RELEASE NOTES

<table>
<thead>
<tr>
<th>Version Number</th>
<th>Date</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version 1.0</td>
<td>15 June 2007</td>
<td>Initial version delivered to the FHWA</td>
</tr>
<tr>
<td>Version 1.1</td>
<td>15 November 2007</td>
<td>Final version based on FHWA feedback</td>
</tr>
</tbody>
</table>

Questions and comments about this document should be directed to:

Dr. William Myers  
Research Applications Laboratory  
NCAR  
P.O. Box 3000  
Boulder, CO 80307  
Ph: 303-497-8412  
E-mail: myers@ucar.edu

Mr. Jim Cowie  
Research Applications Laboratory  
NCAR  
P.O. Box 3000  
Boulder, CO 80307  
Ph: 303-497-2831  
E-mail: cowie@ucar.edu
# TABLE OF CONTENTS

1 INTRODUCTION ........................................................................................10

2 INTENDED AUDIENCE ........................................................................................10

3 OVERVIEW ........................................................................................10

3.1 Disclaimer .................................................................................................................11

3.2 Summary of System Improvements since Release-4 ................................................11

4 RELATED DOCUMENTS ........................................................................................13

5 MDSS SYSTEM OVERVIEW ........................................................................................14

5.1 System Goals ............................................................................................................14

5.2 System Structure .......................................................................................................15

5.2.1 Computer Hardware ......................................................................................15

5.2.2 Communications and Network Interfaces .....................................................16

5.2.3 Software Architecture ...................................................................................16

5.2.4 Computer Languages ....................................................................................17

5.2.5 External Software Requirements ..................................................................17

5.2.6 Inter-Process Communication .......................................................................18

5.2.7 Data Ingest ....................................................................................................18

5.3 MDSS Component Overview ...................................................................................19

5.3.1 External Data Sources ...................................................................................20

5.3.2 Road Condition and Treatment Module (RCTM) ........................................20

5.3.3 Pavement Frost Product .............................................................................22

5.3.4 Alert Generator .............................................................................................23

5.3.5 Treatment Update Network Layer (TUNL) ..................................................23

5.3.6 Thematic Realtime Environmental Distributed Data Services (THREDDS) Server ..........................................................................................................24

5.3.7 WMS Server ..................................................................................................24

5.3.8 MDSS Display ..............................................................................................25

6 SYSTEM OPERATIONS ........................................................................................25

6.1 Process Invocation ....................................................................................................25

6.1.1 Schedule Driven Forecast System ................................................................25

6.1.2 Glue Layer ....................................................................................................26

6.1.3 Display Data Requests ..................................................................................26

6.2 Display Invocation ....................................................................................................26

7 PROCESS ERROR HANDLING ..............................................................................27

7.1 Data Logging ............................................................................................................27

7.1.1 Log Files .......................................................................................................27

7.1.2 Log File Organization .................................................................................27

7.1.3 Log File Hierarchy .......................................................................................27

7.1.4 Log File Naming Conventions ......................................................................28

7.1.5 Executable’s Log File Messages ...................................................................28

7.2 Error Handling ........................................................................................................28

7.2.1 Index Files .....................................................................................................28

7.2.2 Handling Old Data ........................................................................................29

7.2.3 Failures in Client-Server Interactions ...........................................................29
13.6.4.2 Road Forecast File ................................................................. 51
13.6.5 Output ...................................................................................... 51
13.7 Processing ..................................................................................... 54
14 Alert Generator .................................................................................. 57
  14.1 Identification ................................................................................. 57
  14.2 Type .............................................................................................. 57
  14.3 Purpose .......................................................................................... 57
  14.4 Function ........................................................................................ 57
  14.5 Dependencies ................................................................................ 57
  14.6 Interfaces ....................................................................................... 58
    14.6.1 Command Interface ................................................................. 58
    14.6.2 Start Script .............................................................................. 58
    14.6.3 Configuration Files ................................................................. 58
      14.6.3.1 Site List Configuration File .............................................. 58
      14.6.3.2 METAR Neighbor Configuration File ......................... 58
    14.6.4 Inputs ...................................................................................... 59
      14.6.4.1 Observation Data File ..................................................... 59
      14.6.4.2 Road Condition Forecast Data File .............................. 60
    14.6.5 Output ..................................................................................... 60
  14.7 Processing ..................................................................................... 60
15 THREDDS Data Server ................................................................. 61
  15.1 Identification ................................................................................. 61
  15.2 Type .............................................................................................. 61
  15.3 Purpose .......................................................................................... 61
  15.4 Function ........................................................................................ 61
  15.5 Dependencies ................................................................................ 61
  15.6 Interfaces ....................................................................................... 62
    15.6.1 Command Interface ................................................................. 62
    15.6.2 Catalog Files ............................................................................ 62
    15.6.3 Input ......................................................................................... 64
      15.6.3.1 Radar Data ........................................................................ 64
      15.6.3.2 Satellite Data ..................................................................... 66
  15.7 Processing ..................................................................................... 68
16 WMS Server .................................................................................... 69
  16.1 Identification ................................................................................. 69
  16.2 Type .............................................................................................. 69
  16.3 Purpose .......................................................................................... 69
  16.4 Function ........................................................................................ 69
  16.5 Dependencies ................................................................................ 69
  16.6 Interfaces ....................................................................................... 69
17 Utility Processes ............................................................................... 70
  17.1 Merge Variables Process ............................................................. 70
    17.1.1 Identification ........................................................................... 70
    17.1.2 Type ........................................................................................ 70
    17.1.3 Purpose ................................................................................... 70
18.3.2 Get Latest Data Script .................................................................81
18.3.3 Get Closest Data Script ...............................................................81
18.3.4 List Data Script .........................................................................81
18.3.5 Get Permissions Script ...............................................................82
18.3.6 Select Treatment Plan Script ......................................................82
18.3.7 User Defined Treatment Plan Script ..........................................82
18.3.8 Reset Sites Script ......................................................................82
18.3.9 Get Reset Sites Script ...............................................................82
18.4 Function .......................................................................................83
18.5 Dependencies ..............................................................................83
18.6 Interfaces ....................................................................................83
18.6.1 Command Interfaces .................................................................83
18.6.1.1 Get Current Time Script ..........................................................83
18.6.1.2 Get Latest Data and Get Closest Data Scripts .......................84
18.6.1.3 Get Permissions Script ..........................................................84
18.6.1.4 Select Treatment Plan ............................................................85
18.6.1.5 User Defined Treatment Plan ...............................................86
18.6.1.6 Reset Script ..........................................................................88
18.6.1.7 Get Reset Sites Script ...........................................................88
19 DISPLAY .........................................................................................88
19.1 Identification ...............................................................................89
19.2 Type ............................................................................................89
19.3 Purpose ......................................................................................89
19.4 Function ....................................................................................89
19.5 Dependencies ............................................................................93
19.5.1 Java Libraries ...........................................................................93
19.5.1.1 commons-httpclient-3.0-rc2-all.jar .......................................93
19.5.1.2 jdom.jar .............................................................................94
19.5.1.3 jaxen-full.jar ......................................................................94
19.5.1.4 saxpath.jar .........................................................................94
19.5.1.5 netcdf-2.2.18.jar ...............................................................94
19.5.1.6 nlog4j-1.2.21.jar .................................................................94
19.5.1.7 shapefile-2.0-B2.jar ............................................................95
19.5.1.8 core-2.0-B2.jar ..................................................................95
19.5.1.9 resources-2.0-B2.jar .........................................................95
19.5.1.10 JTS-1.3.jar .......................................................................95
19.5.2 Geographic Location Configuration ..........................................95
19.5.2.1 Weather and Road Forecast Sites ........................................96
19.5.2.2 Observation Sites .................................................................96
19.5.2.3 MDSS Places Configuration ...............................................98
19.5.3 Shapefile-based Definitions ......................................................100
19.5.3.1 Alert Zones .........................................................................100
19.5.3.2 Road Segments ...................................................................101
19.5.4 Runtime Environment .............................................................102
19.5.5 Java WebStart Deployment .................................................................102
19.6 Interfaces ..............................................................................................104
  19.6.1 Command Interface ........................................................................104
  19.6.2 Input ................................................................................................104
      19.6.2.1 Weather Forecast Data File .....................................................105
      19.6.2.2 ESS Observation Files ..............................................................105
      19.6.2.3 Road Condition Forecast Files ...............................................105
      19.6.2.4 User-Specified Treatment Road Condition Forecast Files ........105
      19.6.2.5 Input File Format .................................................................106
19.7 Output ...................................................................................................108
20 Future Enhancements and Refinements ......................................................108

APPENDICES

Appendix A: Road Temperature Model (METRo) ..............................................110
Appendix B: Chemical Concentration Algorithms ............................................112
Appendix C: Net Mobility Module ..................................................................125
Appendix D: Rules of Practice Module ............................................................126
Appendix E: Characterize Storm Module .........................................................137
Appendix F: Technical Points of Contact .........................................................143
ACRONYM GLOSSARY

AVL – Automatic Vehicle Location
CDL – Command Description Language
DOT - Department of Transportation
DSS – Decision Support System
ESRI – Environmental Systems Research Institute
ESS – Environmental Sensor Station
FHWA – Federal Highway Administration
FM – Forecast Module
FP – MDSS Functional Prototype
FSL – NOAA, Forecast Systems Laboratory
FTP – File Transfer Protocol
GINI – GOES Ingest NOAAPort Interface
HOTO - Office of Transportation Operations, FHWA
JDK – Java Development Kit
MADIS – Meteorological Assimilation Data Ingest System (NOAA/FSL)
MDSS - Maintenance Decision Support System
METAR – Meteorological Terminal Air Report
METRIO - Model of the Environment and Temperature of Roads
MIT/LL - Massachusetts Institute of Technology - Lincoln Laboratory
MOS – Model Output Statistics
NCEP – National Centers for Environmental Prediction
NetCDF – Network Common Data Format
NSSSL – NOAA, National Severe Storms Laboratory
NOAA – National Oceanic and Atmospheric Administration
NOAAPort – NOAA data stream portal
NCAR - National Center for Atmospheric Research
NWS – National Weather Service
OCD – Operational Concepts Description
OGC – Open Geospatial Consortium
RAL – Research Applications Laboratory, NCAR (formerly RAP)
RAP – Research Applications Program, NCAR
RCTM – Road Condition and Treatment Module
RWFS – Road Weather Forecast System
RWIS – Road Weather Information System
RWMP - Road Weather Management Program
STWDSR - Surface Transportation Weather Decision Support Requirements
THREDDS - Thematic Realtime Environmental Distributed Data Services
TUNL – Treatment Update Network Layer
UCARF – University Corporation for Atmospheric Research Foundation
UNIDATA – University Data program of the UCAR Office of Programs
WIST-DSS - Weather Information for Surface Transportation Decision Support System
WMO – World Meteorological Organization
WMS – Web Map Service
1 INTRODUCTION

This document describes Release-5.0 of the prototype Maintenance Decision Support System (MDSS) technical components (e.g., code and processes) and is organized into two main parts: a main body and a set of appendices. The main body contains technical descriptions of the prototype MDSS software, whereas the appendices include descriptions of algorithms and techniques used in the system. The format of the main body is based on the Institute of Electrical and Electronics Engineers (IEEE) standard for Software User Documentation (ANSI/IEEE standard 1063-1987). This document is organized so that high-level system descriptions are presented first followed by detailed descriptions of each system component. The detailed descriptions are presented as software processes covering process type, purpose, function, dependencies, and interfaces.

2 INTENDED AUDIENCE

The intended audience of this document is software engineers with extensive knowledge of the C++ programming languages and the UNIX operating system. Technical points of contact for the prototype MDSS system are provided in Appendix F.

3 OVERVIEW

This MDSS Project is part of a federal procurement for research projects and deployment advocacy, which is funded through the Intelligent Transportation Systems (ITS) Joint Program Office (JPO) and Office of Transportation Operations of the Federal Highway Administration (FHWA).

It is envisioned that components of the prototype MDSS system developed by this project will be further developed, integrated with other operational components, and deployed by road operating agencies, including state departments of transportation (DOTs), and generally supplied by the private sector.

Five national research centers have participated in the development of the MDSS during the course of the project. The participating national labs include:

- Army Cold Regions Research and Engineering Laboratory (CRREL)
- National Center for Atmospheric Research (NCAR)
- Massachusetts Institute of Technology - Lincoln Laboratory (MIT/LL)
- NOAA Forecast Systems Laboratory (FSL)
- NOAA National Severe Storms Laboratory (NSSL)
3.1 Disclaimer

The MDSS described herein is a prototype software system. For this reason, the MDSS software should be used with caution. It is anticipated that the prototype MDSS software will be used as a springboard toward the development of commercial road weather systems that contain MDSS features and functions. How the materials that make up this release are used to create an operational capability is ultimately up to the private sector firms or other organizations seeking to provide those services. Some may choose to utilize the prototype code while others may use the contents as general guidelines for their own development process. Several private sector companies have already begun development of operational versions of the MDSS. It is hoped that the code and capabilities described herein and delivered as part of MDSS Release-5.0 will accelerate the time to market for operational MDSS technologies.

Because the MDSS is only a prototype system, not all winter maintenance decision support capabilities desired by DOTs have been incorporated. For example, most DOTs have indicated a desire that an operational MDSS interface with their central database to obtain actual treatment data and that the MDSS output its recommended treatments to a DOT archive. Because such interfaces are specific to each DOT they have not been incorporated into the prototype, which is mainly designed to be a generic (i.e., not state specific) proof-of-concept system.

3.2 Summary of System Improvements since Release-4

The MDSS has been refined and extended significantly since Release-4 (Feb 2006), which has been used for demonstrations within Colorado during the 2005-6 and 2006-7 winters. Several parts of the MDSS were upgraded based on experience and user feedback from those seasons. Maintenance personnel from the E-470 Public Highway Authority and the City and County of Denver used Release-4 software during their winter road maintenance operations. Their feedback led to further development and refinement of the system. These software upgrades were developed during the fall and winter of 2006-7. Prototype versions of this software were used during the latter months of the 2007 winter/spring season. They will be used during the Colorado field demonstration covering the 2007-8 winter season.

The Road Condition and Treatment Module (RCTM) was upgraded to use a different road temperature model. The old model, SNTHERM, is no longer supported. After an extensive evaluation period, METRo, a road temperature model developed at Environment Canada was chosen to replace SNTHERM. This model has functionality similar to that of SNTHERM; however, METRo is much easier for users to implement and adapt.

The MDSS display was completely revised. While much of the look and feel remains the same, the underlying code base was ported to an entirely new Java visualization framework. The new framework provides better support for asynchronous data retrieval, time selection, and emerging standard data formats. Data layers may be added or removed easily. Maintenance Districts and management areas may now be nested or overlapped. The same display may be used for real-time planning and archival event reviewing.
Several new capabilities were added to the display. A major extension is the ability to display gridded products such as radar and satellite data. These data are loaded in one of the emerging standard meteorological data formats, NetCDF. Another major extension is the support for dynamic basemaps. These maps show the appropriate resolution of roads, political boundaries, and topographic detail for the current pan and zoom. Plow truck AVL data may now be displayed in real time, allowing managers to view vehicle location, speed and direction, treatment, and observed weather and road conditions. Road, weather, and observation variables may now be displayed at the same time. All road variables may now be displayed on the map. Finally, new alerts were added for road temperatures of wet roads predicted to fall below freezing and for freezing precipitation observed.

The new display is far more localizable than its predecessors. All local configuration is specified in files separate from the application source code. Recompilation would not be required to move the display from one state to another. Nearly all data layer configuration is specified in a single XML file. Geographic data is dynamically requested using the web map service WMS protocol. Road segments and alert zones are specified in Shapefile format.

Additional improvements were made to the display to simplify its code, improve performance, and catch networking or data errors.

A summary of MDSS prototype enhancements since Release-4 is provided below:

**Road Condition and Treatment Module:**

- Replaced SNTHERM with METRo
- Simplification of configuration files

**Display Application:**

- Added new gridded datasets:
  - National radar mosaic
  - Combined satellite visible image
  - Combined satellite infrared image
  - Combined satellite water vapor
- Added real-time truck AVL data
- Added dynamic basemaps
- Added map variable selection for road segments
- Added independent controls for weather, road, and observed variables
- Added road temperature and freezing precipitation alerts
- Enabled archive browsing from the real-time display
- Added arbitrary panning and zooming
- Isolated and simplified configuration

**Frost Module**

- Updated code to handle new configuration files
- Embedded material properties for asphalt, concrete, etc. into code
Alert Generation Module

- Developed new module that generates hazard alerts
- Hazards contain predicted bad road conditions and observed inclement weather

Gridded Data Set Servers

- Developed server capabilities to provide radar, satellite, or other gridded data sets to display
- Publicly available Unidata servers were configured for this purpose
- Technically not part of MDSS, but capability completes proof of concept

4 RELATED DOCUMENTS

For additional information on the MDSS Project, the reader is directed to related project documents listed in Table 1.

Table 1. Related Documents

<table>
<thead>
<tr>
<th>Document and/or Web Sites</th>
<th>Source</th>
</tr>
</thead>
</table>
5 MDSS SYSTEM OVERVIEW

5.1 System Goals

The MDSS described herein is designed to be a functional prototype of a decision support tool used by winter road maintenance managers. The end-user products are designed to satisfy requirements laid out in the Surface Transportation Weather Decision Support Requirements (STWDSR) document. In general, the system is designed to provide timely weather and road condition forecasts coupled with road treatment planning tools.

To make it easier to evaluate and/or implement MDSS components, source code is provided along with this release for all of the modules. This should enable the extraction of pieces of the system for incorporation within another system. Source code can be obtained by registering at the MDSS website:

http://www.rap.ucar.edu/projects/rdwx_mdss

The MDSS ingests weather forecast data at locations important to the users’ operations. These forecast locations are typically at surface observation stations such as RWIS and METAR sites, though they need not be. The weather forecasts at each forecast location serve as input to the pavement heat balance model (e.g., METRo) that predicts the road surface and subsurface temperatures and the snow depth at each forecast lead time. These forecast road conditions are used to generate treatment plans at each site based on Rules of Practice guidelines. The prototype MDSS also includes a graphical user interface display designed for easy interpretation by road maintenance managers. This display application is designed to allow the maintenance manager to generate “what-if” scenarios by setting up customized treatment plans and seeing the resulting predicted road conditions.

The MDSS was designed to be modular so that those interested in specific components could utilize those capabilities without needing all the components. For example, if implementers had access to a different pavement temperature model, they could swap out METRo and replace it with their preferred model. Implementers can download, purchase, or otherwise generate weather forecasts to drive the RCTM. The MDSS components can be run separately and implementers do not have to run all the prototype components.

The MDSS is a research and development effort. As such, overall reliability of the system code has not been exhaustively evaluated; in particular the RCTM code has been tested on numerous winter cases, but it is not possible to test the code for all possible combinations of weather and road conditions.

The described system represents the MDSS prototype as developed through May 2007.

The Release-5.0 MDSS was built and run on the Debian Linux 3.1 (Sarge) operating system. The MDSS server software is modular and is written in the C++ programming language. Scripting languages such as Python and Perl are also used to control system operations. Porting the system to other UNIX-based systems is not expected to be difficult.
The MDSS display has been designed to run on a variety of platforms. It has been developed as a Java application allowing the display to run on any platform that supports Java. This allows a variety of end users to use the display on their existing platforms. The display is a client application which polls and acquires data from one or more servers. This data includes weather and road condition forecasts, observation data such as current air and road temperatures, radar and satellite images, AVL data, and geographic basemaps.

The MDSS includes a number of configuration files. To configure this software for a specific region of interest, the installation guide included with this document should be consulted. The configuration of the RCTM requires the creation of files describing the subsurface structure, traffic levels, route times, default chemicals, etc., at each road forecast site. The configuration of the Treatment Update Network Layer (TUNL) requires defining which hosts and users can access components of the system. Finally, the configuration of the MDSS display requires defining forecast points, observation points and sensor types, road segments, maintenance areas, and alert zones. All these modifications are described either herein or in the MDSS Installation Guide, which can be found in the /docs directory of the MDSS software CD.

5.2 System Structure

The MDSS consists of ingest processes, algorithm processes, and a display process. This section discusses the hardware and software architecture and the communications and network interfaces of the prototype MDSS.

5.2.1 Computer Hardware

The computational requirements of the MDSS are linearly related to the number of forecast sites. For all MDSS site list configurations seen thus far (~500 forecast sites), one standard computer (e.g., PC) is sufficient to run all the subsystems described in this document. The MDSS is scalable, and if the number of forecast sites is sufficiently large, more machines with the same configuration will be required or a single machine with faster processors and more memory will be required.

The display can also be run on this same single platform, but it is envisioned that users at remote sites would most likely be running a display locally. The system described here assumes that the data ingest and algorithms run on a single machine and the output is communicated to the end-users’ machines via the Internet.

Since processes may be user-request driven, the system load will be slightly more unscheduled and unpredictable than described above. These user requests will generally be rare and require a relatively small amount of processing. However, if the user base is large enough and enough requests are generated, the computational resources of the system may be taxed. In this unlikely case, it may be necessary to add further computational resources.

The MDSS hardware specification for Release-5.0 is displayed below. Requirements for running only the display component of the system are less than what is required for running
the RCTM and server components. Minimal PC requirements are shown. Additional memory and a faster processor should provide better display performance.

**RCTM components**

- Dual 3.6 GHz processors
- 300 GB disk space
- 2.0 GB memory

**Display PC**

- 1 GHz processor
- 500 MB disk space
- 512 MB memory

The first configuration should cost less than $5,000. The latter should cost less than $1,000.

## 5.2.2 Communications and Network Interfaces

The MDSS uses standard TCP/IP protocols to communicate data from the server to the client display. The TCP/IP protocol is the underlying standard for most communication on the Internet, so as long as the client has a connection to the Internet, that client can access the MDSS data server(s). For effective performance, an Internet link at DSL speeds would be considered the minimum for the MDSS system. This is on the order of 1.5Mb/s with faster speeds providing better throughput.

## 5.2.3 Software Architecture

The MDSS server algorithms have been designed to run on common UNIX workstations. Currently these subsystems have only been compiled and tested under the Intel-based Linux Debian operating system. The display has been developed as a Java application. As such, the binary-generated Java byte code is machine independent and should run on any properly configured hardware system as specified in Section 19.5.4.

The MDSS is a distributed system. It consists of a number of independent processes each performing a straightforward, well-defined function. The individual modules of the MDSS have been designed to be relatively simple. Each process, while perhaps doing sophisticated processing, has been designed to know very little about the outside world. For example, the processes know nothing about the file system, nor do they consult the system clock to find out the current time. Instead, all information required for processing, such as file names and relevant time parameters are passed to the process as command line arguments. The rationale behind this design is to create a system that is reliable, maintainable, and traceable.

One major advantage of this methodology is that every instance of every process is completely repeatable. Log files (described later) keep track of the command line executed
as well as the status of each process run. Bugs can easily be traced by repeatedly running the code with the same command line.

