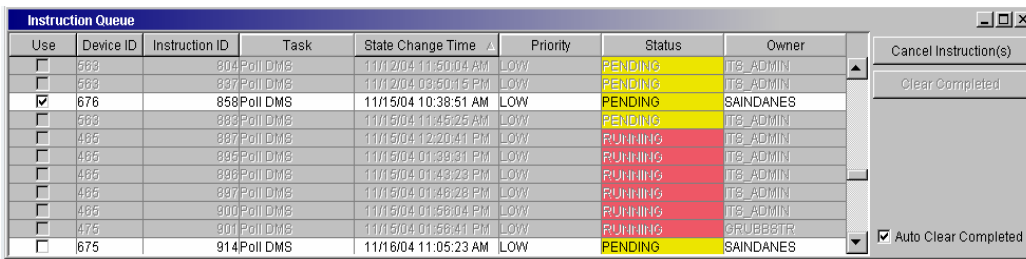


Figure 59: Logging Window

Instruction Queue Window

The instruction queue will present a table containing the set of pending/running instructions. A button will be provided that will allow the user to cancel a pending instruction. A BorderLayout will be used.

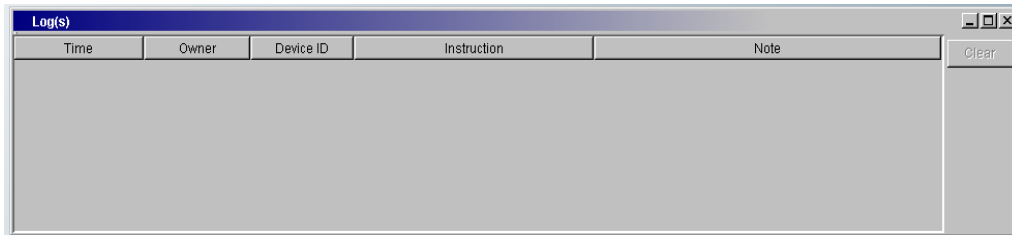


Figure 60: Instruction Queue Frame

CTMS/CTIS	Version: 1.6
GUI Detailed Design	Date: 03-Dec-04

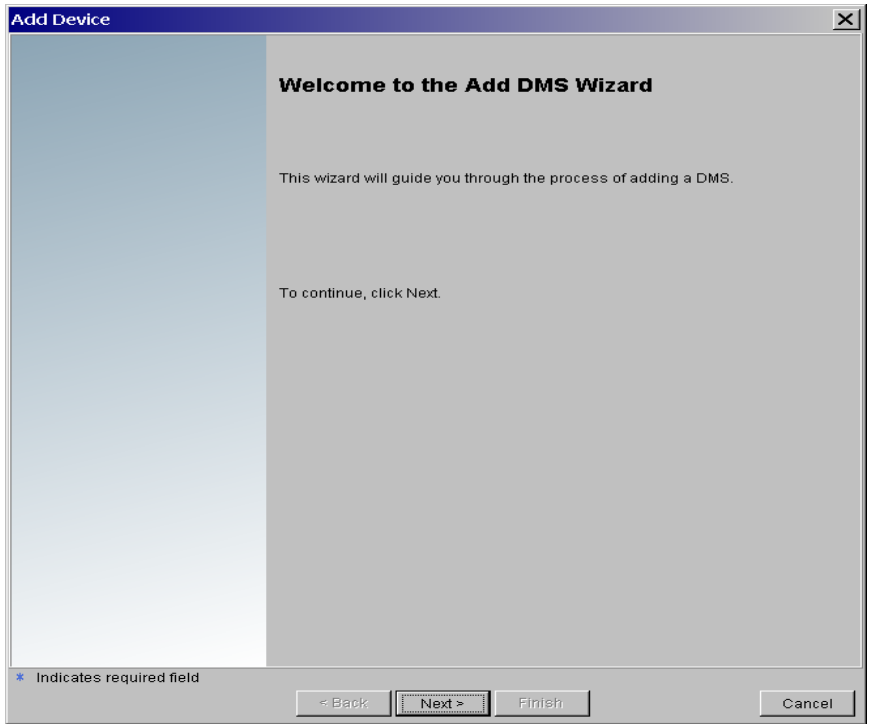


Figure 61: Add DMS Wizard - Introduction

CTMS/CTIS	Version: 1.6
GUI Detailed Design	Date: 03-Dec-04

Add Device

High-Level Device Description.

Please enter the type and location for the device.

Select Device Category:

- * Device Type:
- * Device Manufacturer:

Enter Geographical Location in UTM/13 NAD 83 format:

- * X:
- * Y:
- * Z:

* Indicates required field

< Back **Next >** Finish Cancel

Figure 62: Add DMS Wizard - Category Page

Add Device

Device Description
Please enter the device details for the DMS.

Enter Device Information:

MAT ID:

* Common Name: ...