\section*{5.2.4 Computer Languages}

Each component of the system is coded in one of the following languages:

- C/C++
- Java
- Perl
- Python
- FORTRAN

The MDSS algorithm code is written in C++.

The scripting languages Perl and Python, are used for server-side request processing (CGI handling of display requests) and process invocation scripts. To the greatest extent possible, C and C++ code is POSIX compliant. FORTRAN code exists in the road temperature model, METRo, developed at Environment Canada.

\section*{5.2.5 External Software Requirements}

A number of third-party software packages are required to run the entire prototype MDSS system. These packages are freely available and must be installed before installing the MDSS system code. The table below lists these packages, the minimum or specific version required, and where they can be obtained.

<table>
<thead>
<tr>
<th>Package</th>
<th>Version Range</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>libxml2</td>
<td>2.6.x</td>
<td>ftp://xmlsoft.org/libxml2/</td>
</tr>
<tr>
<td>libxslt</td>
<td>1.1.x</td>
<td>ftp://xmlsoft.org/libxml2/</td>
</tr>
<tr>
<td>Python</td>
<td>2.4</td>
<td><a href="http://www.python.org/download/">http://www.python.org/download/</a></td>
</tr>
<tr>
<td>Numarray</td>
<td>1.5.x</td>
<td><a href="http://downloads.sourceforge.net/numpy/">http://downloads.sourceforge.net/numpy/</a></td>
</tr>
<tr>
<td>libxml2 binding</td>
<td>2.6+</td>
<td>ftp://xmlsoft.org/libxml2/python/</td>
</tr>
<tr>
<td>SWIG</td>
<td>1.3.x</td>
<td><a href="http://downloads.sourceforge.net/swig">http://downloads.sourceforge.net/swig</a></td>
</tr>
<tr>
<td>Perl</td>
<td>5.8+</td>
<td><a href="http://www.perl.org/get.html">http://www.perl.org/get.html</a></td>
</tr>
<tr>
<td>Java JDK</td>
<td>1.6+</td>
<td><a href="http://java.sun.com/javase/downloads/">http://java.sun.com/javase/downloads/</a></td>
</tr>
<tr>
<td>Apache Webserver</td>
<td>2.x</td>
<td><a href="http://httpd.apache.org/">http://httpd.apache.org/</a></td>
</tr>
<tr>
<td>suExec</td>
<td></td>
<td><a href="http://httpd.apache.org/docs/2.0/suexec.html">http://httpd.apache.org/docs/2.0/suexec.html</a></td>
</tr>
<tr>
<td>Apache Ant</td>
<td>1.7+</td>
<td><a href="http://ant.apache.org/">http://ant.apache.org/</a></td>
</tr>
<tr>
<td>PHP</td>
<td>4.x</td>
<td><a href="http://www.php.net/">http://www.php.net/</a></td>
</tr>
<tr>
<td>php4-common</td>
<td></td>
<td></td>
</tr>
<tr>
<td>libzip-0-12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>apache2-mpm-prefork</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GNU gcc/g++/g77</td>
<td>3.3+</td>
<td><a href="http://www.gnu.org/software/">http://www.gnu.org/software/</a></td>
</tr>
<tr>
<td>make/wget</td>
<td></td>
<td></td>
</tr>
<tr>
<td>netCDF</td>
<td>3.6+</td>
<td><a href="http://www.unidata.ucar.edu/packages/netcdf">http://www.unidata.ucar.edu/packages/netcdf</a></td>
</tr>
</tbody>
</table>
All MDSS source code is built using the GNU make and compilers (gcc, g++ and g77), with the exception of the Java display, which uses the Apache Ant utility. NetCDF is used as the primary data format for files in the RCTM. Perl and Python are used as general scripting languages in the TUNL and RCTM. Python numarray, libxml2, libxslt and SWIG are used by the road temperature model, METRo. The Apache webserver and suEXEC as well as PHP are used by the TUNL. And finally, the Java JDK is used to build the display source code, as well as provide the client start-up capability using Java WebStart.

### 5.2.6 Inter-Process Communication

MDSS server inter-process communication is done through the file system. Each process obtains its inputs solely through files on its command line. Many of these files specified are data files generated by upstream processes; many are static binary or ASCII-text data files. The remainder are configuration files that contain lists of sites, forecast variables to be processed, or other static configuration data, as required.

Each process reads its input data from the files on its command line, processes it, and writes its output to a file. The output file name is specified on the command line. The output file name’s format is specified in a file named on the command line.

The interface between the MDSS display and upstream processes is a client-server relationship. The display is the client; its requests are handled by a web server and processed using CGI scripts. The CGI scripts parse the requests and obtain the requested data. These data are then returned to the display.

### 5.2.7 Data Ingest

Several types of live data\(^1\) are required by the system. These data include road weather forecasts, weather and road observations, and grid data sets such as radar and satellite images. Some of these data sets are made available to the display after simple reformatting, but many are used by the MDSS server processes in creating road forecasts and alerts. The downloading, reformatting, or creation of the raw data sets is beyond the scope of this document. It is assumed that these data sets are on disk and available in the appropriate formats for use by the system.

---

\(^1\) Live data refers to data that flow into the system shortly after they are measured or obtained.
5.3 **MDSS Component Overview**

The MDSS consists of individual components which have well-defined roles in the larger system. This section describes in more detail the components which go into making the MDSS. Figure 5.1 illustrates an overview of the MDSS components.

![MDSS Component Overview Diagram](image)

Figure 5.1 Overview of MDSS system.
5.3.1 External Data Sources

Data used by the MDSS are obtained from a variety of sources. These data may include weather forecasts, observations such as RWIS data, radar and satellite images, as well as GPS/AVL data from properly equipped DOT maintenance trucks. Details regarding the retrieval of these data are beyond the scope of this document. However, interface specifications are provided in later chapters which will allow users to convert their local data into the proper format for use.

The Road Weather Forecast System (RWFS) developed by NCAR, has been used to date as the source of point-based weather forecasts to drive the RCTM and provide forecast data to the display. The RWFS uses freely available NCEP model, MOS, and observation data to create a time series of consensus forecasts at each location. A license from the UCAR foundation is required to obtain this code. It is not required that the RWFS be used as a source of weather forecasts; however a suitable replacement which generates output following the interface specifications in section 12 must be used as a replacement.

All forecast and observation data are ultimately placed in a data store which is accessible to internal MDSS processes and external display clients. Data are accessed by display clients via the TUNL or THREDDS mechanisms. The THREDDS server provides grid data such as satellite and radar, while the TUNL is the interface which provides all other observation and forecast data.

In Release-5, static GIS data such as state and county maps, road and topography data are no longer hard-wired into the display. These data are now accessed dynamically at run-time from a WMS server. The raw data and how it is to be displayed are configured on the Web Map Service. This allows far more flexibility in relocating the system to a new geographic domain, as well as allowing the system to display data on different special scales.

5.3.2 Road Condition and Treatment Module (RCTM)

The RCTM ingests and processes environmental forecast data to predict the road surface temperature and snow depths at all forecast sites and lead times. Using the meteorological forecast data, the pavement temperature data, and the snow depth on the road, a predicted mobility index is calculated along with a treatment plan. For prototyping purposes, only a small number of forecast sites have been selected per highway route, and weather and road condition data are only processed for those sites. In an operational version of the system, it is anticipated that the users would identify regularly spaced and/or strategic locations along winter maintenance routes for which they want decision support, and the system would be configured to generate treatment guidance for those locations. The output of the RCTM is passed to the display.

A different treatment plan is calculated for each highway forecast point. The treatment plans may vary spatially and no approach is currently in place to resolve any potential discrepancies between adjacent sites. Such discrepancies may be partially due to differing

---

2 For information on licensing RWFS, contact the MDSS Project Manager referenced in Appendix F.
subsurface structures of the roadways at adjacent points. For example, road conditions on a bridge can be very different from conditions a few hundred meters away. The RCTM is implemented as a single process. This is necessary due to feedback effects of plowing and chemical applications on the road. The treatment plan generation algorithm requires a time series of pavement temperatures, snow depths, and meteorological variables to determine the appropriate first treatment (time and type). Once that treatment has been implemented, the time series of pavement temperatures beyond that treatment time is invalid because of the sensitivity of the road temperature to the snow cover. The snow on the road at the time the pavement temperature time series was calculated will not exist (in the same state) after plowing and chemical treatment. These complex interactions led to a decision to implement the RCTM as a single process.

The driver process handles one site at a time. After a road temperature time series is generated, a treatment plan is developed. If the plan indicates a treatment should be applied, the processing of the site restarts from the first treatment time. The road surface state and meteorological conditions are modified to reflect the treatment action before regenerating another road temperature time series. In the MDSS it is assumed that plowing will always take place as part of a chemical treatment or abrasive application. This changes the road surface state by removing most of the existing snow.

If the RCTM recommends a chemical treatment, the suggested chemical amount is applied. The chemical algorithm then calculates the dissipation and effectiveness of the chemical based on time, traffic, and liquid equivalent precipitation amount. The treatment impacts the road state and a new road temperature time series is calculated starting from the treatment time. This iterative process continues until no treatment is required in the remaining time until the end of the forecast period. See Figure 5.2 for a schematic of this process.

The RCTM has been designed to determine road conditions and treatment plans in three main modes. The RCTM processing modes are as follows:

1) No treatment
2) Recommended treatment
3) User-defined treatment
The first mode assumes that no treatment will take place. This provides the user with information on the state of the road if no maintenance action is taken. The second mode generates a recommended treatment plan using the output from the coded Rules of Practice module (see Appendix D). The third mode generates road conditions based on a user-defined treatment plan(s). This third mode is used both for user-driven road condition calculation requests based on specified treatment plans (i.e. what-if’s), and for calculations of the road conditions under the garages’ current treatment plans. As part of its interaction with this system, each garage must regularly keep the system informed of its plan of action. These (user-specified) current treatment plans are stored on disk and used in these calculations.

5.3.3 Pavement Frost Product

The Frost Potential Algorithm was developed to describe whether or not frost is likely to occur based on weather and road condition forecasts. At its core, a frost deposition algorithm developed by Tina Greenfield (IA-DOT) while at Iowa State University is used to calculate the deposition and evolution of frost on the road. Frost may develop, then melt or sublimate. This algorithm generates a predicted frost state based on environmental conditions.

Since frost deposition is very sensitive to small changes in the environmental conditions, the core frost deposition algorithm is run several times in a row with slightly varied inputs. The goal is to use estimates of the uncertainty in the input weather to produce an estimate of the probability of frost. However, since verification of frost is difficult, this product is termed “frost potential” rather than “frost probability” since generation of a probabilistic forecast should be calibrated.

Figure 5.2. RCTM module data flow diagram. The dashed lines indicate the data flow if the Rules of Practice Module determines that a treatment is required.

Estimates of the air temperature, dew point temperature, and wind speed are varied using estimates of the standard deviation in those forecasts. Estimates of the variance in the
pavement temperature forecasts are less reliable, so those forecasts cannot be perturbed as part of this pseudo-Monte Carlo algorithm. A true Monte Carlo method would generate frost forecast estimates by performing statistically random sampling experiments. Here the sampling is not random, so the algorithm is called pseudo-Monte Carlo.

Each variation of the inputs is weighted based on its “likelihood”. That is, forecasts perturbed far from the original forecasts are given less credence because they are deemed less likely to occur. Forecasts closer to the original forecast are given more weight. In each variation, the existence of frost is saved at each forecast hour as a number between zero (no frost) and one (heavy frost on the pavement). The frost forecasts are then combined in a weighted average to generate a frost potential at each hour in the future. The resultant frost potential forecasts end up as numbers (indexed) between zero and one. Each hourly frost potential forecast is then binned into one of four MDSS alert categories reflecting the severity expected.

5.3.4 Alert Generator

The Alert Generator is a process which calculates weather and road alerts based on current observations and forecasts. Alerts are designed to be simple notifications of pre-determined hazardous weather or road states that are either occurring at the present time or will be occurring in the next few hours. They are intended to attract the attention of operators so that they can examine the condition in more detail. Alerts are generated on a route-by-route basis and are binary in nature, that is, for each route, an alert condition either exists or does not exist.

There are two types of alerts currently implemented in the MDSS server: Weather Alerts and Road Alerts. These alerts are different from the weather and road alerts calculated by the display. Weather alerts are based on precipitation reports from observation data such as METARs and provide real-time indication of potentially significant weather. A Weather Alert is generated when snow or freezing precipitation is reported at or near a specific route. Weather alerts are updated as new observation data are received. Road alerts are more predictive in nature and use the most recent road condition forecasts to determine if an alert is warranted. A Road Alert is generated for a route if the road temperature goes below freezing and precipitation is expected, or if the road is currently wet and the road temperature is expected to go below freezing. Road Alerts are updated each time a new forecast is available and are valid until the next forecast update.

5.3.5 Treatment Update Network Layer (TUNL)

The Treatment Update Network Layer is a collection of Python CGI scripts which are run by the web server in response to a request from the display. The display may request the time of the most recent type of data, request the data itself, or may ask to run a what-if scenario or select a treatment. The TUNL scripts are responsible for handling the request by running subsequent scripts or processes which have access to the MDSS data, then returning the result. The result is either some information about the data (time or filename), or the data itself.
The TUNL scripts also provide an authentication interface between the display and the MDSS server so that access to certain data is granted only to those authorized to do so. Permissions files maintained on the server are used to define the names of hosts which are allowed to perform certain restricted actions. The restricted actions are limited to the ability to select treatment plans, and view selected treatment plans. All other data requests (weather and road forecasts, RWIS data, etc.) are not restricted and hence accessible to anyone.

All MDSS data are delivered to the display via the TUNL layer except those data delivered by the THREDDS and WMS servers.

5.3.6 Thematic Real-time Environmental Distributed Data Services (THREDDS) Server

The THREDDS server provides gridded data sets to the display. In MDSS Release-5, these include radar and satellite images. THREDDS was chosen as the mechanism to provide this data for several reasons. First, THREDDS is a relatively easy to use open-source solution with growing support in the meteorological community. Second, it provides the necessary time aggregation and works with known data formats (netCDF, GINI, etc.). THREDDS also allows on-the-fly grid sectoring and sampling which improves performance and minimizes client-server round-trip time. Finally, it is envisioned that THREDDS could eventually replace the TUNL layer in providing all the MDSS data.

5.3.7 WMS Server

The WMS server provides dynamic basemap images for use by the displays. Environmental Systems Research Institute’s (ESRI) ArcIMS server was chosen as the WMS implementation for testing, but any WMS-compliant server should work equally well.

When a display is started, it requests a capabilities file from the WMS server. The capabilities file defines the available geographic datasets, their extents, and their map projections. Every time a user pans or zooms in the display, a request for a new basemap image is made to the WMS server. The request includes the geographic bounding box of the image desired, the list of available geographic datasets to include, the map projection desired, and the dimensions of the returned image. The WMS server constructs a new image from the request parameters and streams that image back to the display. The display then places the image on the screen under the other datasets.

The WMS server component may be configured specifically for an MDSS installation, or the display could be configured to point to any one of several publicly-accessible WMS services which exist on the internet. The detailed configuration of a new WMS server is beyond the scope of this document. Implementation details include, but are not limited to: the choice of WMS implementation, the definition of the HTTP interface to the server, the
choice and pre-processing of geographic datasets, the rendering styles of geographic elements, and the scale ranges at which elements are rendered.

5.3.8 MDSS Display

The MDSS display application provides an interactive display of data generated by the other MDSS components. The display application provides an interactive weather alert map, shows time series graphs of weather and road condition forecast products, shows recommended road treatments, and provides a mechanism for users to perform ‘what-if?’ scenarios with different road treatments.

The display system interfaces with a server to obtain its forecast data. These interactions are handled by a web server, which has access to the MDSS system data store. The web server invokes CGI scripts to handle the requests generated by the display. These requests are a mixture of polled and user-driven requests.

The display obtains the current time, data files and/or their status, and user permissions through this interface. It also uses the interface to pass user-defined treatment plan definitions to the server and return the resultant road condition to the display. For users in the DOT garages with permissions to set current treatment plans, the display uses the interface to update the currently selected plan for a route when the user changes the selected treatment plan.

The display also interfaces with other servers for optional datasets. These include the THREDDS server for radar and satellite datasets, the WMS server for the dynamic basemaps, and the AVL server for real-time vehicle information. These may be different servers or could be combined into a single entity.

6 SYSTEM OPERATIONS

6.1 Process Invocation

6.1.1 Schedule Driven Forecast System

Processes in the RCTM run on a schedule. The schedule is determined by examination of typical data arrival times and process run times. As much of the processing is sequential, it is necessary to allow sufficient time between process invocations to ensure that the preceding process will have finished. On the other hand, these inter-process scheduling gaps should be as short as is reasonable to ensure timelier end-to-end processing.

Rather than using a custom scheduler, the MDSS uses the UNIX system scheduling utility cron. This flexible utility is documented online in the UNIX man pages. One drawback of using cron is its temporal resolution. The finest granularity available using cron is one minute. For the MDSS system, this is not a particularly oppressive restriction since the lead times the end user expects to use are more on the order of several hours. The trade-off of
less custom development is made in exchange for slightly more latency in the system output.

6.1.2 Glue Layer

The knowledge of the file system layout lies within a relatively small piece of code known as the "glue layer". The glue layer's responsibility is to stick the system together. It creates each process' command line through its knowledge of the file system and other system resources.

When cron determines that it is time to take an action, it starts the executable listed in the schedule. For MDSS, this is typically a Python super-script whose job is to start and oversee the execution of the process. This process sets a timer to ensure that the process it invokes finishes within a reasonable length of time. The super-script then invokes another Python script specific to the scheduled task.

The job of the Python script associated with the scheduled task is to determine the command line arguments for the scheduled C/C++ module's execution. It does this through its knowledge of the layout of the file system. There exist text-based "index files" at strategic locations within the file system that contain the name of the most recently generated file of a certain type. The Python script reads the appropriate index files to obtain the required input file names. These file names will go directly onto the command line. The Python script also knows the location of the static files that are required by the C/C++ module. Their file names are also gathered and put on the command line. The Python script may also query the system to obtain the current time.

This Python script then executes the command line it has created. The process' return code is processed to determine whether it was successful in completing its task. Upon success, the script then ensures that the index files are updated with the new file names. If the process does not run to a successful completion, the index files are not updated, and other processes will not be able to use the erroneous file.

6.1.3 Display Data Requests

Processes may also be user-request driven and hence unscheduled. These requests are initiated by user actions in the display application, and then handled by a web server that invokes CGI Python scripts. The CGI scripts parse the input parameters and attempt to satisfy the request. These Python and Perl scripts may either consult the file system or invoke other scripts or C++ based programs to obtain the results. An example of the latter would be a user-defined treatment plan request. In this case, the Python script creates or gathers a number of configuration files, invokes the road condition module, monitors its progress, and returns the output.

6.2 Display Invocation
The display application runs as a child process of Java WebStart from Sun Microsystems. The user is able to invoke WebStart and the application by navigating to its URL in a web browser, supplying the URL to WebStart’s command line, starting the previously loaded application from the WebStart manager, or starting the application from a previously placed desktop shortcut. The display requires a connection to the Internet. For more information on WebStart, see 19.5.5.

7 PROCESS ERROR HANDLING

7.1 Data Logging

7.1.1 Log Files

The MDSS Forecast Subsystem scripts and processes generate output indicating their progress and exit status. This textual output is written to MDSS system log files. Examination of these log files allows the system administrator to easily verify the status of each completed process and identify where problems occurred.

All messages are logged to files in the MDSS file system, with the exception of TUNL messages which are sent to the web server’s log file(s) and display messages which are logged to Java WebStart.

7.1.2 Log File Organization

Three sets of log files are maintained for each process. One file is associated with the Python super-script to record its actions. A second log file is associated with the Python script that creates and executes the command line. The command line that is actually executed is written to this log file. This allows a developer to re-create and debug any problems. The third log file contains output from the C/C++ module itself.

7.1.3 Log File Hierarchy

The first two sets of log files are each generated by a Python script. They are responsible for logging information about their internal status as well as monitoring and reporting on their children's status. The third log file only reports on its internal status since it has no children.

Each script or process is required to write log messages indicating that it has started. It must also indicate if it has started another process and the command line associated with that invocation. It also must report on the status code returned by its child. Finally, it must log that it is done and report its status code. Specific formats exist for reporting starting and ending status. Searching the files for bad status messages is an easy means for finding problems.
7.1.4 Log File Naming Conventions

The log files are all located in a log directory. For each C/C++ executable, one set of the three log files is created every day. The date associated with the files is part of the log files' names. The logging information for all runs of a particular executable during a given day is contained in these three log files.

The names of the log files differ only in their suffix. The Python super-script's log file has an .epl suffix. The second Python layer's log file has a .pyl suffix for "Python Layer". Finally the C/C++ executable's log file has an .asc extension. As an example, the log file names for the road condition process (with recommended treatments), rc_rec_tmt, for 13 December 2006 would be:

rc_rec_tmt.20061213.epl
rc_rec_tmt.20061213.pyl
rc_rec_tmt.20061213.asc

7.1.5 Executable’s Log File Messages

The log files written by the executable have standardized output. Each line in the file starts with the time the log message was written to the file. The next word on any line in the file is required to come from the following set which describes the function of the log message.

- **Starting** Required at the start of the program.
- **Info** Used for informational messages.
- **Warning** Indicates non-fatal error conditions.
- **Error** Indicates that a fatal error occurred. Explanation of the error must be provided.
- **Ending** Required last statement before exiting. Exit status should also be indicated.

These standardized messages allow easy monitoring of the log files. Searching for the string "Error" identifies all gracefully exiting failures in the system. Searching for unmatched "Starting" and "Ending" pairs can point to non-graceful exits like segmentation violations.

7.2 Error Handling

7.2.1 Index Files

Index files exist in key locations throughout the file system. These files keep a list of all the files created by successfully completed processes. If an error occurs in the processing, the current output file which contains untrustworthy data is not registered in the index file. In this case, the file remains on the disk but is not visible to the system. The Python glue layer only looks for file names listed in an index file when trying to determine the input files to
be used on the next command line. Therefore, processes which fail have the same effect on the system as if they were never invoked.

### 7.2.2 Handling Old Data

If a process fails and the downstream process requires input of that type, the processing chain could fail. To avoid this possibility, each process must be able to handle data files that may not line up temporally with the other input data files. In this way, if a process failed, the output file from an earlier run of that process would be used to provide the required data. Though these data are older (i.e., not the most recent forecast run), they are still generally adequate for use in creating a reasonable forecast. Hence, the system will use data from a previous forecast run as long as it is valid for the appropriate time. Clearly, if the span of data in the old file does not overlap enough with the current process’ time window, problems may ensue. This may result in the system providing unreliable output. However, usually these problems can be detected and addressed early enough for an operator to effectively deal with the situation. Typically a system failure (or data outage) would be known many hours before reaching a critical stage where the system would not function properly. The operator should have plenty of time to notify users of impending problems.

A real-time system monitor is not being provided as part of the MDSS prototype. It is expected that a system monitor would be developed and implemented as part of an operational MDSS system.

### 7.2.3 Failures in Client-Server Interactions

For a variety of reasons, problems may arise in the interactions between the display (client) and the server. Missing data or other system problems may result in a FAILURE code being returned to the client application when the server is accessed. Latency or a failure in the communication link may cause the client to never receive a response from the server. Script or process failure within the server’s processing could also cause a lack of response from the server. The goal of the MDSS system is to handle all of these failure cases robustly.

Client-server failure cases that the MDSS system attempts to handle are those that result in a return code of FAILURE from the server communications (TUNL) layer. These include everything from missing data to calls to the RCTM module which time out due to a processing error. Every script in the TUNL layer contains logic to return the FAILURE code if it encounters unexpected circumstances such as missing data or system calls that do not return in a reasonable length of time. These scripts are reasonably well tested in regard to returning FAILURE to the client display.

Failure cases that may be poorly handled in the display are those that result in a partial response. For example, if data are being delivered to the display from the TUNL and the data stops mid-stream, the display will eventually time out the connection. This has the potential to leave some partially-loaded data on the display and result in an inconsistent state. These cases are extremely rare and usually indicate a problem on the web server.
8 INTERPROCESS COMMUNICATION

8.1 File System Communication

MDSS Road Weather Forecast Subsystem inter-process communication is done through the file system; that is, data flows from one program to another by reading and writing files. Multiple simultaneous readers of the same files are allowed. Index files are used to ensure that a file is ready for use by the rest of the system.

Each process obtains input data generated upstream solely through filenames specified on its command line. The process reads this input data from the specified files, processes it, and writes its output to a file. The output filename is also specified on the command line. The output file format is specified in a file whose name is specified on the command line.

8.2 File Formats

Files within the MDSS are in text or binary format. The netCDF format is the only binary format used internally within the MDSS. The netCDF format is a standard developed for scientific applications. It is widely used in the meteorological community. Information on the format can be obtained at:

http://www.unidata.ucar.edu/packages/netcdf

The netCDF format data files use the .nc extension. For reference information on software packages that may be used for manipulating or displaying netCDF data, see:

http://www.unidata.ucar.edu/packages/netcdf/software.html

8.2.1 Typical MDSS netCDF Formats

NetCDF is an extremely flexible self-defining format. Within the MDSS, only simple instances of the format are used. Typically, complex structures are avoided. Instead, the data are broken out by variable. They are stored in multidimensional arrays much like the data structures typically used in FORTRAN programming. The array dimensions are hard-wired within the format. Typically the only dimension that might need to be changed when implementing this system would be the max_sites dimension. This should be adjusted to be at least as large as the number of forecast sites.

ASCII text files that describe the format of the binary netCDF output files, exist within the system. These files, called Command Description Language (CDL) files, have a .cdl extension. These are the files which need to be edited to reflect forecast site list size when configuring the system.

8.2.2 Text File Formats

Formats for text files are ad hoc and described within this document.
8.3 Display Request Inter-process Communication

The display formats its requests in several ways. For requests via the TUNL, the syntax is that of an HTTP ‘GET,’ including form parameters: the web server address, followed by the CGI script path and name, followed by a question mark, followed by a list of key-value paired arguments delimited by ampersands. These arguments are parsed by the appropriate CGI Python script. For requests to the THREDDS server, a URL for the dataset is used in a call to the netCDF-Java API. This URL consists of the server name and path, the dataset name and some information bounding the time and geographical extent of the data being requested. The THREDDS server uses this information to return the actual data. For requests to the WMS server, an HTTP ‘GET’ URL is used. In that case, the base portion of the URL is the host, path, and service name. The form parameters include the geographic extent, desired layers, map projection, and desired image dimensions.

9 CONFIGURATION FILES

9.1 Configuration File Overview

The MDSS system uses configuration files in various formats to tailor the ultimate output of the system. The configuration files are plain-text ASCII files. In general, the default configuration files will not need to be modified. An exception to this is if the user changes the forecast site list. In this case, modifications to a number of other configuration files (and possibly CDL files) will be required to generate the new static configuration files. See the installation guide that is being provided as part of this release (on CD) for more information.

9.2 Site List Configuration Files

Site list files are used by various components of the system in order to process and output data for specific sites. The site list files are text files which contain information about known sites, one site per line. The fields in each line are separated by a semi-colon. Each line has the following format:

SITE_ID;WMO_ID;ICAO_ID;LAT;LON;ELEV;REGION;NAME;STATE;COUNTRY

where:

SITE_ID = a unique 8-digit number
WMO_ID = WMO ID number of site (-9999 if not a WMO site)
ICAO_ID = ICAO or other abbreviation for site (“----” if not known)
LAT = latitude in decimal degrees (negative for south latitude)
LON = longitude in decimal degrees (negative for west longitude)
ELEV = elevation above sea level in meters
REGION = global region defined by WMO (North America == 4)
NAME = the display name of the site
STATE = two-letter state abbreviation
COUNTRY = country name (e.g. UNITED STATES)

Note that site list files should be sorted in ascending order based on the SITE_ID, each of which must be unique. Site list files are parsed using the ";" delimiter, so there is some leeway in the size of certain fields. However it is best to make any added sites conform to existing sites as much as possible.

10 REDUNDANCY

The decision on whether to provide redundant components in the system is driven by two main factors:

- The consequences of a failure
- The cost of providing redundancy

Since the MDSS prototype is a prototype and not considered a safety-critical system, a moderately conservative approach to redundancy was considered.

Most of the input data required by the MDSS can be received in a timely fashion over the Internet with typical T1 bandwidth (~1MB/sec). The communications network infrastructure represents a large recurrent cost, with monthly payments to the communications vendor company. It is therefore not cost effective to plan on redundant Internet communications links.

The server will provide the output forecast and treatment plan data over the Internet to the end users’ displays. It is not foreseen to be cost effective to provide a duplicate Internet connection. Operational versions of the system could utilize Internet or Intranet network communications.

Complete redundancy of the road forecast subsystem can be obtained by having two duplicate processing systems (hardware/software). Unless an extremely large site configuration is used, the cost of these redundant machines should be relatively minor. Software to detect a failure and switchover to the backup machine would have to be developed.

Components of the LAN, e.g. routers and switches, have been found to be extremely reliable and it is not considered necessary to have complete redundancy. However, one extra unit of each type should be configured and ready to be substituted should failure occur.

For the machines in the system, sufficient spare equipment should be kept on hand to allow for quick replacement of the faulty machine or component.
11 DATA INGEST

No data ingest infrastructure is specified for the system. Several data sets are required by the RCTM and Display. These can be generated manually, purchased, or downloaded from the internet. This document does not specify or suggest how the input data is generated. Instead it simply specifies the interface specifications required by the MDSS components.

12 ROAD CONDITION AND TREATMENT MODULE (RCTM)

The source code for this section can be found on the MDSS CD in the /src/apps/roadcond_mdss/ directory.

12.1 Identification

roadcond_mdss

12.2 Type

Process

12.3 Purpose

The purpose of this process is to produce road (pavement surface) condition forecasts and treatment plan-related data. The road conditions depend on the treatments applied to the road. Three treatment options are available: 1) no treatment, 2) system recommended treatments from the Rules of Practice module, and 3) user-defined treatments. These forecasts are made site-by-site for each site in the list supplied to the program. The forecast road condition/surface condition state is stored as surface temperature, snow depth, pavement state (snow packed, wet, dry, chemically wet, etc.), and so on. Treatment-related data produced are the treatment plans developed or used in the program run and a time series of the chemical concentration on the road. These treatment plans and road conditions are of primary interest to the end user and are made available to the display.

12.4 Function

The RCTM reads weather condition, road state, observational history, treatment option, and site configuration data. These data are processed to produce the road condition output data of interest to the end users. Its output data files are made available to the Frost Process, Merge Variables Process and the Forecast Re-formatter for preparation before being passed to the display.
More information on the Road Temperature, Net Mobility, Chemical Concentration, and Rules of Practice modules is provided in the appendices.

12.5 Dependencies

The RCTM requires weather condition data. These weather data must be valid, i.e. non-missing, in all relevant fields at every forecast site from the road condition forecast starting time through the ending time (48 hours later). If critical variables are missing within the 48-hour time frame, `roadcond_mdss` will exit and not provide any new forecast files. In this case, the display will continue to run with its most recent forecast data.

Road characteristics (e.g., pavement material types and thicknesses, etc.) for each site are required. These characteristics must be properly parameterized for use by METRo. Additionally, road segment specific parameters such as idealized traffic level, number of lanes, route length, and typical route treatment times must be specified for each site/segment.

METRo, the road temperature model used within the RCTM, initializes its subsurface state using a history of observations at each site. An observational history must contain data for between 3 and 12 hours before the weather forecasts begin. Twelve hours is optimal. Some data may be missing, but at least one time with a complete set of observations is required. Note that these may be pseudo-observations. That is, if a forecast site does not have a subsurface sensor, an estimate of the subsurface may be used in the observational history.

12.6 Interfaces

12.6.1 Command Interface

```
% roadcond_mdss forecast_time site_list_file bridge_site_list_file cdl_file
site_configuration_file treatment_option user_def_treatment_file previous_road_cond_file
obs_history_file weather_forecast_file output_file [-d debug_level] [-l log_file]
```

where:

- `forecast_time` is the UNIX time of the first road condition forecast to be generated. This will be truncated to the top of the hour.
- `site_list_file` is the name of the text file containing the list of sites to be processed.
- `bridge_site_list_file` is the name of the text file containing the list of bridge sites to be processed. This should be a subset of `site_list_file`.
- `cdl_file` is the name of the CDL file describing the output format.
- `site_configuration_file` is the name of the configuration file containing information specific to a roadway segment. This includes
  1) the subsurface layers at each site.
  2) idealized hourly traffic levels, number of lanes, and length of the road segment.
  3) Default chemical type and other site-specific defaults.
- `treatment_option` is the desired treatment option.
0 = No Treatment
1 = Suggested Treatment to be generated from Rules of Practice
2 = User-defined treatment

user_def_treatment_file is the name of the file containing the user-defined treatment plan. This argument is ignored if treatment_option is not 2. Commonly, “None” is used for this argument if treatment_option is not 2.

previous_road_cond_file is the name of the output file from an earlier run of roadcond_mdss. A run from 3 or more hours earlier is typically used. The earlier run must be from within the previous 48 hours, and preferably within the last 12 hours. This file contains the roadway’s chemical state information.

obs_history_file is the name of the netCDF file containing the recent weather, pavement, and subsurface observations from the sites.

weather_forecast_file is the name of the weather forecast netCDF file

output_file is the name of the netCDF output file to be produced.

debug_level is the level of debugging information to be output to the log file. The default level is 0. Higher values (up to 9 will produce more debugging output).

log_file is the file to which log output should be written. If not specified, this output goes to stdout.

12.6.2 Start Script

The roadcond_mdss program is started by a Python script (in /scripts/python). The script gathers the names of the appropriate input files and static (configuration) files to create the command line. Then it runs the program using this command line. Treatment options and static data files are set within Python "wrapper" scripts called ep_rc_no_tmt.py, ep_rc_rec_tmt.py and ep_rc_cur_tmt.py for no treatment, recommended treatment, and current (selected) treatment, respectively. Each of these scripts sets arguments and then runs a script called run_proc.py, which in turn actually executes the roadcond_mdss program on the real-time data.

Roadcond_mdss can also be invoked by clients to perform what-if’s or user-defined treatments via a Python script in the TUNL. This script constructs the command line, runs the program, and passes the output file back to the display.

12.6.3 Configuration Files

12.6.3.1 Site List Configuration File

This file contains information on the output sites. The format of this file is described in Section 9.2 of this document. A sample site list file can be found on the MDSS CD under /data/static_data/site_list. The file name is road_cond_sites.asc.

12.6.3.2 Bridge Site List Configuration File
This file contains information specifying which of the output sites are bridges. The format of this file is the same as for the site list file and is described in Section 9.2 of this document. A sample bridge site list file can be found on the MDSS CD under /data/static_data/site_list. The file name is bridge_cond_sites.asc.

### 12.6.3.3 Site Configuration File

The Site Configuration file contains the subsurface characteristics at each forecast site as well as other site/segment characteristics such as traffic levels, route timing, treatment strategy, default chemicals, etc. The subsurface structure is broken down into several layers, each with an associated material type (found in the materials configuration file). Note that layers are listed from the lowest layer up. These site-specific variables should be provided based on input from the DOT.

This site configuration file is a text file. It must contain information on all the sites listed in the Site List configuration file. That is, the RCTM must be able to determine the characteristics for every site that it is tasked with handling. The configuration data is listed site-by-site. Sites should be sorted by site ID in increasing order. The format for a single site is given below. An example for one site follows the format description. A sample file with multiple sites can be found on the MDSS CD at /data/static_data/config/site_config.conf. Text on a line that appears after a # is treated as a comment.

```plaintext
SITE_ID: site_id # site ID number
TZ_CODE: time_zone # time zone
NUM_LAYERS: num_layers # number of material types in subsurface
LAYER_MAT: layer_mat_1 layer_mat_2 layer_mat_3 ... # one layer code number per material type
LAYER_THICKNESS: thick_1 thick_2 ... # thickness of each node in meters
TRAFFIC: level_1 level_2 ... level_24 # traffic levels for each hour (start at midnight)
ROUTE_LENGTH: length # length of route (miles)
ROUTE_LANES: num_lanes # total number of lanes on segment
ROUTE_TREATMENT_TIME: treat_time # time required to treat route (minutes)
TREATMENT_STRATEGY: strategy # recommended treatment strategy choice
SNOW_PLOW_THRESHOLD: thresh # min snow to trigger plow treatment (mm)
CHEM_TYPE: type # recommended chemical to be used
CHEM_FORM: form # recommended chemical form (dry, wet, etc)
CHEM_MIN_APPLICATION_RATE: min_rate # lowest recommended application rate
CHEM_MAX_APPLICATION_RATE: max_rate # highest recommended application rate
CHEM_RATE_INCREMENT: inc # treatment rate increment
PRETREAT_TYPE: pt_type # pretreatment chemical to recommend
PRETREAT_FORM: pt_form # pretreatment chemical form to recommend
PRETREAT_MIN_APPLICATION_RATE: pt_rate # lowest recommended pretreatment rate
PRETREAT_MAX_APPLICATION_RATE: pt_rate # highest recommended pretreatment rate
PRETREAT_RATE_INCREMENT: pt_inc # pretreatment rate increment
```

TZ_CODE is used to establish the local time zone for the site. The format for TZ_CODE should follow the TZ environment variable. (See the timezone `man` page for a description of the TZ variable.)

Note that there should be num_layers values provided after LAYER_MAT. The thickness of each layer should be listed after LAYER_THICKNESS. Layers are listed from the
lowest (deepest) layer upwards. There is no need to specify the material type below the developed roadbed.

Material types are:
- 91 = Concrete (cement)
- 92 = Crushed rock
- 94 = Asphalt

Recognized traffic levels are:
- 1 = Low (less than 250 vehicles per hour per lane)
- 2 = Medium (between 250 and 2000 vehicles per hour per lane)
- 3 = High (more than 2000 vehicles per hour per lane)

Recognized Treatment Strategies are:
- 0 = Trigger on Hazardous Events
- 1 = Trigger at every Route Time while storm is ongoing

Types are:
- 0 = Plow Only
- 1 = NaCl
- 2 = CaCl₂
- 3 = MgCl₂
- 4 = CaMg Acetate
- 5 = K Acetate
- 6 = Caliber
- 7 = IceSlicer
- 8 = IceBan

Note that CaMg Acetate, (K) Potassium Acetate, and IceBan are placeholders and are not yet supported.

Recognized Chemical Forms are:
- 0 = Dry
- 1 = Prewet
- 2 = Liquid

Units for Min/Max Application rates are in lbs/lane-mile for dry or pre-wet and in gal/lane-mile for liquid. The same units are used for the increment rates.

Here is a sample configuration for one site.

# IA 210 – Rural Primary
SITE_ID: 74449041
TZ_CODE: CST6CDT
NUM LAYERS: 4
LAYER MAT: 92 94
LAYER THICKNESS: 0.15 0.25
TRAFFIC: 1 1 1 1 1 1 1 2 2 2 2 2 2 2 3 3 3 2 1 1 1 1 1 1
ROUTE LENGTH: 41.2
ROUTE_LANES: 2
ROUTE_TREATMENT_TIME: 167
TREATMENT_STRATEGY: 0
SNOW_PLOW_THRESHOLD: 25.4
CHEM_TYPE: 6
CHEM_FORM: 1
CHEM_MIN_APPLICATION_RATE: 60
CHEM_MAX_APPLICATION_RATE: 200
CHEM_RATE_INCREMENT: 20
PRETREAT_TYPE: 6
PRETREAT_FORM: 2
PRETREAT_MIN_APPLICATION_RATE: 40
PRETREAT_MAX_APPLICATION_RATE: 100
PRETREAT_RATE_INCREMENT: 10

12.6.4 Inputs

12.6.4.1 Weather Forecast Data File

The weather forecast input data file is in netCDF format. The required fields are described below in the CDL file. Other fields may appear in the weather forecast file. They will be ignored.

Forc_time must be 00Z of a particular day. The forecast valid times are one hour apart from that time. The 48 hours used by the RCTM will lie somewhere within this time period – not necessarily at the start of the input data. The data start time to be used by the process is determined by the command line argument forecast_time. This file’s name is passed to the RCTM as a command line argument.

The bolded variables below in the CDL specification are required by roadcond_mdss. Additional variables are required by the display, the frost potential module, or for backward compatibility. These additional variables need not contain data but must be present in the input netCDF file. It is assumed that one weather forecast file will contain all the relevant information. The CDL file can be found on the MDSS CD at /data/static_data/cdl/mesh_derive.cdl. Sample weather forecast netCDF files can be found in /data/mesh_fcst/20070412.

The CDL file describing this netCDF file format is:

```netcdf mesh_derive {
  dimensions:
    max_site_num = 180;  // number of locations
    days = 4;           // number of days
    fc_times_per_day = 24; // fcst times per day

  variables:
    double creation_time;
    creation_time:long_name = “time at which forecast file was created”;
```
creation_time:units = "seconds since 1970-1-1 00:00:00";

double forc_time;
    forc_time:long_name = "time of earliest forecast";
    forc_time:units = "seconds since 1970-1-1 00:00:00";

int num_sites;
    num_sites:long_name = "number of actual sites";

int site_list(max_site_num);
    site_list:long_name = "forecast site list";
    site_list:_FillValue = -99999;

float T(max_site_num, days, fc_times_per_day);
    T:long_name = "temperature";
    T:units = "degrees Celsius";

float dewpt(max_site_num, days, fc_times_per_day);
    dewpt:long_name = "dewpoint";
    dewpt:units = "degrees Celsius";

float cloud_cov(max_site_num, days, fc_times_per_day);
    cloud_cov:long_name = "cloud cover";
    cloud_cov:units = "percent";

float cprob_rain(max_site_num, days, fc_times_per_day);
    cprob_rain:long_name = "conditional probability of rain";
    cprob_rain:units = "percent";

float cprob_snow(max_site_num, days, fc_times_per_day);
    cprob_snow:long_name = "conditional probability of snow";
    cprob_snow:units = "percent";

float cprob_ice(max_site_num, days, fc_times_per_day);
    cprob_ice:long_name = "conditional probability of ice";
    cprob_ice:units = "percent";

float wind_speed(max_site_num, days, fc_times_per_day);
    wind_speed:long_name = "windspeed";
    wind_speed:units = "meters per second";

float wind_speed_mph(max_site_num, days, fc_times_per_day);
    wind_speed_mph:long_name = "windspeed in mph";
    wind_speed_mph:units = "miles per hour";

float wind_dir(max_site_num, days, fc_times_per_day);
    wind_dir:long_name = "wind direction clockwise from north";
    wind_dir:units = "degrees north";

39
float rh(max_site_num, days, fc_times_per_day);
rh:long_name = "relative humidity";
rh:units = "decimal";

float rh_pct(max_site_num, days, fc_times_per_day);
rh_pct:long_name = "percent relative humidity";
rh_pct:units = "percent";

float precip_rate(max_site_num, days, fc_times_per_day);
precip_rate:long_name = "precip (SWE) rate";
precip_rate:units = "mm/hr";

float precip_rate_inches(max_site_num, days, fc_times_per_day);
precip_rate_inches:long_name = "precip (SWE) rate";
precip_rate_inches:units = "in/hr";

float precip_type(max_site_num, days, fc_times_per_day);
precip_type:long_name = "precipitation type";
precip_type:units = "0=NONE, 1=RAIN, 2=SNOW, 5=ICE";

float precip_accum_inches(max_site_num, days, fc_times_per_day);
precip_accum_inches:long_name = "3 hr precip accumulation";
precip_accum_inches:units = "inches";

float TempF(max_site_num, days, fc_times_per_day);
TempF:long_name = "temperature";
TempF:units = "degrees Fahrenheit";

float dewptF(max_site_num, days, fc_times_per_day);
dewptF:long_name = "dewpoint";
dewptF:units = "degrees Fahrenheit";

float snow_rate_inches(max_site_num, days, fc_times_per_day);
snow_rate_inches:long_name = "snowfall rate";
snow_rate_inches:units = "in/hr";

float snow_accum_inches(max_site_num, days, fc_times_per_day);
snow_accum_inches:long_name = "3 hr snowfall accumulation";
snow_accum_inches:units = "inches";

float snow_accum_total(max_site_num, days, fc_times_per_day);
snow_accum_total:long_name = "snowfall accumulation since start of forecast";
snow_accum_total:units = "mm";

float snow_accum_total_inches(max_site_num, days, fc_times_per_day);
snow_accum_total_inches:long_name = "snowfall accumulation since start of forecast";
snow_accum_total_inches:units = "inches";
```csharp
float snow_accum_48hr_total_inches(max_site_num, days, fc_times_per_day);
snow_accum_48hr_total_inches:long_name = "48hr snowfall accumulation since start of forecast";
snow_accum_48hr_total_inches:units = "inches";

float prob_precip03_pct(max_site_num, days, fc_times_per_day);
prob_precip03_pct:long_name = "probability of precipitation, 3 hr";
prob_precip03_pct:units = "percent (0-100)";

float blowing_snow_potential(max_site_num, days, fc_times_per_day);
blowing_snow_potential:long_name = "blowing snow potential";
blowing_snow_potential:units = "index (0-3) (low-high)";

float blowing_snow_pot_vals(max_site_num, days, fc_times_per_day);
blowing_snow_pot_vals:long_name = "blowing snow potential values";
blowing_snow_pot_vals:units = "floating point (0-1) (low-high)";

float P_sfc(max_site_num, days, fc_times_per_day);
P_sfc:long_name = "Pressure at 2m above sfc";
P_sfc:units = "millibars";

float dlwrf_sfc(max_site_num, days, fc_times_per_day);
dlwrf_sfc:long_name = "downward long wave radiation flux at surface";
dlwrf_sfc:units = "W/m2";

float dswrf_sfc(max_site_num, days, fc_times_per_day);
dswrf_sfc:long_name = "downward short wave radiation flux at surface";
dswrf_sfc:units = "W/m2";
}

12.6.4.2 Previous Road Conditions Forecast Data File

If available, the netCDF data file containing road conditions from a previous roadcond_mdss run should be used. Otherwise the filename “None” should be used for the previous road conditions data file. The snow depth and the state of the chemicals on the road are read in. The CDL file describing this netCDF file format is the same as the output file described in the next section. This file’s name is passed to the RCTM as a command line argument.

12.6.4.3 Observations History File

The road temperature model, METRo, requires an observational history at each forecast site. METRo uses these observations to derive an initial estimate of the current road subsurface state. At least three hours of history are required. Twelve hours of history are optimal. At least
```
one of the hours in the history must have a complete set of observational data. More information can be found online in the METRo documentation.