* Security Group:

Status:

Bring Online

Notes:

* Direction:

* DMS Technology:

* Access:

* Type:

Enter Highway Information:

* Highway:

* Mile Marker:

Select Secondary Highways:

Related	Highway ID	Highway Name
<input type="checkbox"/>	1	C 470
<input type="checkbox"/>	2	CO 1
<input type="checkbox"/>	3	CO 10
<input type="checkbox"/>	4	CO 103
<input type="checkbox"/>	5	CO 105
<input type="checkbox"/>	6	CO 119
<input type="checkbox"/>	7	CO 125
<input type="checkbox"/>	8	CO 14

* Indicates required field

< Back Next > Finish Cancel

Figure 63: Add DMS Wizard - Device Info Page

Add Device

Communications Setup

Please enter the communications parameters for the DMS Device.

Select and Configure Connection Type:

* Connection Type:

Enter Dialup/Fiber Details:

Speed (Baud Rate):

Stop Bits:

Flow Control:

Data Bits:

Parity:

* Telephone Number: () -

Callback Number: () -

Configure Connection Parameters:

Pool Name:

Protocol:

* Multidrop Address:

* Communications Timeout Duration:

Polling Description

Polling Enabled

Polling Frequency:

* Indicates required field

Figure 64: Add DMS Wizard - Communication Page for Dialup

Add Device
Communications Setup
Please enter the communications parameters for the DMS Device.

Select and Configure Connection Type:
* Connection Type: **Fiber**

Enter Dialup/Fiber Details:
Speed (Baud Rate): 9600
Stop Bits: 1
Flow Control: None
Data Bits: 8
Parity: None

Configure Connection Parameters:
Pool Name: testpool3
Protocol: PMPP
* Multidrop Address: 1
* Communications Timeout Duration: 1

Polling Description
 Polling Enabled
Polling Frequency:

* Indicates required field

< Back Next > Finish Cancel

Figure 65: Add DMS Wizard - Communication Page for Fiber

Add Device

Communications Setup

Please enter the communications parameters for the DMS Device.

Select and Configure Connection Type:

* Connection Type: IP

Enter IP Details:

* IP Address:

* Port Number:

Configure Connection Parameters:

Pool Name: IP Pool 2 Configure Pool...

Protocol: PMPP

* Multidrop Address: 1

* Communications Timeout Duration: 1

Polling Description

Polling Enabled

Polling Frequency:

* Indicates required field

< Back
Next >
Finish
Cancel

Figure 66: Add DMS Wizard - Communication Page for IP

CTMS/CTIS	Version: 1.6
GUI Detailed Design	Date: 03-Dec-04

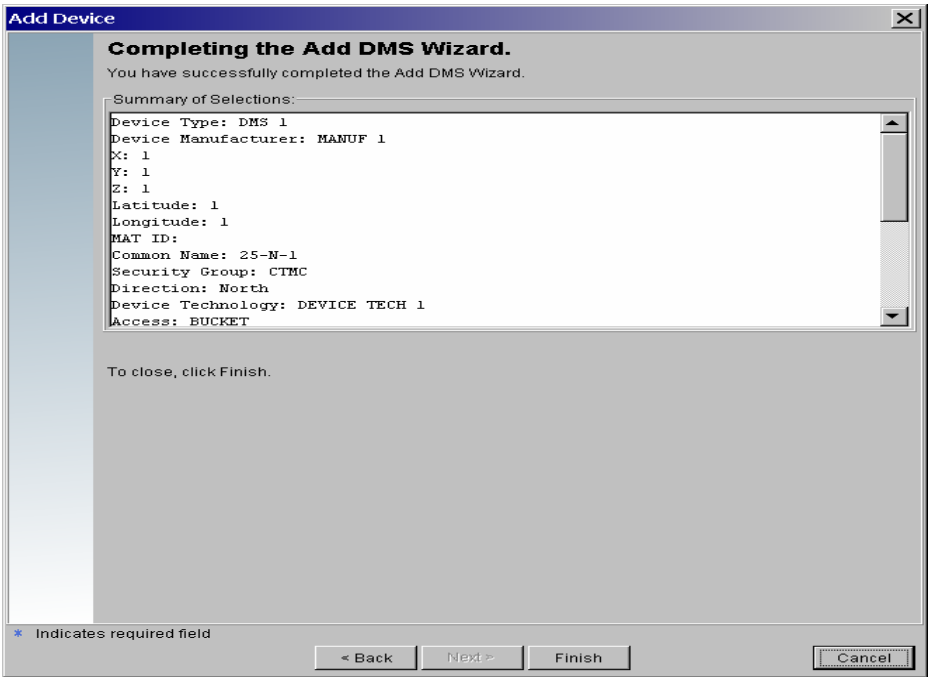


Figure 67: Add DMS Wizard - Summary Page

Configure Sign (676 - 470N044 SPRINGS DEVICE) [X]

General | Details | Dimensions | Equipment | Communications | One-Wire | Multi-Defaults

Generic Device Description

Select Device Category:

* Device Type:

* Device Manufacturer:

Enter Geographical Location in UTM/13 NAD 83 format:

* X:

* Y:

* Z:

* Indicates required field

Figure 68: Configure DMS - General Page

Configure Sign (676 - 470N044 SPRINGS DEVICE)

General | **Details** | Dimensions | Equipment | Communications | One-Wire | Multi-Defaults

Enter Device Information:

MAT ID:

* Direction: North

* Common Name: 470N044 SPRINGS DEVICE ...