The observation history CDL file can be found on the MDSS CD at /data/static_data/cdl/obs_history.cdl. Sample observation history netCDF files are located under /data/obs_history/20070412.

The CDL file describing this netCDF file format is:

```netcdf obs_history {

dimensions:
    max_site_num = 25 ;
    num_times = 48 ;

variables:
    double creation_time ;
    creation_time:long_name = "time at which forecast file was created" ;
    creation_time:units = "seconds since 1970-1-1 00:00:00" ;

    double obs_time ;
    obs_time:long_name = "time of earliest obs" ;
    obs_time:units = "seconds since 1970-1-1 00:00:00" ;

    int num_sites ;
    num_sites:long_name = "number of actual_sites" ;

    int site_list(max_site_num) ;
    site_list:long_name = "forecast site id numbers" ;

    int type ;
    type:long_name = "cdl file type" ;

    float T(max_site_num, num_times) ;
    T:long_name = "2m air temperature" ;
    T:units = "degrees Celsius" ;

    float dewpt(max_site_num, num_times) ;
    dewpt:long_name = "dew point temperature" ;
    dewpt:units = "degrees Celsius" ;

    float wind_speed(max_site_num, num_times) ;
    wind_speed:long_name = "windspeed" ;
    wind_speed:units = "meters per second" ;

    int Precip(max_site_num, num_times) ;
    Precip:long_name = "Presence of Precip" ;
    Precip:units = "0=None, 1=Precip" ;

    float road_T(max_site_num, num_times) ;
    road_T:long_name = "road surface temperature" ;
    road_T:units = "degrees Celsius" ;

    float bridge_T(max_site_num, num_times) ;
    bridge_T:long_name = "bridge surface temperature" ;
    bridge_T:units = "degrees Celsius" ;

    int road_condition(max_site_num, num_times) ;
    road_condition:long_name = "road condition" ;
    road_condition:units = "33=dry, 34=wet" ;

    float subsurface_T(max_site_num, num_times) ;
    subsurface_T:long_name = "subsurface temperature" ;
}
12.6.4.4 User-Defined Treatment Plan File

If the treatment option on the command line is set to “user-defined treatment”, a file is read that contains treatment specifications. Generally speaking, this treatment option can be used for several purposes. Operationally, it is used for what-if scenarios generated by the display and also used to generate the outcome of the garage’s current plan for a specific route. It can also be used in testing by creating a treatment plan by hand.

The treatment plan is specified in the following way. The site and number of treatments are specified. Then each of the treatments is specified on one line. They should appear in chronological order. The format is:

Site_id
Num_treatments
Treatment_time Application_rate Chem_type Chem_form
...
Treatment_time Application_rate Chem_type Chem_form

Treatment_time is the UNIX time of the treatment. Application_rate is the application rate in either gallons or lbs per lane mile. Chem_type is the chemical type. Chem_form is the form of the chemical.

The Application_rate (gallons vs lbs per lane-mile) is interpreted according to the chemical form. The enumerations of Chem_type and Chem_form are given in the next section.

Here is a sample user-defined treatment file which describes a treatment plan including a liquid pretreatment and dry chemical application six hours later.

72469108
2
1124478000 90 1 2
1124499600 200 1 0

12.6.5 Output

The road conditions data file is in netCDF format. Many derivative fields, e.g. unit converted fields, are calculated for the display to lessen its complexity. Note that the output data starts at 00Z of a particular day. The 48 hours of RCTM data will be surrounded by missing data. The desired output file’s name is passed to the roadcond_mdss as a command line argument. A
sample CDL file can be found on the MDSS CD at /data/static_data/cdl/road_cond.cdl. Sample road condition netCDF files can be found under /data/rc_rec_tmt/20070412.

The CDL file describing this netCDF file format is:

```plaintext
netcdf road_cond {
    dimensions:
        days = 4; // number of days
        fc_times_per_day = 24; // fcst times per day
        max_site_num = 25; // number of locations (set larger than site list length)
        max_str_len = 80; // max length for treatment explanation string
        num_times = 48; // number of times road conditions are computed

    variables:
        double creation_time;
            creation_time:long_name = "time at which forecast file was created";
            creation_time:units = "seconds since 1970-1-1 00:00:00";
        double forc_time;
            forc_time:long_name = "time of earliest forecast";
            forc_time:units = "seconds since 1970-1-1 00:00:00";
        int num_sites;
            num_sites:long_name = "number of actual_sites";
        int site_list(max_site_num);
            site_list:long_name = "forecast site id numbers";
        int type;
            type:long_name = "cdl file type";

    // Start of forecast variables //
        float application_rate(max_site_num, days, fc_times_per_day);
            application_rate:long_name = "chemical application rate";
            application_rate:units = "lb/mile or gal/mile";
            application_rate:reference = "units depend on chem_form";
        float apply_chem(max_site_num, days, fc_times_per_day);
            apply_chem:long_name = "apply chemicals";
            apply_chem:values = "0 or 1";
        float available_H2O(max_site_num, days, fc_times_per_day);
            available_H2O:long_name = "water available for chemical dilution";
            available_H2O:units = "lb/ft2";
        float available_chem(max_site_num, days, fc_times_per_day);
            available_chem:long_name = "pure de-icing chemicals on road";
```
available_chem:units = "lb/ft2";

float chem_form(max_site_num, days, fc_times_per_day);
chem_form:long_name = "chemical form";
chem_form:value0 = "Dry";
chem_form:value1 = "Prewet";
chem_form:value2 = "Liquid";

float chem_type(max_site_num, days, fc_times_per_day);
chem_type:long_name = "chemical type";
chem_type:value0 = "Not set";
chem_type:value1 = "NaCl";
chem_type:value2 = "CaCl2";
chem_type:value3 = "MgCl2";
chem_type:value4 = "CaMg Acetate"; // Not Supported
chem_type:value5 = "K Acetate"; // Not Supported
chem_type:value6 = "Caliber";
chem_type:value7 = "IceSlicer";
chem_type:value8 = "IceBan"; // Not Supported

float chemical_concentration(max_site_num, days, fc_times_per_day);
chemical_concentration:long_name = "chemical concentration";
chemical_concentration:units = "percent";

float do_plowing(max_site_num, days, fc_times_per_day);
do_plowing:long_name = "do plowing";
do_plowing:values = "0 or 1";

float mobility(max_site_num, days, fc_times_per_day);
mobility:long_name = "net mobility";
mobility:values = "0.0 to 1.0";

float nominal_chem(max_site_num, days, fc_times_per_day);
nominal_chem:long_name = "theoretical chem concentration";
nominal_chem:units = "percent";

float phase_type(max_site_num, days, fc_times_per_day);
phase_type:long_name = "road water phase";
phase_type:value0 = "Dry";
phase_type:value1 = "Wet";
phase_type:value2 = "Chemically wet";
phase_type:value3 = "Chemically ice";
phase_type:value4 = "Slush";
phase_type:value5 = "Snow";
phase_type:value6 = "Ice";

float precip_type(max_site_num, days, fc_times_per_day);
p precip_type:long_name = "precip type on road";
p precip_type:value0 = "None";
precip_type: value1 = "Rain";
precip_type: value2 = "Snow";
precip_type: value3 = "Mixed rain/snow";
precip_type: value4 = "Mixed snow/rain";
precip_type: value5 = "Freezing rain"

float subsurface_T(max_site_num, days, fc_times_per_day);
subsurface_T: long_name = "road subsurface temperature (40cm) ";
subsurface_T: units = "degrees Celsius"

float road_T(max_site_num, days, fc_times_per_day);
road_T: long_name = "road surface temperature, C";
road_T: units = "degrees Celsius"

float road_TempF(max_site_num, days, fc_times_per_day);
road_TempF: long_name = "road surface temperature, F";
road_TempF: units = "degrees Fahrenheit"

float snow_depth(max_site_num, days, fc_times_per_day);
snow_depth: long_name = "snow depth on road, mm"
snow_depth: units = "mm"

float snow_depth_inches(max_site_num, days, fc_times_per_day);
snow_depth_inches: long_name = "snow depth on road, inches"
snow_depth_inches: units = "in"

float solution_type(max_site_num, days, fc_times_per_day);
solution_type: long_name = "chemical solution type"
solution_type: value0 = "Not set"
solution_type: value1 = "NaCl"
solution_type: value2 = "CaCl2"
solution_type: value3 = "MgCl2"
solution_type: value4 = "CaMg Acetate"; // Not Supported
solution_type: value5 = "K Acetate"; // Not Supported
solution_type: value6 = "Caliber"
solution_type: value7 = "IceSlicer"
solution_type: value8 = "IceBan"; // Not Supported

char treatment_explanation(max_site_num, days, fc_times_per_day, max_str_len);
treatment_explanation: long_name = "treatment explanation string"

float treatment_time(max_site_num, days, fc_times_per_day);
treatment_time: long_name = "offset from current hour to apply treatment"
treatment_time: units = "hour"
treatment_time: values = "-1 to 48"

data:

type = 2;
}
12.7 Processing

The roadcond_mdss processing consists of initialization steps followed by a loop, which processes the road conditions for every site. At the end of each iteration through the site loop, the road condition data are stored in arrays formatted for writing to the output file. Once the site loop has completed, the data arrays are written to the output file as shown schematically in Figure 12.1.

Figure 12.1. Roadcond_mdss processing for any given site.
1. Read the site list file and set up associated data structures.
2. Read the traffic and road subsurface configuration data file.
3. Read the observational history file and create observational history time series.
4. Read in the previous roadcond_mdss output file (if available) snow depth and chemical attributes.
5. Read the weather forecast data file and create the weather data time series structure.
6. Create the output file and read parameters set within the output CDL file.
7. Write the forecast time, site list, etc. to the output file.
8. Allocate and initialize output data array space.

In the loop over the sites, the following operations are performed:

1. Initialize data values for this site. Extract site-specific weather, traffic, subsurface temperature profile, etc.
2. Begin loop over all forecast hours
   2.1 Calculate Road Temperature and Snow Depth
      2.1.1 Create METRo input files from weather and observational history data.
      2.1.2 Run METRo externally, wait for its completion.
         See Appendix A.
      2.1.3 Parse the METRo output file to obtain road temperature and snow depth time series.
   2.2 Calculate the chemical concentration on road surface based on weather and road conditions (see Appendix B).
   2.3 Create storm characteristics summary.
   2.4 Calculate the mobility index at each forecast lead time (see Appendix C).
   2.5 Determine if more treatments are required.
      2.5.1 If no treatment option is selected, no more treatments are performed.
      2.5.2 If user-defined treatments are selected, read treatment list to determine the next treatment (if any).
      2.5.3 If suggested treatment option is selected, the Rules of Practice module determines if another treatment is required (see Appendix D).
   2.6 If plow-only or chemical application.
      2.6.1 Remove snow from road due to plowing.
      2.6.2 Calculate the chemical concentration on road surface due to any treatments (see Appendix B).
      2.6.3 Calculate Road Temperature and Snow Depth (as in 2.1.x above)
      2.6.4 Calculate the mobility index at each forecast lead time (see Appendix C).
3. If we have more forecast hours to examine, go to top of loop for forecast hours to check for more treatments. This next pass through the loop starts at the end of the last examination period and overwrites existing data beyond that time.
4. Store this site’s data in arrays to be written to the output data file.

After the site loop has been completed, the following step is performed:
1. Set the forecasted phase of the water/solution on the roadway at all time periods.
2. Convert any liquid treatments from dry to liquid units.
3. Write the output data arrays to the output file and close the file.
4. Exit with successful status

13 Frost Potential Algorithm
The source code for this section can be found on the MDSS CD in the /src/apps/frost/ directory.

13.1 Identification
frost

13.2 Type
Process

13.3 Purpose
The purpose of the Frost Potential Algorithm is to generate a severity index describing the potential for frost to occur given the forecast pavement and weather conditions. The algorithm is designed to address the uncertainty in the weather predictions for specific variables. These frost potential forecasts are made site-by-site for each site supplied to the program. The output includes a number between zero and one describing the frost potential. Larger numbers are indicative of a situation where frost is more likely to exist on the road. These values are used to derive categorical frost potential forecasts ranging from “None” to “High”.

13.4 Function
The frost deposition process reads weather and pavement state forecasts as well as site configuration data. These data are processed to produce the frost potential output data. The categorized frost potential output is destined for the display application. Its output data files are made available to the Merge Variables Process and the Forecast Re-formatter for preparation before being passed to the display.

13.5 Dependencies
The frost deposition module requires weather forecasts. It reads in and uses the air temperature, dew point temperature, wind speed, precipitation rate, and surface pressure forecasts. It also requires the pavement surface temperature forecasts generated by roadcond_mdss. These forecast data must be valid, i.e. non-missing, in all relevant fields at
every forecast site from the road condition forecast starting time through the ending time (48 hours later).

The site configuration data are required to determine the thermal properties of the pavement surface material at each site. These materials are typically either asphalt or concrete.

13.6 Interfaces

13.6.1 Command Interface

% frost forecast_time site_list cdl_file site_configuration_file weather_forecast_file
road_forecast_file output_file [-d debug_level] [-l log_file]

where:
forecast_time is the UNIX time of the first road condition forecast to be generated.
This will be truncated to the top of the hour.
site_list is the name of the text file containing the list of sites to be processed.
cdl_file is the name of the CDL file describing the output format
site_configuration_file is the name of the configuration file containing information
specific to roadway segments.
weather_forecast_file is the name of the weather forecast netCDF input file
road_forecast_file is the name of the netCDF output file from roadcond_mdss
output_file is the name of the netCDF output file to be produced.
default debug_level is the level of debugging information to be output to the log file. The
default level is 0.
log_file is the file to which log output should be written.

13.6.2 Start Script

The frost program is started by a Python script which gathers the names of the appropriate
input files, static (configuration) files, and then runs the program. Forecast time and static
data files are set within Python "wrapper" scripts called ep_rc_frost.py and ep_bc_frost.py
for roadway and bridge frost calculations, respectively. Each of these scripts sets arguments
and then runs a script called run_proc.py, which in turn actually executes the frost program
on the real-time data.

Frost can also be invoked by the user-defined Python script in the TUNL. This script
constructs the command line, runs the program, and passes the output file back to the display.
13.6.3 Configuration Files

13.6.3.1 Site List Configuration File

This file contains information on the output sites. The format of this file is described in Section 9.2 of this document.

13.6.3.2 Site Configuration File

See Section 12.6.3.3 of this document.

13.6.4 Inputs

13.6.4.1 Weather Forecast Data File

See Section 12.6.4.1 of this document.

13.6.4.2 Road Forecast File

This file contains the output from roadcond_mdss and is described in Section 12.6.5 of this document.

13.6.5 Output

The frost potential output is in netCDF format. Many of the output data fields are copied from the input file. This includes the fields that are required by the display. Note that the output data starts at 00Z of a particular day. The 48 hours of frost output data will be surrounded by missing data. The desired output file’s name is passed to frost as a command line argument. A sample CDL file can be found on the MDSS CD at /data/static_data/cdl/frost.cdl. Sample frost netCDF files can be found under /data/rc_frost/20070412.

The CDL file describing this netCDF file format is:

```ncdf frost {
dimensions:
    days = 4;    // number of days
    fc_times_per_day = 24; // fcst times per day
    max_site_num = 25;  // number of locations
    max_str_len = 80; // max length for strings
    num_times = 48; // number of times road conditions are computed
variables:
    double creation_time;
    creation_time:long_name = "time at which forecast file was created";
    creation_time:units = "seconds since 1970-1-1 00:00:00" ;
```
double forc_time;
  forc_time:long_name = "time of earliest forecast";
  forc_time:units = "seconds since 1970-1-1 00:00:00";

int num_sites;
  num_sites:long_name = "number of actual sites";

int site_list(max_site_num);
  site_list:long_name = "forecast site id numbers";

int type;
  type:long_name = "cdl file type";

// Forecast variables
float application_rate(max_site_num, days, fc_times_per_day);
  application_rate:long_name = "chemical application rate";
  application_rate:units = "lb/mile or gal/mile";
  application_rate:reference = "units depend on chem_form";

float apply_chem(max_site_num, days, fc_times_per_day);
  apply_chem:long_name = "apply chemicals";
  apply_chem:values = "0 or 1";

float available_H2O(max_site_num, days, fc_times_per_day);
  available_H2O:long_name = "water available for chemical dilution";
  available_H2O:units = "lb/ft2";

float available_chem(max_site_num, days, fc_times_per_day);
  available_chem:long_name = "pure de-icing chemicals on road";
  available_chem:units = "lb/ft2";

float chem_form(max_site_num, days, fc_times_per_day);
  chem_form:long_name = "chemical form";
  chem_form:value0 = "Dry";
  chem_form:value1 = "Prewet";
  chem_form:value2 = "Liquid";

float chem_type(max_site_num, days, fc_times_per_day);
  chem_type:long_name = "chemical type";
  chem_type:value0 = "Not set";
  chem_type:value1 = "NaCl";
  chem_type:value2 = "CaCl2";
  chem_type:value3 = "MgCl2";
  chem_type:value4 = "CaMg Acetate"; // Not Supported
  chem_type:value5 = "K Acetate";    // Not Supported
  chem_type:value6 = "Caliber";
  chem_type:value7 = "IceSlicer";
  chem_type:value8 = "IceBan";      // Not Supported
float chemical_concentration(max_site_num, days, fc_times_per_day);
chemical_concentration:long_name = "chemical concentration";
chemical_concentration:units = "percent";

float do_plowing(max_site_num, days, fc_times_per_day);
do_plowing:long_name = "do plowing";
do_plowing:values = "0 or 1";

float frost_potential(max_site_num, days, fc_times_per_day);
frost_potential:long_name = "potential for frost on road";
frost_potential:values = "0.0 to 1.0";

float frost_potential_index(max_site_num, days, fc_times_per_day);
frost_potential_index:long_name = "potential for frost on road";
frost_potential_index:value0 = "None";
frost_potential_index:value1 = "Low";
frost_potential_index:value2 = "Medium";
frost_potential_index:value3 = "High";

float mobility(max_site_num, days, fc_times_per_day);
mobility:long_name = "net mobility";
mobility:values = "0.0 to 1.0";

float nominal_chem(max_site_num, days, fc_times_per_day);
nominal_chem:long_name = "theoretical chem concentration";
nominal_chem:units = "percent";

float phase_type(max_site_num, days, fc_times_per_day);
phase_type:long_name = "road water phase";
phase_type:value0 = "Dry";
phase_type:value1 = "Wet";
phase_type:value2 = "Chemically wet";
phase_type:value3 = "Chemically ice";
phase_type:value4 = "Slush";
phase_type:value5 = "Snow";
phase_type:value6 = "Ice";

float precip_type(max_site_num, days, fc_times_per_day);
precip_type:long_name = "precip type on road";
precip_type:value0 = "None";
precip_type:value1 = "Rain";
precip_type:value2 = "Snow";
precip_type:value3 = "Mixed rain/snow";
precip_type:value4 = "Mixed snow/rain";
precip_type:value5 = "Freezing rain";

float road_T(max_site_num, days, fc_times_per_day);
road_T:long_name = "road surface temperature, C";
The frost processing consists of initialization steps followed by a loop, which generates the frost potential for every site. At the end of each site’s frost calculation, the frost potential data is formatted for writing to the output file. Once the entire site loop has completed, the data arrays are written to the output file. The processing flow is described here and shown schematically in Figure 13.1.
1. Initialize

2. “Monte Carlo” Loop
   - 2.1 Initialize Weather Variable Perturbations
   - 2.2 Determine Perturbation Weight
   - 2.3 Calculate Frost Deposition Time Series
   - 2.4 Update Weighted Sum of Frost Potential

3. Another Perturbation??
   - Yes
   - 4. Normalize and Store this Site’s Frost Potential Times Series
   - No

Figure 13.1 Frost algorithm processing for any given site.
Flow:

1. Read the site list file and set up associated data structures.
2. Read the weather forecast data file and create the weather data time series structure.
3. Read the road temperature forecast data file and create the required data time series structure.
4. Read the site configuration file to determine pavement surface type.
5. Read in the parameter file containing information describing the step sizes and number of iterations to be used in generating “Monte Carlo” permutations.
6. Create the output file and read parameters set within the output CDL file.
7. Write the forecast time, site list, etc. to the output file.
8. Allocate and initialize output data array space.

In the loop over the sites, the following operations are performed:

1. Initialize data values for this site. Extract site-specific weather and pavement surface temperature time series.
2. Begin loop over perturbations to forecast time series.
   2.1 Calculate perturbed time series.
      2.1.1 Calculate offsets from forecast values.
      2.1.2 Add offsets to forecast values.
   2.2 Calculate the weight associated with this perturbation.
      2.2.1 Calculate perturbation likelihood for each perturbed variable.
      2.2.2 Less likely perturbations get less weight.
      2.2.3 Multiplicative combination of likelihoods.
   2.3 Calculate frost deposition time series.
      2.3.1 Apply Tina Greenfield’s (Iowa State University) frost deposition model to forecast time series.
      2.3.2 Calculate frost potential by applying interest map to frost depth at each time step.
   2.4 Update frost potential weighted sum.
      2.4.1 Multiply perturbation weight by frost potential at each forecast time.
      2.4.2 Add to existing frost potential weighted sum at each forecast time.
      2.4.3 Accumulate total weight used thus far.
3. Repeat for all specified iterations.
   4.1 Normalize frost potential time series by dividing by total weight.
   4.2 Update output data arrays.

After the site loop has been completed, the following steps are performed:

1. Write the output data arrays to the output file and close the file.
2. Exit with successful status.
14 Alert Generator

The source code for this section can be found on the MDSS CD in the /src/apps/alert/ directory.

14.1 Identification

alert

14.2 Type

Process

14.3 Purpose

The purpose of the Alert Generator is to generate notifications about important events which might impact operations. They provide operators with a quick visual cue regarding near-term weather threats.

14.4 Function

The Alert Generator uses the most recent observation and forecast weather and road data to generate an alert status for a set of road segments. Weather and road conditions which result in an alert include the existence of frozen or freezing precipitation and wet roads with temperatures below freezing. Alerts are updated when new observation or forecast data are available.

14.5 Dependencies

The Alert Generator requires observations of current weather and forecast data of road temperature and road state. Observations of current weather near the road segments are used to examine whether frozen or freezing precipitation is occurring. If such precipitation exists at a location close to a route, then a Weather Alert is generated for that route. Currently, METAR reports are used to provide the weather conditions near road segments. One or more METARs may be assigned to each route so that the next METAR in the list can be checked if the first one is missing. For Road Alerts, the most recent road forecast is examined to determine if the road will be freezing with precipitation over the next several hours. Road Alerts are also generated based on a list of routes.
14.6 Interfaces

14.6.1 Command Interface

% alert forecast_time site_list_file metar_nbr_file obs_file input_rc_file output_file [-d debug_level] [-l log_file]

where:

forecast_time is the UNIX time of the first road condition forecast to be generated.
site_list_file is the name of the text file containing the list of sites to be processed.
metar_nbr_file is the name of the file containing nearest METAR(s) to each site.
obs_file is the name of the netcdf file containing the weather observations.
input_rc_file is the name of the netcdf road condition forecast data file.
output_file is the name of the output file which will contain the alerts.
debug_level is the level of debugging information to be output to the log file. The default level is 0. Higher values (up to 9 will produce more debugging output).
log_file is the file to which log output should be written. If not specified, this output goes to stdout.

14.6.2 Start Script

The alert program is started by a Python script (in /scripts/python). The script gathers the names of the appropriate input files and static (configuration) files to create the command line. Then it runs the program using this command line. Forecast time and static data files are set within the Python “wrapper” script called ep_alert.py. This script sets arguments and then runs a script called run_proc.py, which in turn actually executes the alert program on the real-time data.

14.6.3 Configuration Files

14.6.3.1 Site List Configuration File

This file contains information on the output sites. The format of this file is described in Section 9.2 of this document.

14.6.3.2 METAR Neighbor Configuration File

This file is an ASCII text file. Each line contains the information about one site and its neighboring METARs. At the start of the line, the site ID is listed, followed by a colon. After that, the ID of each of the neighboring METARs is listed. Commas separate the METAR site IDs. A maximum of 5 neighbors is currently allowed. This can be modified within the source code.
An example of a few lines from a METAR Neighbor Configuration file:

72469100:72570021
72469101:72469020
72469102:72469017,72469005,72565000,72469020

A sample neighbor configuration file can be found on the MDSS CD at /data/static_data/config/obs_nbrs.txt. Known METAR IDs can be found in the file /data/static_data/site_list/metar_list.asc.

### 14.6.4 Inputs

#### 14.6.4.1 Observation Data File

The observation data file contains one day’s data for all sites. It is described in the cdl file listed below. Very few of the fields in the observation file are used in the generation of the weather alerts.

```plaintext
netcdf int_obs {   // netCDF definition for int_obs file

  dimensions:
    recNum = UNLIMITED;
    times_per_day = 24;
    daily_time = 1;

  variables:
    int type;
    type:long_name = "cdl file type";

    double creation_time;
    creation_time:long_name = "time at which file was created";
    creation_time:units = "seconds since 1970-1-1 00:00:00";

    double time_nominal(times_per_day);
    time_nominal:long_name = "observation time";
    time_nominal:units = "seconds since 1970-01-01 00 UTC";

    int num_sites;
    num_sites:long_name = "number of actual_sites";

    int site_list(recNum);
    site_list:long_name = "site id number";

    float cprob_snow(recNum, times_per_day);
    cprob_snow:long_name = "conditional probability of snow";
    cprob_snow:units = "decimal";
}
```
float cprob_ice(recNum, times_per_day);
cprob_ice:long_name = "conditional probability of ice";
cprob_ice:units = "decimal";

float prob_precip01(recNum, times_per_day);
prob_precip01:long_name = "probability of precipitation, 1 hr";
prob_precip01:units = "decimal";

data:
    type = 6;
}

14.6.4.2 Road Condition Forecast Data File

The road conditions forecast data is typically generated by roadcond_mdss. It should be in the format described in Section 12.6.5.

14.6.5 Output

The alert output file is in ASCII utilizing XML tags. For weather alerts, the observed precipitation type, the observation time, and location information are included. The road alerts contain information describing the predicted hazard time and the expected road state. If there is no alert for a given site at the current time, it does not need to be included in the output alert file. An example output file:

```
<Alert>
    <Site siteId="72469074">
        <RoadAlert time="1177430400" phaseType="Chemically Treated Ice"/>
        <ObsAlert time="1177430400" precipType="Snow" location="72469017"/>
    </Site>
    <Site siteId="72469075">
        <RoadAlert time="1177430400" phaseType="Chemically Treated Ice"/>
        <ObsAlert time="1177430400" precipType="Snow" location="72469017"/>
    </Site>
    <Site siteId="72469081">
        <ObsAlert time="1177430400" precipType="Snow" location="72469017"/>
    </Site>
    <Site siteId="72469099">
        <RoadAlert time="1177430400" phaseType="Snow"/>
    </Site>
</Alert>
```

14.7 Processing

Weather and road condition alerts are generated in the same process. Since Weather Alerts are based on observations, these are run once an hour after the hour’s METARs have been processed. In the future, the alerts could be run more frequently to catch METAR specials
in a more timely fashion. The alerts are generated using a site list which identifies the set of routes for the alerts and their neighboring METARs to examine. More than one METAR may be listed for each route, and the worst case report is used. If freezing or frozen precipitation is reported at any of the neighboring METARs, then an alert is generated. Since an alert is triggered if any of the neighboring METARs observes frozen precipitation, a site’s METAR-neighbor configuration should not contain too distant METARs.

Road Alerts are generated based on the road forecasts made in the forecast cycle runs. A site list is used to identify which routes to examine for alert status. For each route, the forecast for that site is examined. If the road is expected to freeze and precipitation is expected, or if the road is already wet and expected to freeze, then an alert is generated. Even though forecasts are currently only generated once every three hours, these road condition alerts are updated whenever the more frequent weather alert processing takes place. When that occurs, the next 3 hours of the forecast are considered in the road condition alert process.

15 THREDDS Data Server

15.1 Identification

Thematic Realtime Environmental Distributed Data Services; a free product available from Unidata.

15.2 Type

Process

15.3 Purpose

The purpose of the THREDDS data server is to serve gridded data (satellite and radar) to clients as requested.

15.4 Function

THREDDS runs on a host which contains satellite and radar data in netCDF format. The data are projected to a latitude-longitude grid to simplify the display process and speed access. Requests for data are received and processed by the server which returns the requested data as an array of values.

15.5 Dependencies

Satellite images are received from NOAAPort in GINI format and converted to latitude-longitude grids in netCDF format. Several different geographic scales and satellite channels
are processed and made available to the server. The client requests a specific channel and scale it wants based on the current view area.

Radar data are received from Meteorlogix and converted to netCDF format. The radar products are a mosaic of all the US NEXRAD radars projected on a 1-km grid. This product contains dBZ levels for three different types of precipitation: liquid, frozen and mixed. Note that the original data is a commercial product from Meteorlogix. If users do not have access to this product, they can still create their own radar data source by following the netCDF format guidelines presented later in this section.

THREDDS processes are Java servlets which run under the Apache Tomcat web server. Hence Java and the Apache Tomcat web server must be installed in the server host. Details are available from http://www.unidata.ucar.edu/projects/THREDDS.

15.6 Interfaces

15.6.1 Command Interface

THREDDS is started when the Tomcat web server is started. A simple start-up script which starts the server is located at $TOMCAT_ROOT/bin/startup.sh, where $TOMCAT_ROOT is the top level directory where the server is installed. Since the server should always be running, it is advisable to install this command in one of the “rc file” start-up scripts.

15.6.2 Catalog Files

The THREDDS data server uses a set of files for cataloging the data which it will make available. Catalog files are located in $TOMCAT_ROOT/content/thredds and are composed of XML commands. The main catalog file (catalog.xml) defines the set of data which will be served out. Below is an example of a catalog file.

```xml
<?xml version="1.0" encoding="UTF-8"?
<catalog name="THREDDS Server Default Catalog : You must change this to fit your server!"
    xmlns="http://www.unidata.ucar.edu/namespaces/thredds/InvCatalog/v1.0"
    xmlns:xlink="http://www.w3.org/1999/xlink">

    <service name="thisDODS" serviceType="OpenDAP" base="/thredds/dodsC/" />

    <catalogRef xlink:title="Dec IR Conus Satellite Aggregation"
        xlink:href="decConusIRAgg.xml" name="Dec IR Conus Satellite Aggregation"/>
    <catalogRef xlink:title="Dec IR East Satellite Aggregation"
        xlink:href="decEastIRAgg.xml" name="Dec IR East Satellite Aggregation"/>
    <catalogRef xlink:title="Dec IR West Satellite Aggregation"
        xlink:href="decWestIRAgg.xml" name="Dec IR West Satellite Aggregation"/>
    <catalogRef xlink:title="Dec VIS Conus Satellite Aggregation"
        xlink:href="decConusVISAgg.xml" name="Dec VIS Conus Satellite Aggregation"/>
</catalog>
```
The catalog file lists the set of possible dataset aggregation names in each of the “catalogRef” tags. An aggregation is the set of files available over a time period for that particular dataset. In each catalog reference, a title for the dataset is given, along with the location of the XML file which contains additional detail about that dataset. For example, the first entry in this list, decConusIRAgg.xml contains the following content.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<catalog xmlns="http://www.unidata.ucar.edu/namespaces/thredds/InvCatalog/v1.0"
  xmlns:xlink="http://www.w3.org/1999/xlink"
  name="THREDDS-IDD OPeNDAP Data Server" version="1.0.1">
  <service name="myDods" serviceType="OpenDAP" base="/thredds/dodsC/">
    <datasetRoot path="ignored" location="content/decIR/"/>
  </service>
  <dataset name="SUPER-NATIONAL_latlon IR Aggregation" ID="superIR"
    urlPath="data/decIR">
    <serviceName>myDods</serviceName>
    <netcdf xmlns="http://www.unidata.ucar.edu/namespaces/netcdf/ncml-2.2">
      <aggregation dimName="time" type="joinNew" recheckEvery="5 min">
        <variableAgg name="micron11"/>
        <scan dateFormatMark="SUPER-NATIONAL_latlon_IR_#yyyyMMdd_HHmm"
          location="/d2/dicast/st/dec_data/satellite/IR/SUPER-NATIONAL_latlon/"
          suffix=".nc"/>
      </aggregation>
    </netcdf>
  </dataset>
</catalog>
```

The details in this file include the dataset name, data file location, file format, variable naming, and time aggregation settings. The “recheckEvery” variable specifies how frequently the THREDDS server should rescan the directory for new files. Also, the “dateFormatMark” variable specifies the filename syntax and how date and time information is encoded in the filename.
15.6.3 Input

Input data for the THREDDS server can be in netCDF, GINI, GRIB, HDF and other formats. For simplicity however, all data served in the MDSS Release-5, are in netCDF format and projected onto a latitude-longitude grid. The processes which perform these steps are beyond the scope of this document, however, the end formats are described below.

15.6.3.1 Radar Data

Radar data are updated every five minutes and cover the entire CONUS domain at 1km resolution. (Update frequency and coverage depends on the radar data provider.) Data values contained in the “dbz” variable below fall in 3 ranges which correspond to the type of precipitation occurring. Values 1-15 denote increasing levels of rain, 17-31 denote freezing or mixed precipitation, and 33-47 denote snow. The structure of a netCDF radar data file is shown below.

```netcdf
USRA.cdl {

dimensions:
   lon = 5120 ;
   lat = 5760 ;
   enc = 49 ;
   str_len = 100 ;

variables:
   double reftime ;
      reftime:long_name = "reference time" ;
      reftime:units = "seconds since 1970-1-1 00:00" ;
   double valtime ;
      valtime:long_name = "valid time" ;
      valtime:units = "seconds since 1970-1-1 00:00" ;
   char grid_type(str_len) ;
      grid_type:long_name = "grid type" ;
   int Nlo ;
      Nlo:long_name = "number of points along longitude" ;
   int Nla ;
      Nla:long_name = "number of points along latitude" ;
   float La1 ;
      La1:long_name = "latitude of first grid point" ;
      La1:units = "degrees_north" ;
   float Lo1 ;
      Lo1:long_name = "longitude of first grid point" ;
      Lo1:units = "degrees_east" ;
   float La2 ;
      La2:long_name = "latitude of last grid point" ;
      La2:units = "degrees_north" ;
   float Lo2 ;
      Lo2:long_name = "longitude of last grid point" ;
```

Lo2:units = "degrees_east";
float Dla;
    Dla:long_name = "lat-direction grid length";
    Dla:units = "degrees_north";
float Dlo;
    Dlo:long_name = "lon-direction grid length";
    Dlo:units = "degrees_east";
int Nj;
    Nj:long_name = "number of points along longitude";
int Ni;
    Ni:long_name = "number of points along latitude";
float Di;
    Di:long_name = "lon-direction grid length";
    Di:units = "degrees_east";
float Dj;
    Dj:long_name = "lat-direction grid length";
    Dj:units = "degrees_north";
float lat(lat);
    lat:long_name = "Latitude";
    lat:units = "degrees_north";
    lat:_CoordinateAxisType = "Lat";
float lon(lon);
    lon:long_name = "Longitude";
    lon:units = "degrees_east";
    lon:_CoordinateAxisType = "Lon";
char encoding(enc, str_len);
    encoding:long_name = "dbz byte encoding";
byte dbz(lat, lon);
    dbz:long_name = "Meteorlogix Radar Reflectivity";
    dbz:standard_name = "radar_reflectivity";
    dbz:_FillValue = 16b;
    dbz:valid_range = 0b, 64b;
    dbz:units = "encoded_DBZ";
    dbz:_CoordinateAxes = "time lat lon";

// global attributes:

://:Conventions = "CF-1.0";
:Conventions = ":_Coordinates";

data:

grid_type = "Latitude/longitude";
Nla = 5760;
Nlo = 5120;
Nj = 5760;
Ni = 5120;
La1 = 14.25;
Lo1 = -129.00;
La2 = 63.75 ;
Lo2 = -63.00 ;
Dla = 0.00859567 ;
Dlo = 0.0128931 ;
Di = 0.00859567 ;
Dj = 0.0128931 ;

// Lat and lon arrays are the latitude and longitude values at each grid point.
// They are omitted here for space considerations.

lat = (..);
lon = (..);

15.6.3.2 Satellite Data

GINI images are used as the source of satellite data, and since these images cover many channels (wavelengths) and geographic areas at differing resolutions, a number of products are used to provide full satellite coverage. For large-scale viewing areas, the display requests the lower resolution products. As the viewing area of the display shrinks, (i.e., zooming in), higher-resolution images are requested.

Data values in satellite images range from 0-255, and represent albedo for the visible channel and temperature (warm to cold) for the IR and WV channels. The table below lists the full set of satellite images used in the system. The aggregation name can be found in the catalog file mentioned above. Variable name is the name of the variable in the netCDF file for that particular dataset. Also listed below is an example of the format of one of the netCDF satellite files.

<table>
<thead>
<tr>
<th>Aggregation Name</th>
<th>GINI Sector</th>
<th>Channel Name</th>
<th>Resolution</th>
<th>Variable Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dec IR Conus</td>
<td>SUPER-NATIONAL</td>
<td>Infrared</td>
<td>8km</td>
<td>micron11</td>
</tr>
<tr>
<td>Dec IR East</td>
<td>EAST-CONUS</td>
<td>Infrared</td>
<td>4km</td>
<td>micron11</td>
</tr>
<tr>
<td>Dec IR West</td>
<td>WEST-CONUS</td>
<td>Infrared</td>
<td>4km</td>
<td>micron11</td>
</tr>
<tr>
<td>Dec VIS Conus</td>
<td>SUPER-NATIONAL</td>
<td>Visible</td>
<td>8km</td>
<td>Visible</td>
</tr>
<tr>
<td>Dec VIS East</td>
<td>EAST-CONUS</td>
<td>Visible</td>
<td>1km</td>
<td>Visible</td>
</tr>
<tr>
<td>Dec VIS West</td>
<td>WEST-CONUS</td>
<td>Visible</td>
<td>1km</td>
<td>Visible</td>
</tr>
<tr>
<td>Dec WV Conus</td>
<td>SUPER-NATIONAL</td>
<td>Water Vapor</td>
<td>8km</td>
<td>micron6_7</td>
</tr>
<tr>
<td>Dec WV East</td>
<td>EAST-CONUS</td>
<td>Water Vapor</td>
<td>4km</td>
<td>micron6_7</td>
</tr>
<tr>
<td>Dec WV West</td>
<td>WEST-CONUS</td>
<td>Water Vapor</td>
<td>8km</td>
<td>micron6_7</td>
</tr>
</tbody>
</table>

netcdf CONUS_IR.cdl {

dimensions:
  //     time = UNLIMITED ;
  lat = 800 ;
  lon = 1300 ;
  str_len = 100 ;
}
variables:

double reftime ;
    reftime:long_name = "reference time" ;
    reftime:units = "seconds since 1970-1-1 00:00" ;

double valtime ;
    valtime:long_name = "valid time" ;
    valtime:units = "seconds since 1970-1-1 00:00" ;

int grid_type_code ;
    grid_type_code:long_name = "GRIB-1 GDS data representation type" ;

int file_type ;
    file_type:long_name = "scan, 0=west, 1=east" ;

char grid_type(str_len) ;
    grid_type:long_name = "GRIB-1 grid type" ;

char grid_name(str_len) ;
    grid_name:long_name = "grid name" ;

char x_dim(str_len) ;
    x_dim:long_name = "x dimension name" ;

char y_dim(str_len) ;
    y_dim:long_name = "y dimension name" ;

int Ni ;
    Ni:long_name = "number of points along i-axis" ;

int Nj ;
    Nj:long_name = "number of points along j-axis" ;

float La1 ;
    La1:long_name = "latitude of first grid point" ;
    La1:units = "degrees_north" ;

float Lo1 ;
    Lo1:long_name = "longitude of first grid point" ;
    Lo1:units = "degrees_east" ;

float Di ;
    Di:long_name = "i-direction grid length" ;
    Di:units = "m" ;

float Dj ;
    Dj:long_name = "j-direction grid length" ;
    Dj:units = "m" ;

byte micron11(lat, lon) ;
    micron11:long_name = "radiance counts" ;
    micron11:units = "counts" ;
    micron11:_CoordinateAxes = "time lat lon" ;
    micron11:_FillValue = '¥x7f' ;
    micron11:missing_value = '¥x7f' ;

float lon(lon) ;
    lon:long_name = "Longitude" ;
    lon:units = "degrees_east" ;
    lon:_CoordinateAxisType = "Lon" ;

float lat(lat) ;
    lat:long_name = "Latitude" ;
    lat:units = "degrees_north" ;
    lat:_CoordinateAxisType = "Lat" ;
// global attributes:

  //:Conventions = "CF-1.0" ;
  Conventions = "_Coordinates" ;

data:

  grid_type_code = 0 ;
  grid_type = "Latitude/Longitude" ;
  grid_name = "Satellite" ;
  file_type = 1 ;
  x_dim = "lon" ;
  y_dim = "lat" ;
  Ni = 1300 ;
  Nj = 800 ;
  Di = 0.0715 ;
  Dj = 0.0715 ;
  La1 = 15.0 ;
  Lo1 = -145. ;

  // Lat and lon arrays are the latitude and longitude values at each grid point.
  // They are omitted here for space considerations.

  lat = (..);
  lon = (..);
}

15.7 Processing

Satellite and radar data are received, decoded into netCDF, and re-projected onto a lat-lon grid in the background by external processes. The THREDDS server re-scans the directories containing these files based on the recheck rate specified in each dataset’s aggregation XML file. The THREDDS server is then able to respond to requests from clients for particular datasets, as it maintains the available data list for each product set in the server.

Clients such as the MDSS display use the netCDF-Java API to request product data from the THREDDS server. After querying the server for the availability of products, a request is made for the product data covering a particular bounding box area at a particular sampling rate. The server then returns the requested data to the client and waits for the next request.
16 WMS Server

16.1 Identification

Web Map Service; a specification from the Open Geospatial Consortium (OGC), for dynamically serving metadata, geographic images, and feature information.

16.2 Type

Process

16.3 Purpose

The purpose of the WMS server is to serve digital basemap images to the display clients.

16.4 Function

The WMS server receives requests for images in the form of HTTP ‘GET’ requests. It runs on a host, which contains geographic datasets. The data are rendered by the service into a digital image. The rendering may style features differently, prioritize the presence and/or layering of geographic features, and determine label placement and style. The service returns a digital image in the requested format.

16.5 Dependencies

Relevant geographic datasets must be available for the WMS service to render. At a minimum, these should include some type of geographic context for the MDSS road segments: coastlines, national boundaries, state boundaries, county boundaries, municipal boundaries, interstates, US highways, state highways, county roads, city streets, topography, rivers, lakes, and/or places of interest. The preparation of these datasets may be specific to the chosen WMS implementation and is beyond the scope of this document.

16.6 Interfaces

The point of using a Web Map Service is its standard interface definition. The details of the WMS protocol are documented on the Open Geospatial Consortium web site (http://www.opengeospatial.org/standards/wms). In the case of the MDSS display client, the ESRI ArcIMS implementation of the WMS specification was tested. The request has not been tested on other implementations of the WMS specification. To obtain a digital image from the server, the display makes a request of the form:
http://host/path/service?REQUEST=map&VERSION=1.1.1
&SRS=EPSG:none&TRANSPARENT=true&ServiceName=service
&BBOX=-109.0,39.6,-102.0,41.07&LAYERS=1,2,3&STYLES=,,
&FORMAT=image/png&WIDTH=700&HEIGHT=415

where:
host is the domain on which the service is installed.
path is the path to the service on the web server.
service is the name of the WMS service.
REQUEST is the type of request, initially “capabilities,” then “map”.
VERSION is the version of the WMS specification (1.1.1).
SRS is the spatial reference system (projection) of the resulting image.
“none” indicates the native projection of the data should be used.
TRANSPARENT indicates that the image should have a transparent background.
ServiceName is the name of the WMS service from which to request an image.
BBOX is the lat/lon bounding box of the data request in WSNE order.
LAYERS is the list of layer numbers requested. The display requests all layers.
STYLES is the custom style information for each layer. Use default.
FORMAT is the digital image format of the response. The display uses png.
WIDTH is the desired width of the image in pixels.
HEIGHT is the desired height of the image in pixels.

17 Utility Processes

17.1 Merge Variables Process

17.1.1 Identification
merge_var

17.1.2 Type

Process

17.1.3 Purpose

The merge variables process is a relatively simple routine that combines the output of the
two main RCTM software modules, roadcond_mdss and frost. The goal is to have a single
output file ready for conversion to a format readable by the display. The necessary
variables are read from each file and written to the output file.

17.1.4 Function

The merge variables process reads a parameter file which describes which variables are to
be extracted from the two input files. These variables’ data are simply read into arrays and
written out into the output file. The variable name may be changed. The output name is specified in the parameter file. For example, the “RoadT” variable describing the forecast bridge surface temperatures may be written out as “BridgeT” to distinguish it from the roadway forecast RoadT variable.