* DMS Technology: Other

* Security Group: Colorado Springs

* Access: Bucket

Status: Ok

* Type: VMSCHAR

Bring Online

Notes:

Enter Highway Information:

* Highway: C 470

* Mile Marker: 44

Select Secondary Highways:

Related	Highway ID	Highway Name
<input checked="" type="checkbox"/>	1	C 470
<input type="checkbox"/>	2	CO 1
<input type="checkbox"/>	5	CO 105
<input checked="" type="checkbox"/>	7	CO 125
<input type="checkbox"/>	8	CO 14
<input type="checkbox"/>	9	CO 392
<input checked="" type="checkbox"/>	10	CO 402
<input type="checkbox"/>	11	CO 52
<input type="checkbox"/>	12	CO 56
<input type="checkbox"/>	13	CO 60
<input type="checkbox"/>	14	CO 66

* Indicates required field

Read From Sign OK Cancel

Figure 69: Configure DMS - Details Page

Edit Common Name

Highway (###): 470

Direction: N

Mile Marker (###): 0

Common Name:

OK Cancel

Figure 70: Edit Common Name

Configure Sign

General | Details | **Dimensions** | Equipment | Communications | One-Wire | Multi-Defaults

Describe Sign Dimensions

Matrix Type: CHARACTER

Legend Exists

Module Size

Columns:

Rows:

Physical Size

Height:

Width:

Sign Size

Lines:

Modules:

Pixel Pitch

Horizontal:

Vertical:

Border Size

Horizontal:

Vertical:

* Indicates required field

Read From Sign OK Cancel

Figure 71: Configure DMS Dimensions Page

Configure Sign (676 - 470N044 SPRINGS DEVICE) [X]

General | Details | Dimensions | **Equipment** | Communications | One-Wire | Multi-Defaults

Equipment Description

Photocell:	<input type="text" value="0"/>	Airflow Filters:	<input type="text" value="0"/>
Sign Temperature:	<input type="text" value="0"/>	Door Sensors:	<input type="text" value="0"/>
Ambient Temperature:	<input type="text" value="0"/>	Brightness Levels:	<input type="text" value="0"/>
LED Power Supplies:	<input type="text" value="0"/>	Photocell Level:	<input type="text" value="255"/>
Airflow Fans:	<input type="text" value="0"/>	Surge Power Levels:	<input type="text" value="0"/>
Airflow Sensors:	<input type="text" value="0"/>	Communication Levels:	<input type="text" value="0"/>

Pixel Service

Interval:	<input type="text" value="0"/>
Duration:	<input type="text" value="0"/>
Synchronize:	<input type="text" value="0"/>

* Indicates required field

Read From Sign OK Cancel

Figure 72: Configure DMS - Equipment Page

Configure Sign (676 - 470N044 SPRINGS DEVICE)

General | Details | Dimensions | Equipment | **Communications** | One-Wire | Multi-Defaults

Select and Configure Connection Type:

* Connection Type:

Enter Dialup/Fiber Details:

Speed (Baud Rate):

Stop Bits:

Flow Control:

Data Bits:

Parity:

* Telephone Number:

Callback Number:

Configure Connection Parameters:

Pool Name:

Protocol:

* Multidrop Address:

* Communications Timeout Duration:

Polling Description

Polling Enabled

Polling Frequency:

* Indicates required field

Figure 73: Configure DMS - Communication Page

Configure sign.

General | Details | Dimensions | Equipment | **Communications** | One-Wire | Multi-Defaults

Select and Configure Connection Type:

Connection Type: **SOCKET** ▼

Enter IP Details:

* IP Address:

* Port Number:

Configure Connection Parameters:

Pool Type: **POOL TYPE 1** ▼

Protocol: **PMPP** ▼

* Multidrop Address:

* Communications Timeout Duration:

Polling Description

Polling Enabled

Polling Frequency:

* Indicates required field

Figure 74: Configure DMS Communication Page

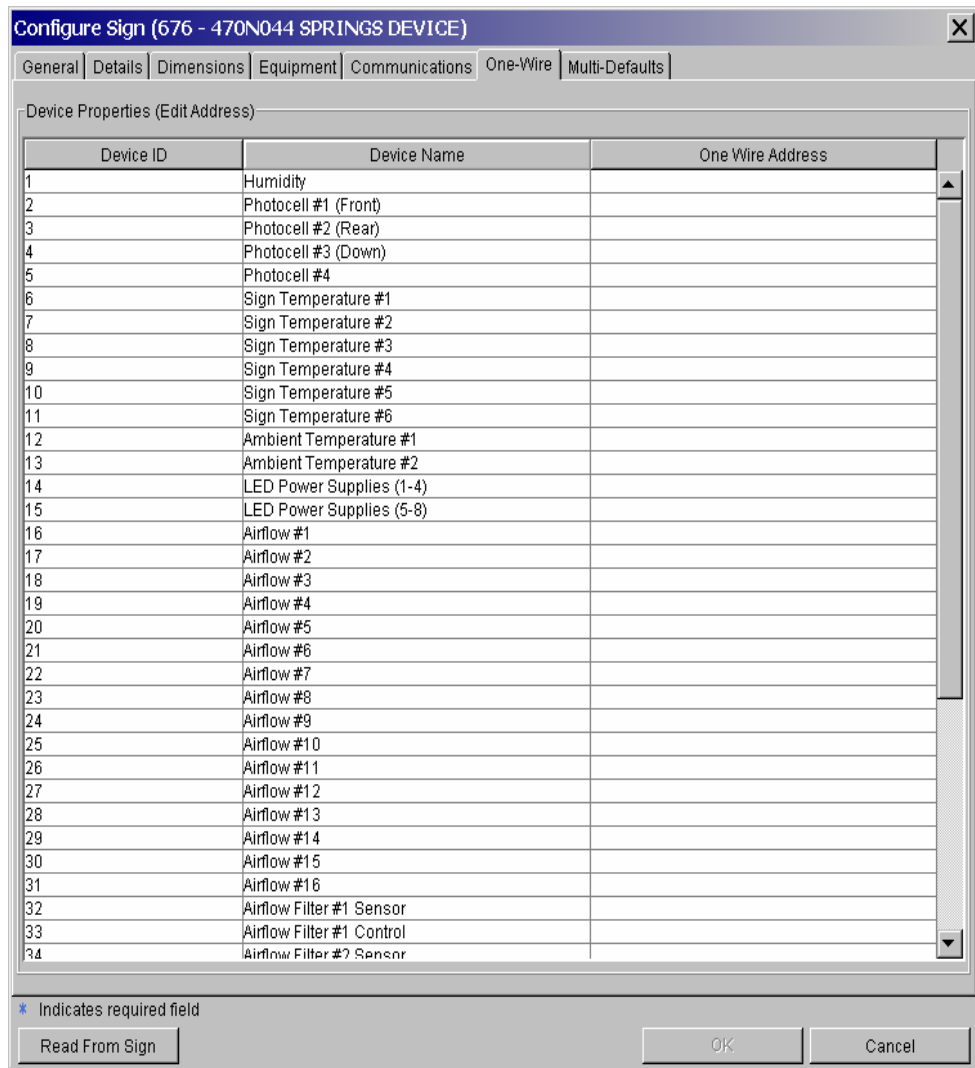


Figure 75: Configure DMS - One Wire Page

Configure Sign (676 - 470N044 SPRINGS DEVICE) [X]

General | Details | Dimensions | Equipment | Communications | One-Wire | **Multi-Defaults**

Sign Defaults

Describe Appearance

Font Name: ▼

Page Justification: ▼

Line Justification: ▼

Background Color: ▼

Foreground Color: ▼

Describe Display Times

Page On (secs):

Page Off (secs):

Page Flash On (secs):

Page Flash Off (secs):

* Indicates required field

Figure 76: Configure DMS - Multi Defaults Page

Manage Communications Pool

Manage Pool Dialog displays all communication pools available within CTMS.

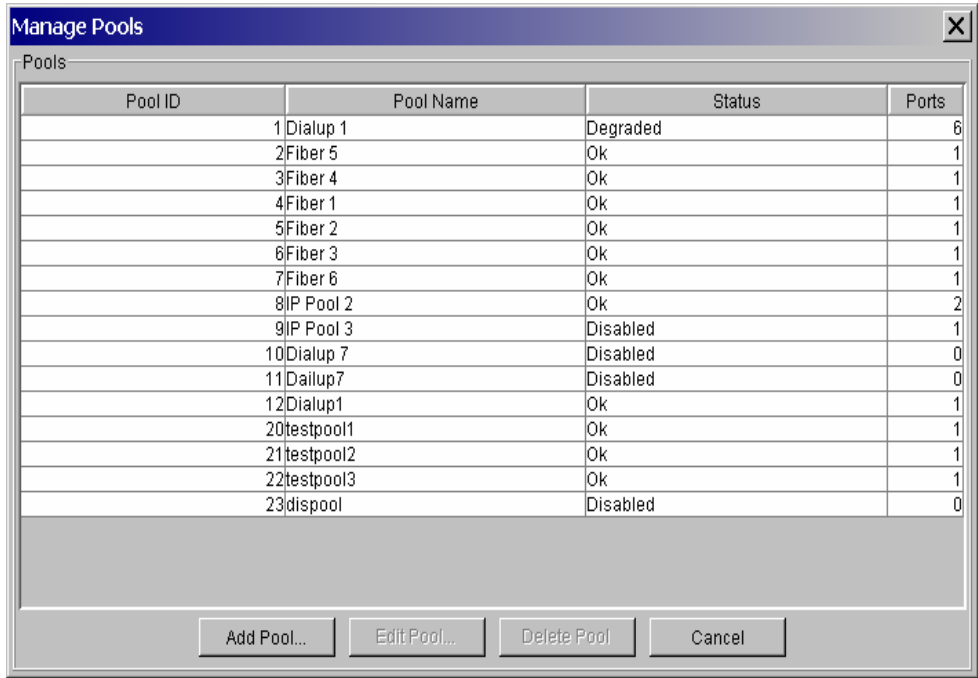


Figure 77: Manage Communications Pool

CTMS/CTIS	Version: 1.6
GUI Detailed Design	Date: 03-Dec-04

Add New Pool

Add New Pool Dialog allows CTMS user to add a New Pool.