17.1.5 Dependencies

The merge variables process requires two input files and a parameter file describing which variables to read from each. The named variables must exist in their respective files.

17.1.6 Interfaces

17.1.6.1 Command Interface

% merge_var forecast_gen_time site_list_file cdl_file input_file_1 input_file_2 output_file
[-d debug_level] [-l log_file]

where:
  forecast_gen_time is the UNIX time of the first road condition forecast to be generated. This will be truncated to the top of the hour.
  site_list_file is the name of the text file containing the list of sites to be processed.
  cdl_file is the name of the CDL file describing the output format
  input_file_1 is the name of the first input file
  input_file_2 is the name of the second input file
  output_file is the name of the netCDF output file to be produced.
  debug_level is the level of debugging information to be output to the log file. The default level is 0.
  log_file is the file to which log output should be written.

17.1.6.2 Start Script

_Merge_var_ is started by a Python script which gathers the names of the appropriate input files, static (configuration) files, and then runs the program. Forecast generation time and static data files are set within Python "wrapper" scripts called ep_merge_no_tmt.py, ep_merge_rec_tmt.py, and ep_merge_cur_tmt.py for merging no treatment, recommended treatment, and user-defined treatment runs, respectively. Each of these scripts sets arguments and then runs a script called run_proc.py, which in turn actually executes the _merge_var_ program on the real-time data.

_Merge_var_ can also be invoked by the user-defined Python script in the TUNL. This script constructs the command line, runs the program, and passes the output file back to the display.

17.1.6.3 Configuration Files

17.1.6.3.1 Site List Configuration File
This file contains information on the output sites. The format of this file is described in Section 9.2 of this document.

17.1.6.4 Inputs

17.1.6.4.1 First Input Data File

This file contains the output from a roadcond_mdss run and is described in Section 12.6.5 of this document.

17.1.6.4.2 Second Input Data File

This file contains the output from frost and is described in Section 13.6.5 of this document.

17.1.6.5 Output

The merge variables output is in netCDF format. Note that the output data starts at 00Z of a particular day. The 48 hours of merged output data will be surrounded by missing data. The desired output file’s name is passed to the merge_var as a command line argument. A sample CDL file can be found on the MDSS CD at /data/static_data/cdl/merge_tmt.cdl. Sample merge

The CDL file describing this netCDF file format is:

```cdl
netcdf merge_tmt {
  dimensions:
    days = 4; // number of days
    fc_times_per_day = 24; // fcst times per day
    max_site_num = 25; // number of locations
    max_str_len = 80; // max string length
    num_times = 48; // number of times road condtions are computed

  variables:
    double creation_time;
      creation_time:long_name = "time at which forecast file was created";
      creation_time:units = "seconds since 1970-1-1 00:00:00";

    double forc_time;
      forc_time:long_name = "time of earliest forecast";
      forc_time:units = "seconds since 1970-1-1 00:00:00";

    int num_sites;
      num_sites:long_name = "number of actual_sites";

    int site_list(max_site_num);
      site_list:long_name = "forecast site id numbers";
}
```
int type;
    type:long_name = "cdl file type";

    // Forecast variables
    float application_rate(max_site_num, days, fc_times_per_day);
    application_rate:long_name = "chemical application rate";
    application_rate:units = "lb/mile or gal/mile";
    application_rate:reference = "units depend on chem_form";

    float apply_chem(max_site_num, days, fc_times_per_day);
    apply_chem:long_name = "apply chemicals";
    apply_chem:values = "0 or 1";

    float available_H2O(max_site_num, days, fc_times_per_day);
    available_H2O:long_name = "water available for chemical dilution";
    available_H2O:units = "lb/ft2";

    float available_chem(max_site_num, days, fc_times_per_day);
    available_chem:long_name = "pure de-icing chemicals on road";
    available_chem:units = "lb/ft2";

    float bridge_T(max_site_num, days, fc_times_per_day);
    bridge_T:long_name = "bridge surface temperature, C";
    bridge_T:units = "degrees Celsius";

    float bridge_TempF(max_site_num, days, fc_times_per_day);
    bridge_TempF:long_name = "road surface temperature, F";
    bridge_TempF:units = "degrees Fahrenheit";

    float bridge_frost_potential(max_site_num, days, fc_times_per_day);
    bridge_frost_potential:long_name = "potential for frost on bridge";
    bridge_frost_potential:value0 = "None";
    bridge_frost_potential:value1 = "Low";
    bridge_frost_potential:value2 = "Medium";
    bridge_frost_potential:value3 = "High";

    float chem_form(max_site_num, days, fc_times_per_day);
    chem_form:long_name = "chemical form";
    chem_form:value0 = "Dry";
    chem_form:value1 = "Prewet";
    chem_form:value2 = "Liquid";

    float chem_type(max_site_num, days, fc_times_per_day);
    chem_type:long_name = "chemical type";
    chem_type:value0 = "Not set";
    chem_type:value1 = "NaCl";
    chem_type:value2 = "CaCl2";
chem_type:value3 = "MgCl2";  // Not Supported
chem_type:value4 = "CaMg Acetate";  // Not Supported
chem_type:value5 = "K Acetate";  // Not Supported
chem_type:value6 = "Caliber";
chem_type:value7 = "IceSlicer";  // Not Supported
chem_type:value8 = "IceBan";

float chemical_concentration(max_site_num, days, fc_times_per_day);
chemical_concentration:long_name = "chemical concentration";
chemical_concentration:units = "percent";

float do_plowing(max_site_num, days, fc_times_per_day);
do_plowing:long_name = "do plowing";
do_plowing:values = "0 or 1";

float mobility(max_site_num, days, fc_times_per_day);
mobility:long_name = "net mobility";
mobility:values = "0.0 to 1.0";

float nominal_chem(max_site_num, days, fc_times_per_day);
nominal_chem:long_name = "theoretical chem concentration";
nominal_chem:units = "percent";

float phase_type(max_site_num, days, fc_times_per_day);
phase_type:long_name = "road water phase";
phase_type:value0 = "Dry";
phase_type:value1 = "Wet";
phase_type:value2 = "Chemically wet";
phase_type:value3 = "Chemically ice";
phase_type:value4 = "Slush";
phase_type:value5 = "Snow";
phase_type:value6 = "Ice";

float precip_type(max_site_num, days, fc_times_per_day);
precip_type:long_name = "precip type on road";
precip_type:value0 = "None";
precip_type:value1 = "Rain";
precip_type:value2 = "Snow";
precip_type:value3 = "Mixed rain/snow";
precip_type:value4 = "Mixed snow/rain";
precip_type:value5 = "Freezing rain";

float road_T(max_site_num, days, fc_times_per_day);
road_T:long_name = "road surface temperature, C";
road_T:units = "degrees Celsius";

float road_TempF(max_site_num, days, fc_times_per_day);
road_TempF:long_name = "road surface temperature, F";
road_TempF:units = "degrees Fahrenheit";
The merge_var processing is fairly straightforward. After generating the lists of the input variable names to be read from each of the two input files and the names for each of those variables in the output file, each variable is read in turn and written to the output file.

17.2 Forecast Data Re-formatter

The fcst2bin source code can be found on the MDSS CD in /src/apps/fcst2bin.

17.2.1 Identification

fcst2bin

17.2.2 Type

Process
17.2.3 Purpose

The purpose of the Forecast Data Re-formatter is to extract and translate forecast data into the binary format used by the display.

17.2.4 Function

The input weather forecast and RCTM output netCDF files have similar formats. The Forecast Data Re-formatter reads a file of this format, extracts the variables listed in a configuration file, and writes these variables to a file in the format desired by the display. The output contains a time series of forecast data for a list of sites.

17.2.5 Dependencies

The sites listed in the site list configuration file used by this process must be a subset of the input file site list. See Section 9.2 of this document for site list file specifications. The variables listed in the variable list configuration file must all exist in the input file.

17.2.6 Interfaces

17.2.6.1 Command Interface

```
% fcst2bin forecast_time input_file site_list var_config_file output_file [-l log_file]
```

where:
- `forecast_time` is the UNIX time of the first forecast of the 48-hour forecast period. This will be truncated to the top of the hour.
- `input_file` is the name of the file containing data to be extracted.
- `site_list` is the name of the text file containing the list of sites to be processed.
- `var_config_file` is the name of the configuration file containing the list of variables which will be extracted.
- `output_file` is the name of the display-ready formatted file to be produced.
- `log_file` is the file to which log output should be written.

17.2.6.2 Start Script

The `fcst2bin` program is started by a Python script, which gathers arguments and current real-time data filenames. The following list of scripts are used to do the conversions:

```
ep_conv_wx.py  converts weather forecast data
ep_conv_no_tmt.py  converts the no treatment option data
ep_conv_rec_tmt.py  converts the recommended treatment option data
ep_conv_cur_tmt.py  converts the current (selected) treatment option data
```

Each of these "wrapper" scripts sets variables (static files, etc.) and then runs another script called `run_proc.py` which in turn executes `fcst2bin` using the most recent real-time data. These scripts are included with the MDSS CD in the /scripts/python subdirectory.
fcst2bin can also be invoked by the user_treatment.py script (located in the same /scripts/python directory above) via the TUNL. This script constructs the command line, runs the program, and passes the output file back to the display.

17.2.6.3 Configuration Files

17.2.6.3.1 Site List Configuration File

This file contains information on the RCTM output sites. The format of this file is described in Section 9.2 of this document.

17.2.6.3.2 Variable List Configuration File

The variable list configuration file is a simple ASCII text file which lists each variable that the user wishes to be converted into the output file. Each line in the file consists of a single variable name exactly as it appears in the input file's CDL. These files are located on the MDSS CD under /data/static_data/config.

The current contents of the weather conversion file (disp_wx_var_names.asc) are:

TempF
dewptF
rh_pct
wind_speed_mph
wind_dir
precip_rate_inches
precip_type
precip_accum_inches
snow_rate_inches
snow_accum_inches
snow_accum_total_inches
snow_accum_48hr_total_inches
prob_precip03_pct
cprob_rain
cprob_snow
cprob_ice
blowing_snow_potential

The current contents of the road condition conversion file (disp_re_var_names.asc) are:

road_TempF
snow_depth_inches
mobility
chemical_concentration
do_plowing
apply_chem
treatment_time
The weather forecast and road condition forecast data file are in netCDF format. This process will work with any forecast file having the same layout. That is, the same number of days and forecast times per day as in the CDL files described in Sections 12.6.4.1 and 12.6.5.

17.2.8 Output

The reformatted data file created is in the format expected by the display. See Section 19.6.2.5 for a description of that format.

17.3 Observation Data Re-formatter

The meso2bin source code can be found on the MDSS CD in /src/apps/meso2bin.

17.3.1 Identification

meso2bin

17.3.2 Type

Process

17.3.3 Purpose

The purpose of the Observation Data Re-formatter is to extract and translate observation data into the binary format used by the display.

17.3.4 Function

Files containing Environmental Sensor Station (ESS) observations from the MADIS data feed are in netCDF format. The Observation Data Re-formatter reads files of this format, extracts the variables listed in a configuration file, and writes these variables to a file in the
format desired by the display. The output contains the most recent ESS observations from the input files for a list of sites.

17.3.5 Dependencies

The sites listed in the site list configuration file used by this process must be a subset of the input file site list. See Section 9.2 of this document for site list file specifications. The variables listed in the variable list configuration file must all exist in the input file.

17.3.6 Interfaces

17.3.6.1 Command Interface

```
% meso2bin site_list_file site_var_file num_files input_file(s)... output_file
   [-l logfile_base] [-d debug_level]
```

where:
- `site_list_file` is the name of the text file containing the full site information for the sites to be extracted.
- `site_var_file` is the name of the file containing the list of sites and variables to be extracted.
- `num_files` is the number of input files to be found on the command line.
- `input_file(s)` is a list of files containing data to be extracted.
- `output_file` is the name of the display-ready formatted file to be produced.
- `logfile_base` is the file to which log output should be written.
- `debug_level` controls the level of logging.

17.3.6.2 Start Script

The `meso2bin` program is started by a Python script, which gathers arguments and current real-time data filenames. The following script is used to do the conversions:

```
ep_conv_meso.py   converts ESS observation data
```

This "wrapper" script sets variables (static files, etc) and then runs another script called `conv_meso.py` which in turn executes meso2bin using the most recent real-time data. These scripts are included with the MDSS CD in the /scripts/python subdirectory.

17.3.6.3 Configuration Files

17.3.6.3.1 Site List Configuration File

This file contains information on the output sites. The format of this file is described in Section 9.2 of this document. See /data/static_data/site_list/rwis_sites.asc on the MDSS CD for the most recent version of this file.
17.3.6.3.2 Site-Variable Configuration File

The configuration file is a simple ASCII text file which lists each site ID and the set of variables desired for that site, separated by semi-colons. Each site must exist in the Site List Configuration File, and the variable must be found in the set of input file(s). See the file site_var_file.asc under /data/static_data/site_list on the MDSS CD.

17.3.7 Input

The RWIS observation files are in netCDF format. This process will work with any observation file having the same layout as the MADIS netCDF files.

17.3.8 Output

The reformatted data file created is in the format expected by the display. See Section 19.6.2.5 for a description of that format.

18 Treatment Update Network Layer (TUNL)

The Treatment Update Network Layer is a collection of scripts that handle requests initiated by the MDSS display application. This, more often than not, involves consulting the file system and returning a particular facet of the state of the system. In the cases where the display determines that forecast data are required, the TUNL scripts gather the data, either by directly consulting the file system or by invoking processes to generate the requested data.

The TUNL scripts are divided into two directories on the MDSS CD: /scripts/cgi/cgi-bin, and /scripts/cgi/cgi-bin-dicast. The scripts are segregated for security reasons. Scripts that simply request data or meta-data are located in the cgi-bin/ directory, whereas scripts that will run applications or write data to the local disk are located in cgi-bin-dicast/. (Dicast refers to the user who owns the scripts in this context.)

18.1 Identification

N/A

18.2 Type

Subsystem, i.e., collection of interconnected scripts.
18.3 Purpose

The MDSS display application needs to interact with the MDSS data store in order to obtain the data it is tasked with presenting. To achieve this it consults the MDSS data store machine through its web server. TUNL software provides the functionality to query the state of the system. The majority of the queries are done through polling. That is, the display regularly queries the MDSS system to determine, for example, the current time and whether or not new data are available. If new data are available, the display can download the new data through the TUNL. The TUNL also provides the functionality to generate road conditions based on user-defined treatment plans. The Python CGI scripts that may be invoked by the web server are listed below.

18.3.1 Get Current Time Script

The time.cgi script gets the system time from the web server machine. The web server’s system clock is assumed to be synchronized with the rest of the MDSS system. Use of this script ensures that the display is not affected by a client whose system clock is not properly set. The display polls to ensure that the displayed system time is current. This script can be found under /scripts/cgi/cgi-bin.

18.3.2 Get Latest Data Script

The latest_data.cgi script gets the latest data from the system data store. It can be called in either of two modes. It can return the valid time of the latest data or return the data itself. The former is typically used in a polling fashion to determine if new data has been generated. If it determines that new data exist, the second mode is used to download that data. The script can also be given a UNIX time argument. When this is given the script returns only the latest in a certain time window; the time window being from the given time to the given time plus the system forecast interval. This script can be found under /scripts/cgi/cgi-bin.

18.3.3 Get Closest Data Script

The closest_data.cgi script is similar to latest_data.cgi. However, rather than finding the latest data, it finds the data closest in time to the time argument passed by the display. The script can also be given a ‘beforeFlag’. The script then would return the closest in time before the given time argument. This script can be found under /scripts/cgi/cgi-bin.

18.3.4 List Data Script

The list_data.cgi script returns a list of all available data times for a particular type of data. This script can be found under /scripts/cgi/cgi-bin.
18.3.5 Get Permissions Script

The `get_permission.cgi` script checks the permissions of the requesting host. The script searches for the host name in several system files. These files contain lists of hosts that are allowed to perform certain operations. These operations include whether the display user is allowed to 1) see the current plan for the given segment, or, 2) select treatments for the segment. This script can be found under `/scripts/cgi/cgi-bin`.

18.3.6 Select Treatment Plan Script

The `select_plan.cgi` script saves the selected treatment plan. This selected plan contains the course of action that the garage intends to follow on the specified route. Only those users who have proper permissions can save the selected plan for a route. This is usually someone working at the DOT garage in charge of that route. This script can be found under `/scripts/cgi/cgi-bin-dicast`.

18.3.7 User Defined Treatment Plan Script

The `user_defined_plan.cgi` script runs a treatment plan defined by the user within the display. This involves gathering and creating a number of configuration files used by the RCTM. The process `roadcond_mdss` is invoked to run the specified plan and determine the resultant road conditions. This script can be found under `/scripts/cgi/cgi-bin-dicast`.

18.3.8 Reset Sites Script

The `resetSites.cgi` script allows the user to reset the road conditions within the RCTM data store for specified sites. This action may be required because the reality of the road situation does not match the system’s model of reality. Currently, the only allowed actions are a reset of the snow depth and chemical mass on the road to zero. That is, the user should invoke this script when it is known that there is no snow and/or no chemicals on the road and the display indicates that the system thinks otherwise. The effect of the reset action will be seen at the next forecast update which will be generated starting with a clear road. Permissions are checked to determine whether the user is allowed to reset the indicated site(s). This script can be found under `/scripts/cgi/cgi-bin-dicast`.

18.3.9 Get Reset Sites Script

The `getResetSites.cgi` script queries the system to determine which sites are scheduled to be reset in the next forecast generation cycle. This allows the display to indicate to the users the status of that list. This script can be found under `/scripts/cgi/cgi-bin`.
18.4 Function

TUNL is a collection of CGI scripts written in Python. Each CGI script is directly invoked by the web server. The CGI scripts parse arguments and then either directly query the file system or invoke other Python scripts. Each successive invocation returns data to its parent. Eventually, the requested data are returned to the CGI script that passes it back to the display. The first Python script called also sets a timer and monitors its children to ensure that the processing has not hung. Should the processing either fail or time out, a failure message is returned to the display.

18.5 Dependencies

TUNL requires read/write access to the MDSS data directory tree which is located based on the value of the RCTM_ROOT_DIR environment variable. Typically write access can be restricted to small sub-trees. These write-accessible sub-trees include directories containing users’ treatment plans, temporary work space used in the generation of user-defined treatment plans, and request log files.

TUNL also requires properly formatted requests from the MDSS display. Improperly generated or communicated requests are gracefully handled.

18.6 Interfaces

18.6.1 Command Interfaces

As with all CGI scripts, the inputs are in ASCII text. All the scripts return data to the display. The returned data can either be text or binary data depending on the query. The data type is described in the first part of the returned message. The content type headers returned to the display are either “Content-Type: text/plain” or “Content-Type:text/octet-stream” followed by a blank line. The third line of the returned text is a return string indicating the status of the request. Possible values are “SUCCESS” or “FAILURE”. The fourth line of the returned text is either the full path of the file found or the number of files found. Starting on the fifth line is either the UNIX time of the file(s), one per line, or the actual data.

The following scripts are all CGI scripts written in the Python language. They can be found on the MDSS CD in the /scripts/cgi directory.

18.6.1.1 Get Current Time Script

Name: time.cgi

Inputs: None

Outputs:
18.6.1.2 Get Latest Data and Get Closest Data Scripts

Name: latest_data.cgi, closest_data.cgi, list_data.cgi

Inputs:

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Definition</th>
<th>Recognized Values</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Location</td>
<td>Data Location</td>
<td>“nt”</td>
<td>No treatment roadcond_mdss run request</td>
</tr>
<tr>
<td></td>
<td></td>
<td>“ct”</td>
<td>current treatment plan roadcond_mdss run request</td>
</tr>
<tr>
<td></td>
<td></td>
<td>“st”</td>
<td>suggested treatment plan roadcond_mdss run request</td>
</tr>
<tr>
<td></td>
<td></td>
<td>“wx”</td>
<td>weather forecast request</td>
</tr>
<tr>
<td></td>
<td></td>
<td>“rwis”</td>
<td>weather observation request</td>
</tr>
<tr>
<td></td>
<td></td>
<td>“fh”</td>
<td>Forecast history request</td>
</tr>
<tr>
<td></td>
<td></td>
<td>“oh”</td>
<td>Observation history request</td>
</tr>
<tr>
<td></td>
<td></td>
<td>“avl”</td>
<td>GPS/AVL data request</td>
</tr>
<tr>
<td></td>
<td></td>
<td>“nwsww”</td>
<td>NWS watches &amp; warning (not used)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>“alert”</td>
<td>Alert data request</td>
</tr>
<tr>
<td>Data</td>
<td>Data Type Requested</td>
<td>“filetime”</td>
<td>Data time request</td>
</tr>
<tr>
<td></td>
<td></td>
<td>“file”</td>
<td>Data file request</td>
</tr>
<tr>
<td>Host</td>
<td>Display Hostname</td>
<td></td>
<td>Used only if location = “ct”</td>
</tr>
<tr>
<td>beforeFlag</td>
<td>Before Given Time</td>
<td>“on”</td>
<td>Used only with closest_data.cgi</td>
</tr>
<tr>
<td>time</td>
<td>Reference UNIX Time</td>
<td>10 digits</td>
<td>Used with closest_data.cgi, optional with latest_data.cgi, not used in list_data.cgi</td>
</tr>
</tbody>
</table>

Output:

<table>
<thead>
<tr>
<th>Data</th>
<th>Type</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
<td>Text</td>
<td>UNIX time of data found (if data = “filetime”)</td>
</tr>
</tbody>
</table>

18.6.1.3 Get Permissions Script

Name: get_permission.cgi
<table>
<thead>
<tr>
<th><strong>Variable Name</strong></th>
<th><strong>Definition</strong></th>
<th><strong>Recognized Values</strong></th>
<th><strong>Comments</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>Site ID</td>
<td>8 digits</td>
<td></td>
</tr>
<tr>
<td>host</td>
<td>Display Hostname</td>
<td></td>
<td>Needed to verify permissions</td>
</tr>
<tr>
<td>numTx</td>
<td>Number of Treatments in Plan</td>
<td>0–99</td>
<td></td>
</tr>
<tr>
<td>time</td>
<td>Treatment time</td>
<td>10 digits</td>
<td>Comma separated list of numTx UNIX times</td>
</tr>
<tr>
<td>rate</td>
<td>Chemical Application Rate</td>
<td>0-9999</td>
<td>Repeated numTx times. Chemical application rates in lb/lane-mi or gal/lane-mi (depends on chem_form)</td>
</tr>
<tr>
<td>chem</td>
<td>Chemical type</td>
<td></td>
<td>Repeated numTx times. Chemical application types</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>Plow Only</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>NaCl</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>CaCl₂</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>MgCl₂</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>CaMg Acetate</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>K Acetate</td>
<td></td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>Caliber</td>
<td></td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>IceSlicer</td>
<td></td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>IceBan</td>
<td></td>
</tr>
<tr>
<td>chem_form</td>
<td>Chemical form</td>
<td></td>
<td>Repeated numTx times. Chemical application forms</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>Dry</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Prewet</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>Liquid</td>
<td></td>
</tr>
</tbody>
</table>

**Inputs:**

**Output:**
If the action is allowed, “SUCCESS” is returned. Otherwise, “FAILURE” is returned.

**18.6.1.4 Select Treatment Plan**

**Name:** select_plan.cgi
Inputs:

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Definition</th>
<th>Recognized Values</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>Site ID</td>
<td>8 digits</td>
<td></td>
</tr>
<tr>
<td>host</td>
<td>Display Hostname</td>
<td></td>
<td>Needed to verify permissions</td>
</tr>
<tr>
<td>numTx</td>
<td>Number of Treatments in Plan</td>
<td>0–99</td>
<td></td>
</tr>
<tr>
<td>time</td>
<td>Treatment time</td>
<td>10 digits</td>
<td>Comma separated list of numTx UNIX times</td>
</tr>
<tr>
<td>rate</td>
<td>Chemical Application Rate</td>
<td>0-9999</td>
<td>Repeated numTx times. Chemical application rates in lb/lane-mi or gal/lane-mi (depends on chem_form)</td>
</tr>
<tr>
<td>chem</td>
<td>Chemical type</td>
<td>Repeated numTx times. Chemical application types</td>
<td></td>
</tr>
<tr>
<td>chem_form</td>
<td>Chemical form</td>
<td>Repeated numTx times. Chemical application forms</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Definition</th>
<th>Recognized Values</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>Site ID</td>
<td>8 digits</td>
<td></td>
</tr>
<tr>
<td>host</td>
<td>Display Hostname</td>
<td></td>
<td>Needed to verify permissions</td>
</tr>
<tr>
<td>numTx</td>
<td>Number of Treatments in Plan</td>
<td>0–99</td>
<td></td>
</tr>
<tr>
<td>time</td>
<td>Treatment time</td>
<td>10 digits</td>
<td>Comma separated list of numTx UNIX times</td>
</tr>
<tr>
<td>rate</td>
<td>Chemical Application Rate</td>
<td>0-9999</td>
<td>Repeated numTx times. Chemical application rates in lb/lane-mi or gal/lane-mi (depends on chem_form)</td>
</tr>
<tr>
<td>chem</td>
<td>Chemical type</td>
<td>Repeated numTx times. Chemical application types</td>
<td></td>
</tr>
<tr>
<td>chem_form</td>
<td>Chemical form</td>
<td>Repeated numTx times. Chemical application forms</td>
<td></td>
</tr>
</tbody>
</table>

Note that for MDSS Release-5, only 5 of the chemical types are supported. CaMg Acetate, K Acetate, and IceBan are not supported.

Output:

The script returns “SUCCESS” or “FAILURE” indicating whether or not the treatment selection was accomplished.

18.6.1.5 User Defined Treatment Plan

Name: user_defined_plan.cgi
Inputs:

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Definition</th>
<th>Recognized Values</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>Site ID</td>
<td>8 digits</td>
<td></td>
</tr>
<tr>
<td>host</td>
<td>Display Hostname</td>
<td></td>
<td>Needed to verify permissions</td>
</tr>
<tr>
<td>numTx</td>
<td>Number of Treatments in Plan</td>
<td>0–99</td>
<td></td>
</tr>
<tr>
<td>time</td>
<td>Treatment time</td>
<td></td>
<td>Repeated numTx times.</td>
</tr>
<tr>
<td>rate</td>
<td>Chemical Application Rate</td>
<td>Repeated numTx times. Chemical application rates in lb/lane-mi or gallons/lane-mi (depends on chem_form)</td>
<td></td>
</tr>
<tr>
<td>chem</td>
<td>Chemical type</td>
<td>Repeated numTx times. Chemical application types</td>
<td></td>
</tr>
<tr>
<td>chem_form</td>
<td>Chemical form</td>
<td>Repeated numTx times. Chemical application forms</td>
<td></td>
</tr>
</tbody>
</table>

Note that for MDSS Release-5, only 5 chemical types are currently supported. CaMg Acetate, K Acetate, and IceBan are not supported.

Output:

If successful, returns the binary data containing the forecast road conditions that were generated based on the specified treatment plan.
18.6.1.6 **Reset Script**

**Name:** resetSites.cgi

**Inputs:**

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Definition</th>
<th>Recognized Values</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host</td>
<td>Name of machine making request</td>
<td></td>
<td>Hostname needs to match name in system permissions file</td>
</tr>
<tr>
<td>Site</td>
<td>Site to have road conditions reset</td>
<td>8 digits</td>
<td>Multiple sites can be reset within the same query by providing repeated site variables</td>
</tr>
</tbody>
</table>

**Output:**

The script returns “SUCCESS” or “FAILURE” indicating whether or not the reset was accomplished.

18.6.1.7 **Get Reset Sites Script**

**Name:** getResetSites.cgi

**Inputs:**

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Definition</th>
<th>Recognized Values</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host</td>
<td>Name of machine making request</td>
<td></td>
<td>Hostname needs to match name in system permissions file</td>
</tr>
</tbody>
</table>

**Output:**

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>sites</td>
<td>List of sites scheduled to be reset</td>
</tr>
</tbody>
</table>

19 **DISPLAY**

The MDSS prototype display application is a Java application that retrieves road weather forecast data and treatment recommendations from the RCTM, and displays them for the user. A tutorial of the display (PowerPoint file) can be found on the MDSS web site:

http://www.rap.ucar.edu/projects/rdwx_mdss/documents/MDSS_Demo_Tutorial_Colorado_Nov04.pps#38

The tutorial provides an easy reference for the major features and functions of the display application.
The architectural goal for the display was to keep it as thin as possible, pushing decision-making algorithms into the server modules as much as possible. Some complexity exists due to the requirement that the application hide information from some users who have a restricted viewing permission level, but this is handled for the most part by the TUNL layer, described above. The other area of complexity lies in the calculation of alerts. The details of this are described in this section.

The Java display source code can be found on the MDSS CD under /src/apps/display/java/src. Several 3rd-party libraries are used by the display and these are located under /src/apps/display/java/ext_lib. All of these libraries are freely available, but may be restricted by license agreements. These dependencies are described in the “Dependencies” section below.

19.1 Identification

xdss.jnlp

19.2 Type

Java Network Launching Protocol file for starting the MDSS display with Java Webstart.

19.3 Purpose

The purpose of the display system is to demonstrate the functionality of the various MDSS components by providing a graphical, interactive view of the output data. In this way the display system can be a starting point for discussions of functional requirements for operational systems based on or incorporating components of MDSS. The MDSS prototype display features and functions were developed with feedback from representatives from IA, MN, WA, UT, NH, and CO DOTs. The domain and routes used in the 2007 system were located in Colorado.

19.4 Function

The display system reads output data from the Forecast and Observation Data Reformatters and provides the user with various derived views of the data. It also provides the user with tactical data such as road weather information system (RWIS) observations, satellite and radar observations, and automatic vehicle location (AVL) reports. Lastly, the display provides an interface to enter user-specified treatments. The treatments are submitted to the RCTM to predict the resulting effect on road condition.

The prototype MDSS display system contains a map showing any datasets the user has chosen to enable. The map allows arbitrary zooming and panning using a double-click to zoom in, a right-click to zoom out, and a click-drag to pan. The display also provides
several preconfigured maintenance area views. These are indicated in a combo box located in the upper-left corner of the display. When a maintenance area is selected in the combo box, the map is zoomed to that area and the alerts are updated to reflect only the roads and forecast points within that maintenance area. In the initial state maintenance area view, the state alert zones on the map are color coded for the worst weather forecasted for the next 48 hours.

The display system provides high-level alerts of forecasts for inclement weather, impaired road conditions, blowing snow, and bridge frost. Weather alerts are provided by grouping the weather conditions into distinct weather alert categories, as shown in Table 19.1

The weather alerts are provided in five ways:

1) A state map shows the worst weather alert category that will occur over the forecast period for each forecast zone. In the current prototype, forecast zones correspond to Colorado counties.

2) A weather alert time bar shows the worst weather alert category for the currently selected maintenance area during three forecast time ranges: 0-12 hours, 12-24 hours, and 24-48 hours.

3) A button below the alert time bar turns red whenever freezing precipitation is detected at any of the observation sites within the currently selected maintenance area.

4) Colored dots show the weather alert category of each weather forecast point at the selected time when the “Weather Alerts” map product is enabled.

5) Tooltip inspection of weather forecast points on the map yields a color bar indicating the weather alert category for every forecast hour, when any “Weather Forecasts” variable is enabled.

<table>
<thead>
<tr>
<th>Weather Alert Category</th>
<th>Precip Type</th>
<th>Precip Rate (Liquid Equiv.) (inches/hr)</th>
<th>Temperature (F)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extreme</td>
<td>Any ice</td>
<td>&gt;= 0.15 “/hr</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Snow</td>
<td>&gt;= 0.15 “/hr</td>
<td></td>
</tr>
<tr>
<td>Poor</td>
<td>Snow</td>
<td>0.05-0.15 “/hr</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Rain</td>
<td>&gt;= 0.25 “/hr</td>
<td>&lt; 35 F</td>
</tr>
<tr>
<td>Marginal</td>
<td>Snow</td>
<td>Any</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Rain</td>
<td>&gt;= 0.1 “/hr</td>
<td></td>
</tr>
<tr>
<td><strong>OK</strong></td>
<td>All other conditions</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Two types of road condition alerts are shown in the display. The most common alert type is the road condition alert based on mobility. The other alert type is the road condition alert generated when the road temperature is expected to fall below freezing for a wet road.

Road condition alerts are displayed in three ways:

1) A road condition alert time bar shows the worst road condition alert category for the roads in the currently selected maintenance area during three forecast time ranges: 0-12 hours, 12-24 hours, and 24-48 hours.

2) A button below the alert time bar turns red whenever the road temperature of a wet road is expected to fall below freezing for any of the roads within the currently selected maintenance area in the next 3 hours.

3) Colored road segments show the road condition alert category for the selected time when one of the “Road Forecasts” variables is enabled in the map. All road condition alerts are based on the predicted road condition after any user-selected treatment has been applied.

4) Tooltip inspection of road segments on the map yields a color bar indicating the road condition alert category at every forecast hour.

The road alert category is derived from the net mobility value for each road. Table 19.2 describes the default mappings of net mobility values to road alerts. The actual thresholds are determined from the level of service attribute of each route definition in the road configuration shapefiles. See Section 19.5.3.2 for more information on the road configuration shapefiles and Appendix C for more information on net mobility.

<table>
<thead>
<tr>
<th>Road Alert Category</th>
<th>Default Mobility Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extreme</td>
<td>0 – 0.25</td>
</tr>
<tr>
<td>Poor</td>
<td>0.26 – 0.50</td>
</tr>
<tr>
<td>Marginal</td>
<td>0.51 – 0.75</td>
</tr>
<tr>
<td>OK</td>
<td>0.76 – 1.0</td>
</tr>
</tbody>
</table>

Blowing snow alerts are generated for each weather forecast site, for each forecast hour. The algorithm for calculating the blowing snow alert categories is contained in the RCTM; alerts are received from the TUNL and are shown without any re-interpretation. The alerts fall into four categories of likelihood: High, Medium, Low, and None.

Blowing snow alerts are shown in 3 ways:

1) A blowing snow alert time bar shows the worst blowing snow alert category for the weather sites in the currently selected maintenance area during three forecast time ranges: 0-12 hours, 12-24 hours, and 24-48 hours.
2) A blowing snow alert time bar shows the blowing snow category for each hour for the currently selected route when a route is selected in the “Selected Plow Route” combo box in the lower part of the display.

3) The same blowing snow alert time bar described in 2) is shown in the “Event Summary” dialog of each route.

Bridge frost alerts are generated for each road segment, for each forecast hour. The algorithm for calculating the bridge frost alert categories is in the RCTM; alerts are received from the TUNL and are shown without any re-interpretation. The alerts fall into the same four categories of likelihood: High, Medium, Low, and None.

Bridge frost alerts are shown in 2 ways:
1) A bridge frost alert time bar shows the worst bridge frost alert category for the road segments in the currently selected maintenance area during three forecast time ranges: 0-12 hours, 12-24 hours, and 24-48 hours.

2) A bridge frost alert time bar shows the bridge frost category for each hour for the currently selected route when a route is selected in the “Selected Plow Route” combo box in the lower part of the display.

All of the datasets listed in the Map Products panel to the left of the map may be independently enabled and disabled. The State Zone Alerts combo box controls the visibility of the color-coded alert zones. When they are enabled, these zones obscure the shaded relief background map. The Weather Forecasts combo box controls the visibility of the weather variables. Only one weather variable may be visible at a time. The weather variables are shown as colored dots on the map. The Road Forecasts combo box controls the visibility of the road variables. Only one road variable may be visible at a time. The road variables are shown as colored segments on the map. The Point Observations combo box controls the visibility of the observed variables. Only one observed variable may be visible at a time. The observed variables are shown as colored squares on the map. The Area Observations combo box controls the visibility of the radar and satellite datasets. Only one of these datasets may be visible at a time. When one of the satellite datasets is visible, it obscures the shaded relief background map. Depending on the extent of radar reflectivity, it may also obscure the shaded relief background map.

For the Colorado demonstration, weather forecasts are generated for 297 forecast points within the state, and road conditions are forecasted for 19 road segments.

Road treatment functionality is provided in the Treatment Selector dialog. This dialog allows the user to get a recommended treatment (based on Rules of Practice in the RCTM) for each of the road segments modeled in the MDSS system. It also allows the user to test user-specified treatments for each segment. If the user is satisfied with the results of a treatment scenario, the user can select it for use in operations. The user may also opt to select one of the other scenarios for use in operations. In either case, the display adjusts the road condition alerts to reflect the predicted road conditions based on the selected treatment.
A summary of predicted weather and road conditions is provided in the Event Summary dialog. There is an Event Summary dialog available for each route in the system. The dialog shows the relative probabilities of rain, snow, and ice, as well as the total probability of precipitation and declared precipitation type, if any. The dialog also shows total snow accumulation on the road with no new treatments, total snow accumulation assuming the ground is cold and no melting occurs, road temperatures with recommended treatment, and wind speeds for each hour, as well as 48-hr maximums and minimums. Finally, the dialog provides blowing snow alert categories and the system's treatment recommendation. The dialog contents may be printed.

The Treatment History dialog for each route displays the recommended and last-selected treatments for the current 3-hr run and the two previous 3-hr runs. The information is presented in graphical or textual format, as desired. The dialog contents may be printed.

The Configuration menu provides access to several useful features. The Show Trucks menu item enables the display of AVL data for all trucks which have reported within the desired time range: the last 12 hours, the last 24 hours, or any time. Only the latest report from each truck is shown. Tooltip inspection of trucks yields a table containing all available details from the truck report. The Show Values menu item enables the rendering of their numerical values alongside the colored indicators of the map.

The Reset Routes menu item of the Configuration dialog allows users to clear a road after a storm. Long after a storm has completed, it is possible that the MDSS's assumed current road conditions have drifted from reality. In this case, it may be desirable to reset the system's initial conditions to a state consistent with a clear road and no residual chemical concentration. Routes may be reset individually or as a group. The reset routes' initial conditions are reset for the next 3-hr system run.

19.5 Dependencies

19.5.1 Java Libraries

The MDSS display depends on several open source and/or freely available Java libraries for processing data. All of these Java Archive files can be found on the MDSS CD in /src/apps/display/java/ext_lib. License files for third-party software can be found in the /copyrights directory. Descriptions of the contents of each library follow:

19.5.1.1 commons-httpclient-3.0-rc2-all.jar

The file ‘commons-httpclient-3.0-rc2-all.jar’ contains classes for efficient HTTP networking. It is used in compliance with the license file provided. The commons-httpclient archive is freely available from the Apache Software Foundation at

http://jakarta.apache.org/commons/httpclient
19.5.1.2 **jdom.jar**

The file ‘jdom.jar’ contains APIs (Application Programming Interface) for parsing XML configuration files. jdom.jar is used in compliance with the license file provided. It is freely available from the JDOM Project at:

[http://www.jdom.org](http://www.jdom.org)

19.5.1.3 **jaxen-full.jar**

The file ‘jaxen-full.jar’ contains a Java XPath engine. Jaxen is a universal object model walker, capable of evaluating XPath expressions across multiple models. It works in conjunction with JDOM to facilitate XML parsing. jaxen-full.jar is used in compliance with the license file provided. It is freely available from the Jaxen project at:

[http://www.jaxen.org](http://www.jaxen.org)

19.5.1.4 **saxpath.jar**

The file ‘saxpath.jar’ contains the “Simple API for XPath.” It provides a single point for XPath expression parsing. This project has since been assimilated into the newer version of jaxen. jaxen.jar is used in compliance with the license file provided. It is freely available from an archive at:


19.5.1.5 **netcdf-2.2.18.jar**

The file ‘netcdf-2.2.18.jar’ contains the NetCDF-Java Library (version 2). It is a Java interface to NetCDF files, built on a package for multidimensional arrays of primitive types. This library is required for processing radar and satellite datasets. netcdf-2.2.18.jar is used in compliance with the license file provided. It is freely available from the Unidata program of the University Corporation for Atmospheric Research at:

[http://www.unidata.ucar.edu/software/netcdf-java](http://www.unidata.ucar.edu/software/netcdf-java)

19.5.1.6 **nlog4j-1.2.21.jar**

The file ‘nlog4j-1.2.21.jar’ contains the logging infrastructure for plugging the netCDF Java library into the standard log4j logging framework. nlog4j-1.2.21.jar is used in compliance with the license file provided. It is freely available from the Unidata program of the University Corporation for Atmospheric Research at:

[http://www.unidata.ucar.edu/software/netcdf-java](http://www.unidata.ucar.edu/software/netcdf-java)
19.5.1.7 shapefile-2.0-B2.jar

The file ‘shapefile-2.0-B2.jar’ contains the shapefile module of the GeoTools library. GeoTools is an open source library which provides standards compliant methods for the manipulation of geospatial data. shapefile-2.0-B2.jar is used in compliance with the license file provided. It is freely available from the GeoTools Project at

http://geotools.codehaus.org

19.5.1.8 core-2.0-B2.jar

The file ‘core-2.0-B2.jar’ contains the core classes of the GeoTools library. GeoTools is an open source library which provides standards compliant methods for the manipulation of geospatial data. core-2.0-B2.jar is used in compliance with the license file provided. It is freely available from the GeoTools Project at

http://geotools.codehaus.org

19.5.1.9 resources-2.0-B2.jar

The file ‘resources-2.0-B2.jar’ contains required resources for the GeoTools library. GeoTools is an open source library which provides standards compliant methods for the manipulation of geospatial data. resources-2.0-B2.jar is used in compliance with the license file provided. It is freely available from the GeoTools Project at

http://geotools.codehaus.org

19.5.1.10 JTS-1.3.jar

The file ‘JTS-1.3.jar’ contains the JTS Topology Suite (JTS). It provides an API for spatial object model and fundamental geometric functions. It implements the geometry model defined in the OpenGIS Consortium “Simple Features Specification” for SQL.APIs. This library is required for shapefile processing. JTS-1.3.jar is used in compliance with the license file provided. It is freely available from Vivid Solutions at:

http://www.vividsolutions.com/jts/jtshome.htm

19.5.2 Geographic Location Configuration

The display's geographic location is determined by several configuration files. In order to localize the display for another region, these files need to be modified. All files are ASCII text. The sensor configuration and maintenance district files are XML. Examples of all of these files can be found on the MDSS CD in the directory:

/src/apps/display/src/edu/ucar/rap/mdss/apps/xdss07
The configuration files are all loaded at the start of the display. After they have been loaded, their contents are stored in memory. Modifications to the configuration after the display has been started have no effect.

### 19.5.2.1 Weather and Road Forecast Sites

The site list configuration file identifies the sites for which the display expects weather forecasts. It provides a mapping from site IDs to the sites’ names and locations. A sample of this file can be found on the MDSS CD as:

```
/src/apps/display/src/edu/ucar/rap/mdss/apps/xdss07/sites.asc
```

This is the same file used by the MDSS system. Its format is documented in Section 9.2. The file contains one site definition per line, with each site definition containing multiple semi-colon-separated fields. The display is only concerned with the following fields of each site definition:

- **SITE_ID** - a unique 8-digit number
  - This number is the MDSS ID described below in Section 19.5.2.3
- **LAT** - latitude in decimal degrees (negative for south latitude)
- **LON** - longitude in decimal degrees (negative for west longitude)
- **ELEV** - elevation above sea level in meters
- **NAME** - the display name of the site

### 19.5.2.2 Observation Sites

The observation sites are configured by two files, a site list similar to the weather forecast site list described above and an XML document describing the sensors which exist at each site.

The observation site list configuration file identifies the sites for which the display expects observation data. An example of this file may be found on the MDSS CD at:

```
/src/apps/display/src/edu/ucar/rap/mdss/apps/xdss07/rwis.asc
```

This configuration file may contain many sites which are also in the weather forecast site list, but this is not required. The two files are independent. Every site listed in the observation site list file should have an entry in the sensors configuration file described below.