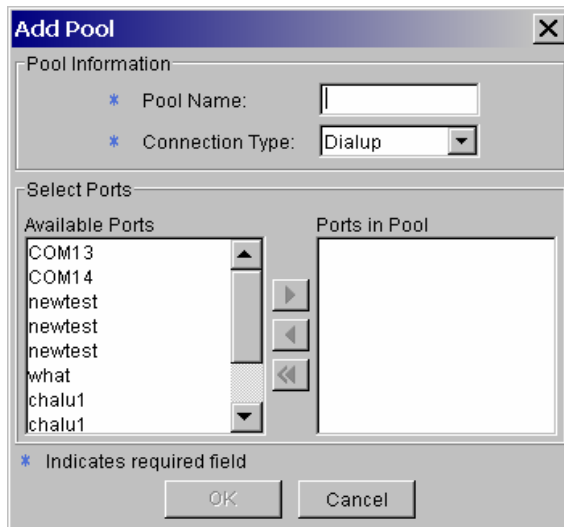


Figure 78: Add Pool

Edit Pool

Edit Pool dialog allows CTMS user to edit an existing pool.

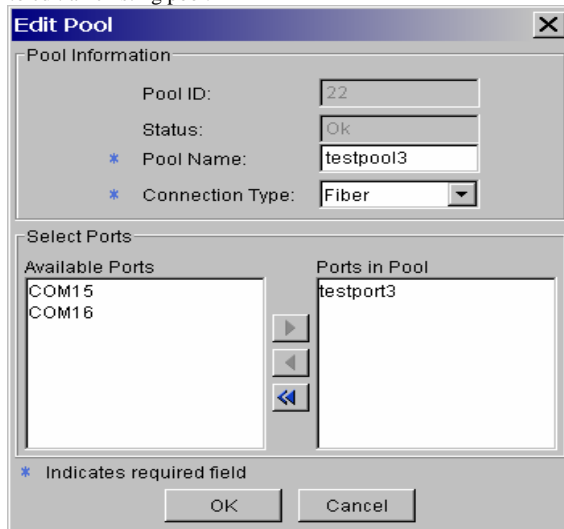


Figure 79: Edit Pool

Manage Communications Ports

Manage Ports Dialog displays all ports currently within CTMS system.

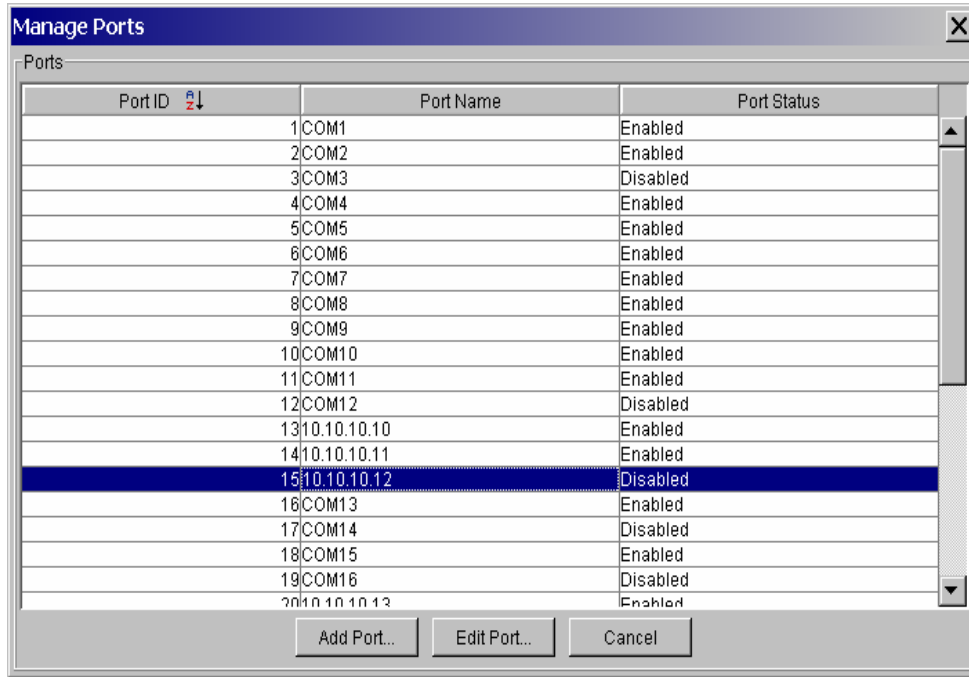


Figure 80: Manage Communications Port

CTMS/CTIS	Version: 1.6
GUI Detailed Design	Date: 03-Dec-04

Add New Port

Add New Port dialog allows CTMS admin to add a new port.

The 'Add Port' dialog box is titled 'Add Port' and has a close button (X) in the top right corner. It contains a section titled 'Port Information' with the following fields:

- * Port Name: [Empty text box]
- * Physical Port Name: [Empty text box]
- * Server IP: [Empty text box]
- Initialization String: [Empty text box]
- * Status: [Enabled] (dropdown menu)
- * Connection Type: [Dialup] (dropdown menu)

At the bottom, there is a legend: '* Indicates required field'. Below the legend are three buttons: 'Test Port', 'OK', and 'Cancel'.

Figure 81: Add New Port

Edit Port

Edit Port allows CTMS admin to edit information about an existing Port.

The 'Edit Port' dialog box is titled 'Edit Port' and has a close button (X) in the top right corner. It contains a section titled 'Port Information' with the following fields:

- * Port ID: [15]
- Pool ID: [9]
- Pool Name: [IP Pool 3]
- * Port Name: [10.10.10.12]
- * Physical Port Name: [10.10.10.12]
- * Server IP: [100.1]
- Initialization String: [Empty text box]
- * Status: [Disabled] (dropdown menu)
- * Connection Type: [IP] (dropdown menu)

At the bottom, there is a legend: '* Indicates required field'. Below the legend are three buttons: 'Test Port', 'OK', and 'Cancel'.

Figure 82: Edit Port

CTMS/CTIS	Version: 1.6
GUI Detailed Design	Date: 03-Dec-04

Help Navigator

Help Navigator is the main Help Window that displays a tree like table of contents (left of navigator) and HTML topic files (right of navigator).

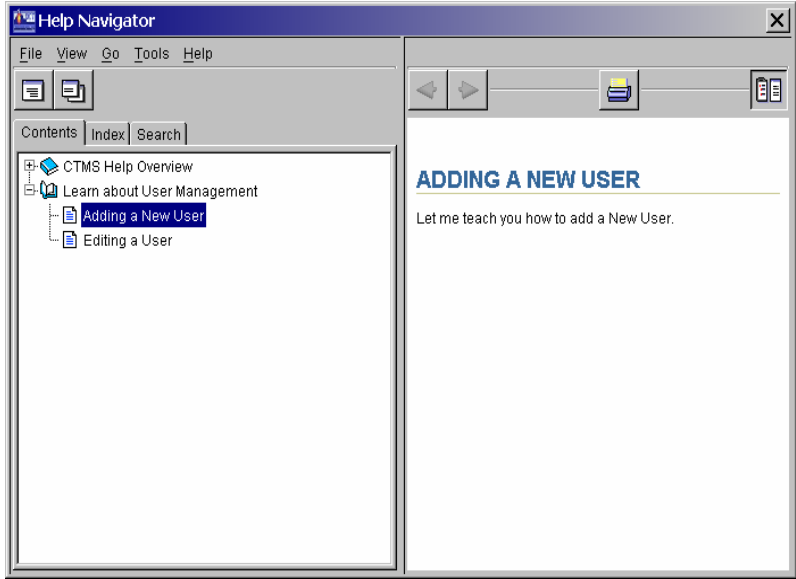


Figure 83: Help Navigator

CTMS/CTIS	Version: 1.6
GUI Detailed Design	Date: 03-Dec-04

Appendix B: The Menu XML Descriptor File

The UI consists of two application windows that each own an independent menu system. Both menu systems will be completely defined through the use of an XML file in order to gain maximum flexibility and control. The menu items are defined in the XML file using the following format (spacing has been modified to increase readability):

```

<menu-config>
<!-- the menu configuration can define multiple menu sets -->
<menu-set name="name used to reference this menubar">
<!-- the first menu for the described menu set : File Menu-->
<menu name="name for this menu"
namekey="key used to lookup the name in the client resources"
mnemonic_idx="index used for the mnemonic - not required"
accelerator="the accelerator keycode - not required"
level="the level for the menu"
use_seperator="false"/>
<!-- the first menu item for the described menu -->
<menu-item name="name for the menu item"
namekey="..."
mnemonic="..."
accelerator="..."
action_name="unique identifier for the ui action"
action_class="cdot.ctms.layer.ui.util.actions.PrintMapAction"/>
<menu-item .../>
...
</menu>
<menu .../>
<!-- the first menu item for the described menu : View Menu-->
<menu-item name="Refresh" namekey="nav_menu_view_refresh" mnemonic="0"/>
<menu-item name="Status Console" namekey="nav_menu_view_status_console" usecheck="true" mnemonic="1"/>
<menu-item name="Toolbar" namekey="nav_menu_view_toolbar" usecheck="true" mnemonic="1"/>
<menu-item name="OverviewMap" namekey="nav_menu_view_overviewmap" usecheck="true" mnemonic="1"/>
</menu>
...
</menu-set>
<menu-set name="name used to reference another menubar">
<!-- the first menu for the menu set -->
<menu name="File" namekey="nav_menu_file" mnemonic_idx="0">
<menu-item name="nav_menu_file_print" accelerator="" mnemonic="0" action_name="" action_class="" />
<menu-item name="name_menu_file_exit" accelerator="" mnemonic="1" action_name="" action_class="" />
</menu>
</menu-set>
...
</menu-config>

```

Table 2: Menu XML Description

CTMS/CTIS	Version: 1.6
GUI Detailed Design	Date: 03-Dec-04

A single XML file may contain one or for menu sets defined using the “menu-set” element tag. Each menu-set may have one or more menus, defined using the “menu” element tag. The menu element is composed of a collection of menu items defined using the “menu-item” tag.

A menu or menu item may have several attributes including name, namekey, mnemonic_idx, accelerator, level, and use_separator. The namekey attribute is used to lookup the menu name contained in the application resources as described in the ctms_labels.properties file. If the namekey is not found, then the name attribute will be used by default. The mnemonic_idx attribute is used to determine which char will serve as the mnemonic, if desired. The accelerator will be used to communicate an accelerator key combination, if desired. The level attribute will be used to communicate where the menu or menu item should be placed in relation to the previous menu or menu item. A level of zero means that the menu/menu-item is on the same level as the previous menu element. A “1” means that the element should go on a sub-menu of the previous menu element. Finally, the use_separator attribute is used to communicate whether the element should be followed by a separator.

Appendix C: The UI Action XML Descriptor File

The UI Action Factory contains a set of actions for all user interactions with the client system. These actions are defined in and loaded from an XML descriptor file. The actions are defined in the XML file using the following format:

```

<action
config>
  <ui
actions>
  <type name="action_key_name" class
name="the_class_name"/>
  <type name="another_action_key_name class
name="another_class_name" iconfile="filename"
      tool
tip
key="tooltip_key"/>
  </ui
actions>
</action
config>

```

Table 3: UI Action XML Description

The collection of action collections are defined in a logical grouping called “action-config”. The set of ui actions is defined by the element called “ui-actions”. Each action within the set of the ui actions is define by the element “type”. Each type must have a name and a class-name. The name is the internal key that will be used to refer to the action within the system. This key must map a static final defined in the UIActionFactory object. The class name is the name of the class that will be instantiated when the action factory is loaded. Each action may also have some optional attributes for defining an icon to use and a key for the tooltip. The “iconfile” attribute may be used to define an image resource that is to be used as the icon for the action. The “tool-tip-key” attribute is used to communicate a key that will be used to lookup the tooltip in the ApplicationProperties object that has loaded to appropriate key-value pairs from the property files.

CTMS/CTIS	Version: 1.6
GUI Detailed Design	Date: 03-Dec-04

Appendix D: The Map Configuration XML File

```

<!--Extended version of ArcXML generated by MapObjects - Java Edition v2.0.0 Build 452.1251 -->
<?xml version="1.0" encoding="UTF-8" ?>
<ARCXML version="1.1">
<CONFIG>
<ENVIRONMENT>
  <LOCALE language="en" country="US"/>
  <UIFONT name="Dialog"/>
  <SCREEN dpi="96"/>
  <SEPARATORS/>
</ENVIRONMENT>
<LAYOUT>
  <DATAFRAME>
    <MAPELEMENT active="true" id="3">
      <PROPERTIES position="0" dominant="false" shownames="false" preserveaspectratio="false"/>
      <ENVELOPE x="60.0" y="60.0" width="100.0" height="100.0"/>
      <MAP>
        <PROPERTIES>
          <ENVELOPE minx="440626.13010736817" miny="4367782.851044892"
maxx="522738.671330891" maxy="4429703.783770828" reaspect="false"/>
          <MAPUNITS units="decimal_degrees"/>
        </PROPERTIES>
        <FOLDERS>
          <FOLDER name="ws-0" type="SDEServer">
            <ATTRIBUTE name="password" value="KVIPLTDKQPOTPMWZUXDXFWAT"/>
            <ATTRIBUTE name="database" value=""/>
            <ATTRIBUTE name="username" value="atms"/>
            <ATTRIBUTE name="instance" value="port:5151"/>
            <ATTRIBUTE name="encrypted" value="true"/>
            <ATTRIBUTE name="server" value="10.82.70.10"/>
          </FOLDER>
        </FOLDERS>
        <LAYER name="State Boundaries" id="0" type="featureclass">
          <DATASET workspace="ws-0" name="ATMS.SDE_STATELINE" type="line"/>
          <SIMPLERENDERER>
            <SIMPLELINESYMBOL color="14,37,113" width="1" captype="round"/>
          </SIMPLERENDERER>
        </LAYER>
        <LAYER name="Counties" id="1" type="featureclass" visible="false">
          <DATASET workspace="ws-0" name="ATMS.SDE_COUNTIES" type="polygon"/>
          <SIMPLERENDERER>
            <SIMPLEPOLYGONSYMBOL fillcolor="207,200,181" boundarycaptype="round"/>
          </SIMPLERENDERER>
        </LAYER>
        <LAYER name="Highways" id="3" type="featureclass">
          <DATASET workspace="ws-0" name="ATMS.SDE_TRAVELCONDITIONS" type="line"/>
          <GROUPRENDERER>
            <VALUEMAPRENDERER lookupfield="TYPE">
              <EXACT label=" CO Highways" value="CO">
                <SIMPLELINESYMBOL color="192,101,73" width="2" captype="round"/>
              </EXACT>
            </VALUEMAPRENDERER>
          </GROUPRENDERER>
        </LAYER>
        <LAYER name="SkyLine VMS" id="6" type="featureclass">
          <DATASET workspace="ws-0" name="ATMS.SDE_VMS" type="point"/>
        </LAYER>
      </MAP>
    </MAPELEMENT>
  </DATAFRAME>
</LAYOUT>
</CONFIG>

```

CTMS/CTIS	Version: 1.6
GUI Detailed Design	Date: 03-Dec-04

```
<VALUEMAPRENDERER lookupfield="STATUS">
<EXACT label="ERROR/Failure" value="-1">
<RASTERMARKERSYMBOL url="" image="C:\testesri\images\vms_failure.gif"/>
</EXACT>
</LAYER>
</MAP>
</MAPELEMENT>
</DATAFRAME>
</LAYOUT>
</CONFIG>
</ARCXML>
```

Table 4: Map Configuration XML

CTMS/CTIS	Version: 1.6
GUI Detailed Design	Date: 03-Dec-04

Appendix E: Help System Example Code

A sample code from HelpAction.java

```
public void doAction(Object actionObject) {
    HelpUtilities.displayHelpforTopic(HelpAction.DEFAULT_KEY);
}
```

Table 5: The Help Action

One single step accomplishes adding Help to the dialog steps involved and can be best understood from this sample code from AddUserDialog.java

```
HelpUtilities.addModalTopic(this, this);
```

Table 6: Example of Adding a Modal Help Topic

This line registers the AddUserDialog with the Help System.

```
public static void addModalTopic(Window comp, IHelpConnector help){
    HelpSystem.getInstance().registerModalWindowtoHelp(comp);
    addTopic(help);
    comp.addWindowListener(new WindowAdapter(){
        public void windowClosed(WindowEvent e){
            HelpSystem.getInstance().unregisterModalWindowtoHelp((Window)e.getSource());
            System.out.print("unregistered help from component" +e.getSource());
        }
    });
}
```

Table 7: Using The Help Dialog

The above lines describe the addModalTopic() method of HelpUtilities. The method accomplishes three things, it registers the component to Help, adds the component to Help and finally unregisters the window after window is closed.

CTMS/CTIS	Version: 1.6
GUI Detailed Design	Date: 03-Dec-04

Appendix F: Client Deployment Guide

This section will describe the steps involved in installing the CTMS client software. The installation of the CTMS client does not require any registry, environment, configuration file updates. Additionally, installation of a Java JRE is not required, since it is packaged in the client distribution.

System Configuration

In preparation for the client installation, the target systems' display may need to be configured. For best results, please follow these guidelines:

- 1) Ensure that the display is setup for multiple monitors. The minimum screen resolution is 1280x1024.

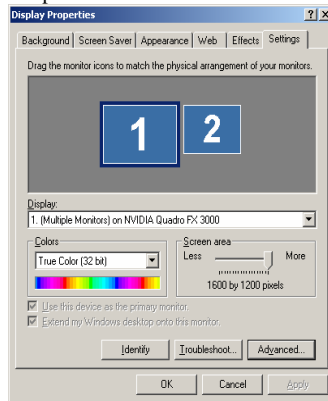


Figure 84: Monitor Settings

- 2) Select the Advanced button, and ensure that the Display Mode is "DualView".

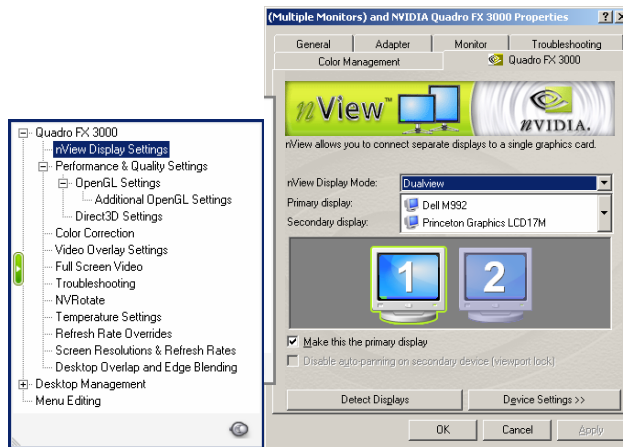


Figure 85: Graphics Device Settings

NOTE: It is highly recommended that a flat panel monitor should not be used, since the image quality is poor.

CTMS/CTIS	Version: 1.6
GUI Detailed Design	Date: 03-Dec-04

Installation Procedure

The following steps will describe the procedure for installing the CTMS software.

- 1) Obtain the client dist that is labeled "ctms-client-xxx-xxx.zip"; where xxx-xxx is the type of server configuration. More than likely, the file will be titled "ctms-client-prod-clustered.zip."
- 2) Unzip the client distribution zip file into any desired location. It is recommended that the destination c:\CTMS be used.
- 3) Create a shortcut (CTMS.lnk) to the .bat file.
 - a. Right click on the batch file to create the shortcut
 - b. Assign an appropriate icon (one will be provided in the near future)
 - c. Copy the shortcut from the bin directory to the system desktop.

Verify Installation

- 1) launch client by executing CTMS\bin\run.bat or using the CTMS.lnk file.
- 2) verify that the login prompt is presented
- 3) login as a valid user
- 4) verify that the map loads
- 5) select a group of devices and verify that the table is presented.

Update Procedure

In the event that a new version of CTMS is released, the client zip may be extracted over the top of the existing software release.