The sensors configuration file is an XML file containing information about which sensors are present at each of the observation sites listed in the observation site list file. An example of this file may be found on the MDSS CD at:

```
/src/apps/display/src/edu/ucar/rap/mdss/apps/xdss07/sensors.xml
```

The schema for this file is as follows:

```xml
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```
The Sensor “name” attributes are used to identify the variables in the data sets loaded from the server. These names must exactly match the names of the variables (“variable_name”), as described in Section 19.6.2.5, “Input File Format.”

The “priority” attributes determine which sensor is shown on the map when several sensors of the same “type” exist. The lowest priority sensor is shown. Other sensors’ values appear in the observation sites' tooltips, but not on the map.

The “description” attributes are displayed next to the sensor information in the observation sites' tooltips. These may be an empty string.
19.5.2.3 MDSS Places Configuration

The topology of MDSS weather forecast sites, road segments, and observation sites within maintenance districts and alert zones is determined by the places configuration file. MDSS categorizes places as being one of:

- **WxSite** – a site for which a weather forecast will be generated
- **ObsSite** – a site at which observations are made. May also be a WxSite
- **RoadSegment** – a segment for which a road condition forecast will be generated
- **ForecastZone** – an area containing other MDSS Places, who’s WxSites can be aggregated to derive a zone weather forecast.
- **MaintenanceDistrict** - an area containing other MDSS Places, from who’s WxSites and RoadSegments road and weather alerts can be derived.

The areal MDSS Places, ForecastZone and MaintenanceDistrict, may be nested within each other. The point MDSS Places, WxSite, ObsSite, and RoadSegment, may be referenced within different areal MDSS Places. These arrangements combine to allow the definition of topologically complex areas of responsibility. Future MDSS localization enhancements could include the definition of county management areas, emergency services zones, and/or disaster areas, overlapping road MaintenanceDistricts.

An example of this file can be found on the MDSS CD as:

/src/apps/display/src/edu/ucar/rap/mdss/apps/xdss07/placesConfig.xml

The schema for this file is as follows:

```xml
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="MDSSRoot">
    <xsd:complexType>
      <xsd:sequence minOccurs="1" maxOccurs="unbounded">
        <xsd:choice>
          <xsd:element ref="MaintenanceDistrict"/>
          <xsd:element ref="ForecastZone"/>
          <xsd:element ref="RoadSegment"/>
        </xsd:choice>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="MaintenanceDistrict">
    <xsd:complexType>
      <xsd:attribute name="id" type="xsd:int" use="required"/>
      <xsd:attribute name="name" type="xsd:string" use="required"/>
      <xsd:sequence minOccurs="1" maxOccurs="unbounded">
        <xsd:choice>
          <xsd:element ref="MaintenanceDistrict"/>
          <xsd:element ref="ForecastZone"/>
          <xsd:element ref="RoadSegment"/>
          <xsd:element ref="ObsSite"/>
        </xsd:choice>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```
The “id” attributes are used to uniquely identify the MDSS place. Internally to the display code, MDSS places are referred to only by their IDs. As a result, each MDSS place requires an ID attribute and these IDs must be unique across all MDSS places. The single exception to this rule is co-located WxSites and ObsSites. Currently, these will have the same ID but be aware of the duality of their identity. There is nothing in the configuration file to prevent the same ID from being applied to multiple places, so a convention is used to minimize the potential for error. IDs for each type of MDSS place are assigned from a range.
MaintenanceDistrict - IDs from 100 to 999
ForecastZone - IDs from 1000 to 9999
RoadSegment - IDs from 10000 to 99999
WxSite and ObsSite - IDs from 10000000 to 99999999

The “name” attributes are used to assign human-readable names to the MDSS places. In most cases, these are the names which will be used in the display. In the case of WxSite and ObsSite, however, the display names are taken from the weather sites list file (sites.asc) and observation sites list file (rwis.asc), as described in Sections 19.5.2.1 and 19.5.2.2.

The road conditions for a road segment are currently forecasted based on the weather at a single “representative weather site.” There is currently only one representative weather site for each road segment. The association of a road segment with its representative weather site is defined in the MDSS places configuration file by assigning the representative weather site as a child element of the road segment, as shown below:

```xml
<RoadSegment id="10004" name="I-70 at Vail Pass">
  <WxSite id="72469100"/>
</RoadSegment>
```

19.5.3 Shapefile-based Definitions

Complex geographic elements, which cannot be represented by a single point, are now represented in shapefiles. Many road maintenance agencies already have their road segment and alert zone information in shapefile format. In most cases, these existing shapefiles may be used to configure the MDSS display with the single minor modification described below.

The shapefile features are mapped to other objects in the display via an attribute named “MDSS_ID” and their names are determined from the attribute “MDSS_NAME.” Alert zones and road segments may be added to MDSS simply by adding these attributes to their feature attribute tables in a GIS editing tool such as ArcMAP. The value of the MDSS_ID attribute should be the MDSS place ID defined in the placesConfig.xml file. The value of the MDSS_NAME feature will determine the feature’s name in the display.

19.5.3.1 Alert Zones

Alert zones are areas over which a composite weather alert is generated. Zone alerts are computed for three time periods over 48 hours, as described in the display Function Section 19.4. The alerts indicate the worst weather condition forecasted for any of the weather forecast sites which lie within the bounds of each zone. These zones are represented as polygon objects in the zone alert configuration shapefile. An example of this configuration can be found on the MDSS CD as:

```
/src/apps/display/data/maps/shapefile/political/northAmerica/usa/colorado/counties.*
```
In this case, the alert zones coincide with Colorado’s counties. The MDSS_ID and MDSS_NAME attributes are set for each of the zone polygons, as shown below:

<table>
<thead>
<tr>
<th>FID</th>
<th>Shape</th>
<th>NAME</th>
<th>CNTY_FIPS</th>
<th>FIPS</th>
<th>SOMI</th>
<th>MDSS_ID</th>
<th>MDSS_NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>Polygon</td>
<td>Rio Blanco</td>
<td>103</td>
<td>06103</td>
<td>3222.9</td>
<td>1052</td>
<td>Rio Blanco</td>
</tr>
<tr>
<td>10</td>
<td>Polygon</td>
<td>Garfield</td>
<td>045</td>
<td>08045</td>
<td>2955.8</td>
<td>1023</td>
<td>Garfield</td>
</tr>
<tr>
<td>11</td>
<td>Polygon</td>
<td>Mesa</td>
<td>077</td>
<td>08077</td>
<td>3341</td>
<td>1039</td>
<td>Mesa</td>
</tr>
<tr>
<td>12</td>
<td>Polygon</td>
<td>Chaffee</td>
<td>016</td>
<td>08016</td>
<td>1315</td>
<td>1008</td>
<td>Chaffee</td>
</tr>
<tr>
<td>13</td>
<td>Polygon</td>
<td>Boulder</td>
<td>013</td>
<td>08013</td>
<td>740.5</td>
<td>1006</td>
<td>Boulder</td>
</tr>
<tr>
<td>14</td>
<td>Polygon</td>
<td>Summit</td>
<td>117</td>
<td>08117</td>
<td>619.3</td>
<td>1059</td>
<td>Summit</td>
</tr>
<tr>
<td>15</td>
<td>Polygon</td>
<td>Eagle</td>
<td>037</td>
<td>08037</td>
<td>1891.8</td>
<td>1019</td>
<td>Eagle</td>
</tr>
<tr>
<td>16</td>
<td>Polygon</td>
<td>Jefferson</td>
<td>069</td>
<td>08059</td>
<td>774.2</td>
<td>1030</td>
<td>Jefferson</td>
</tr>
<tr>
<td>17</td>
<td>Polygon</td>
<td>Park</td>
<td>083</td>
<td>08083</td>
<td>2210.7</td>
<td>1047</td>
<td>Park</td>
</tr>
<tr>
<td>18</td>
<td>Polygon</td>
<td>Lake</td>
<td>065</td>
<td>08056</td>
<td>383.9</td>
<td>1034</td>
<td>Lake</td>
</tr>
<tr>
<td>19</td>
<td>Polygon</td>
<td>Pitkin</td>
<td>067</td>
<td>08037</td>
<td>973.1</td>
<td>1049</td>
<td>Pitkin</td>
</tr>
<tr>
<td>20</td>
<td>Polygon</td>
<td>Delta</td>
<td>029</td>
<td>09029</td>
<td>1140.4</td>
<td>1015</td>
<td>Delta</td>
</tr>
<tr>
<td>21</td>
<td>Polygon</td>
<td>Gilpin</td>
<td>047</td>
<td>08047</td>
<td>150.3</td>
<td>1024</td>
<td>Gilpin</td>
</tr>
</tbody>
</table>

19.5.3.2 Road Segments

Road segments are the lines over which road conditions are forecasted. The representative weather point used to forecast conditions is determined from the MDSS place configuration file described above. The locations over which to render the roads in the display and the level of service to require on each road segment are represented either as polyline or polygon objects in road segment configuration shapefiles. Examples of both configurations can be found on the MDSS CD as:

/src/apps/display/data/maps/shapefile/transport/northAmerica/usa/colorado/mdssRoadAreas.*
and
/src/apps/display/data/maps/shapefile/transport/northAmerica/usa/colorado/mdssRoadSegments.*

These files allow road conditions to be represented on the map either along a segment or over an area. The latter style may be useful for metropolitan areas where the variation in weather conditions, pavement type, traffic volume, and treatment plans is minimal.

In addition to the MDSS_ID and MDSS_NAME attributes, road segment configuration files require an attribute for the level of service. This attribute determines the mobility values which form the thresholds between road alert conditions of Extreme, Poor, Marginal, and OK, as described in the display Function Section 19.4. The level of service attribute is named “MDSS_LOS.” It’s value is required to be a comma-separated list of 3 mobility values. The mobility values must be increasing between 0 and 1, as shown below:
19.5.4 Runtime Environment

The display system can be run on any system which has Java 1.6.x or later installed. Platforms on which the system has been tested include Solaris, Solaris for Intel, Debian Linux, RedHat Linux, Windows NT, 2000, and XP.

The Java 1.6.x or a later runtime environment must be installed on the hardware running the display application. To download and install this software package, see the Sun Microsystems website at: http://java.com. Click on the button for the “Free Java Download.” At the time of this writing, the latest version is Java 1.6.0_01. It includes the latest distribution of Java WebStart.

19.5.5 Java WebStart Deployment

The display system must be deployed to other clients via a web server using Java WebStart, available from Sun Microsystems. WebStart is installed on the client machines of users downloading the display system as described above. The only change necessary to the web server deploying the application is to make an entry in the configuration file mapping a new MIME type to the .jnlp file extension used for deploying applications in this way. For the Apache web server, the following entry is used in the httpd.conf file:

```
AddType application/x-java-jnlp-file JNLP
```

Instructions for downloading and installing Java WebStart on the client computers are available at the Sun Microsystems web site:


For all required 1.6.x and later Java versions, JavaWebStart is bundled with the Java virtual machine, so no additional download is necessary. It is possible to streamline the setup process for display users by providing a web page on the server that uses javascript to test whether WebStart is installed on their computer, and directing them to the appropriate URL to download the necessary WebStart application. Documentation for how to accomplish this is available at:
The web server, where the Java archives are placed for access by Java WebStart clients, must contain an xml document showing the configuration of the MDSS display application. This document, known as a “.jnlp file” (Java Network Launching Protocol) tells Java WebStart how to invoke the display application. Users with Java WebStart properly installed will have Java WebStart automatically invoked by their browsers when they browse to the .jnlp file.

**Note:** When downloading Java WebStart, downloading the display application, and running the live MDSS display, web filtering software (e.g. WebNanny, Cybersitter) should be temporarily disabled. Web filtering software corrupts the authentication encryption of the application’s .jar files when it attempts to view its contents. Java detects this tampering and will refuse to run the application.

Below is an example .jnlp file, used to deploy the prototype application.

```xml
<?xml version="1.0" encoding="utf-8"?>
<!--
 A JNLP File for MDSS Application.
 @version 7.0
 @date 2007/3/16
 @author Arnaud Dumont
 -->

<jnlp spec="6.0+">
  codebase="http://www.rap.ucar.edu/projects/rdwx_mdss/xdss"
  href="xdss.jnlp">

  <information>
    <title>MDSS Prototype</title>
    <vendor>NCAR - RAL</vendor>
    <homepage href="http://www.rap.ucar.edu/projects/rdwx_mdss/"/>
    <description kind="tooltip">Live Colorado MDSS Display</description>
    <description kind="one-line">Live Colorado MDSS Display '06-'07</description>
    <description kind="short">A Maintenance Decision Support System for Colorado Highway Authorities</description>
    <icon href="icon.gif"/>
    <icon kind="splash" href="splash.gif"/>
  </information>

  <security>
    <all-permissions/>
  </security>

  <resources>
    <j2se version="1.6+" initial-heap-size="50331648"
      max-heap-size="536870912"/>
    <jar href=" xdss07.jar" download="eager"/>
    <jar href="JTS-1.3.jar" download="eager"/>
    <jar href="commons-httpclient-3.0-rc2-all.jar" download="eager"/>
  </resources>

</jnlp>
```
19.6 Interfaces

19.6.1 Command Interface

The display system can be run manually from the command line through Java WebStart. The URL of the application may be provided as a command-line argument, as described in the Java WebStart documentation. For example:

```
<path to root of Java installation>/bin/javaws http://domain.com/path/xdss.jnlp
```

Once the application is downloaded using Java WebStart, its jar files are cached on the local computer. When the application is restarted, Java WebStart checks whether the user’s version of the display system is up to date, and automatically downloads a new version if necessary. No download will take place if the user’s version is up to date.

There are several other means of launching the MDSS display and Java WebStart. When a web browser opens the display’s .jnlp file, the MIME type of that file will automatically trigger Java WebStart to launch the display. After the MDSS application has completed its initial download, a desktop shortcut may be created to provide easy launching in the future. Lastly, the display may be restarted by double-clicking its name in the WebStart application cache after WebStart is launched with the command:

```
<path to root of Java installation>/bin/javaws -viewer
```

19.6.2 Input

The MDSS display application reads several binary data files to determine the weather forecast, road condition forecasts, and RWIS observations. All of these files have the same format, as described below. These files are created by the Forecast and Observation Data Re-formatters.
A binary format was chosen because it is much faster to cast binary data than to read ASCII data and convert it to numerical types. A binary format allows approximately 10-fold decrease in the time it takes to ingest a data file. The binary files are also smaller than their ASCII representations would be, significantly reducing network transmission times.

19.6.2.1 Weather Forecast Data File

A single file contains all the weather forecast data from a given forecast run for the display. The file is retrieved from the data server via an apache web server. An example of such a file on the server is:

/data/conv_wx/20070412/conv_wx.20070412.1800.bin

19.6.2.2 ESS Observation Files

A single file contains ESS observations for the latest time for the display. The file is retrieved from the data server via a web server. An example of such a file on the server is:

/data/conv_meso/20070412/conv_meso.20070412.1800.bin

19.6.2.3 Road Condition Forecast Files

Three treatment type data files contain the standard road condition forecast data – one for the road condition with no treatment, one for the road condition with the recommended treatment, and one for the current (selected) treatment. Three recommended and selected treatment files and one no-treatment file are retrieved from the data server via an apache web server. The three recommended and selected treatment files are from the most recent model run and the two previous runs. The no-treatment file is from the most recent model run. Examples of such files on the server are:

/data/conv_no_tmt/20070412/conv_no_tmt.20070412.1800.bin - No Treat.
/data/conv_rec_tmt/20070412/conv_rec_tmt.20070412.1800.bin - Rec. Treat.
/data/conv_cur_tmt/20070412/conv_cur_tmt.20070412.1817.bin - Cur. Treat

19.6.2.4 User-Specified Treatment Road Condition Forecast Files

Files are provided to the display system when user-specified treatments are submitted to the RCTM module via remote communications. These files contain the solution to such ‘what-if’ scenarios. These files are named based on the following convention:

/data/user_treatment/<host>/<site_id>/conv_rc_ut/<YYYYMMDD>/conv_rc_ut.<YYYYMMDD>.<HHMMSS>.bin

Where:
<host> is the hostname of the client requesting the user treatment
<YYYYMMDD> are the Year, Month, and Day of the forecast start, in UTC
<HHMMSS> is the hour, minute, and seconds of the data (UTC)
<site_id> is the location number of the forecast site, currently one of:

72469108 – E470's MSSA Route site
72565010 – E470's MSSC Route site
72565012 – E470's MSSD Route site
72469110 – E470's MSSE Route site
72469115 – City and County of Denver’s District 1 site
72469119 – City and County of Denver’s District 2 site
72469120 – City and County of Denver’s District 3 site
72469122 – City and County of Denver’s District 4 site
72469132 – City and County of Denver’s District 5 site
72565014 – City and County of Denver’s District 6 site
72469100 – CDOT's I-70 Vail Pass Route site
72570038 – CDOT’s I-70 Dowd Junction Route site
72570039 - CDOT's I-70 Wolcott Route site
72469101 – CDOT’s I-70 Floyd Hill Route site
72469099 – CDOT’s I-70 Genesee Route site
72469081 – CDOT’s I-70 Morrison Route site
72469074 – CDOT’s I-25 @ C-470 Route site
72469103 – CDOT’s I-25 @ Exit 191 Route site
72469102 – CDOT's I-25 @ Surrey Ridge site

19.6.2.5 Input File Format

All the weather and road condition files are expressed using the same binary format. The format is as follows:

The following sequence is repeated for each forecast station:

```c
int station_id
```

The following sequence is repeated for each variable for the station:

```c
char[40] variable_name
int num_days
int vals_per_day
double base_time
int data_type
float[ num_days * vals_per_day] values
char[40] "#" – indicates end of a station’s data
```

Where:

Station_id is a numerical station identifier
variable_name is one of:

"TempF"
"dewptF"
"rh_pct"
"wind_speed_mph"
variable_name for RWIS is one of the names defined in the sensors.xml file, found in /src/apps/display/java/src/edu/ucar/rap/mdss/apps/xdss07, currently:
  “roadTemperature1”
  “roadTemperature2”
  “roadTemperature3”
  “roadTemperature4”
  “roadSubsurfaceTemp1”
  “roadSubsurfaceTemp2”
  “temperature”
  “relHumidity”
  “windSpeed”
  “windDir”

num_days is the number of full days in the forecast
vals_per_day is the number of forecast values provided per day
base_time is the UNIX time of the first data value, in UTC time
data_type is the integer type of values in the values array:
  0 = float (where every value is encoded in 4-bytes)
  1 = string (where the first 2 bytes of each value are a short integer indicating the number of 1-byte characters in the string which follows)
values is the array of data values for the variable

19.7 Output

The display application does not generate any output files. Debug information is available from within the Java WebStart Java console. To enable this output from within WebStart, start WebStart with the “-viewer” option, as described above. In the “Java Control Panel,” select the “Advanced” tab. Open the “Java Console” settings category and click the “Show console” radio button.

20 Future Enhancements and Refinements

Revisions to MDSS Release-5.0 will be made, if necessary, based on the results of additional field demonstrations and these releases will be labeled Release-5.x. More significant software enhancements are not anticipated at this time.
Appendix A:
Road Temperature Module

The Road Temperature model used within the RCTM is METRo, the Model of the Environment and Temperature of Roads. METRo is a one-dimensional land-surface model developed at Environment Canada. With the help of observations provided by a road weather station (road weather information system, RWIS) and the atmospheric forecast, METRo can, amongst other things, predict the road conditions with particular interest such as: freezing rain, accumulation of snow, frost and thaw.

METRo was developed in 1999. It links together modules written in a number of computer languages. These include Python, C++, and FORTRAN. It reads in three input files and produces two output files. The input files describe the station characteristics, the recent observational history, and the weather forecast respectively. The output files are the road condition forecast and quality-controlled weather forecast data used in making the road condition forecast. These files are all in XML format. The reason for choosing XML was to make the data human readable, self-documented, and, more importantly, allow us to use many available tools for manipulating the data. The advantages of this human-readable format are offset by the relatively slow parsing of input XML files and generation of output XML files.

METRo, as used in this project, is a single run application. That is, it cannot be used for iterative use on a number of sites. Within the RCTM, configuration files are generated and METRo is called once for each forecast site. It actually may be called more than once per site if treatments are called for. Significant re-workings would have been necessary to make METRo run as a function call within the RCTM. For this reason, METRo is run as a separate process that is invoked within the RCTM. The RCTM creates the METRo input files, starts METRo, then waits until the METRo process has finished and parses its output file.

METRo requires three input files. The first contains a recent history of the observations at the forecast site. In the case that observations are not available from the site, pseudo-observations must be used. One solution is to use data from previous forecasts. The second input file describes the station configuration. This file contains information describing the site, such as whether it is a roadway or bridge, the makeup of the subsurface (materials and thicknesses), and the station location.

Unlike its predecessor within the MDSS (SNTHERM), METRo allows less specification of material properties. That is, the user is not required to know and/or modify the properties of asphalt used on each route. Instead material properties are predefined within METRo and the user can only specify the material type and cannot change the properties without significant modification of the METRo source code.

METRo is free and open source software. The source code can be downloaded and modified as the user feels appropriate. There is also a METRo Wiki web site that contains FAQ, bug reports, wish lists, and troubleshooting and developers’ interactions pages.
For detailed technical information on METRo, the reader is directed to Environment Canada (see appendix F for contact information) and the METRo web site listed below.

**References**


METRo web site: [http://home.gna.org/metro](http://home.gna.org/metro)

Appendix B:
Chemical Concentration Module

The chemical concentration code is part of the Road Condition and Treatment Module. The code can be found on the MDSS CD in the directory /src/apps/roadcond_mdss. The file of interest is ChemConc.cc.

B.1. Algorithm Identifier

ChemConc

B.2. Type

Subprocess

B.3. Software

ChemConc.cc

RulesOfPractice.hh

B.4. Conceptual Overview

The ChemConc algorithm is designed to estimate the chemical concentration of anti-icing and de-icing chemicals as they are applied during the course of a winter storm. There are a wide variety of anti-icing chemicals available to operators; this algorithm currently supports the chemicals shown in Table B1. Maintenance operators choose different types and forms (dry, pre-wet, and liquid) of anti-icing chemicals based, in part, on how well the characteristics of a particular chemical match the forecasted weather conditions.

Table B1. Chemicals supported for user selection

<table>
<thead>
<tr>
<th>Chemicals Supported</th>
</tr>
</thead>
<tbody>
<tr>
<td>NaCl (Sodium Chloride)</td>
</tr>
<tr>
<td>CaCl₂ (Calcium Chloride)</td>
</tr>
<tr>
<td>MgCl₂ (Magnesium Chloride)</td>
</tr>
<tr>
<td>CALIBER (brand name)</td>
</tr>
<tr>
<td>Ice Slicer (brand name)</td>
</tr>
</tbody>
</table>

An essential characteristic of anti-icing chemicals is their ability to reduce the freezing point of water. The phase diagram shown in Figure B1 illustrates the freezing point depression characteristics for various concentrations of salt solution. At a solution concentration of 23.3% the freezing point of water is reduced to –6.02 degrees F. This point represents the peak freezing point depression for this chemical and is called the eutectic point. Solution and temperature combination below the bounding curve on the left will result in ice formation; the curve represents the chemical’s solution point. Conversely, solution and temperature combinations that fall to the right of the bounding curve on the far
side of the diagram will result in unabsorbed chemical. This curve is called the saturation curve. Ideally, anti-icing practices attempt to maintain the chemical concentrations between the solution point (no ice) and the saturation point (no wasted chemical).

Figure B1. Phase diagram for water/sodium chloride solutions.

In addition, practical studies have shown that the true range of anti-icing chemical effectiveness is bounded by the 50% level (½ point) of the eutectic concentration. For NaCl, the ½ point is 11.65% corresponding to a temperature of –10.0 deg C (14 deg F). The algorithm takes this ½ point into account (*C_NACL_HALFPOINT_TEMP*) when calculating the length of time chemicals will be effective. Finally, there are two parameters (*P_CHEM_SAFE_DELTA* & *P_CHEMRATE_SENSITIVITY_PCT*) that may be used to adjust the sensitivity of the algorithm by adding a temperature buffer above or below the eutectic curve. As illustrated in Figure B2, positive values of either parameter will make the system recommend higher treatments, while negative values will result in lower recommended treatments.
Figure B2. Illustration of adjusting sensitivity factors for calculating appropriate chemical rate applications. The red arrow shows adjustments in P_CHEM_SAFE_DELTA and the green arrow adjustments to P_CHEMRATE_SENSITIVITY_PCT.

Figure B3 illustrates the life history of anti-icing chemicals as they are applied before, during and after a storm. Spreader trucks (or tankers in the case of liquid chemicals) are used to spread the selected anti-icing chemical. As the truck delivers the chemical, the force of the compound hitting the road causes some of the chemical to fall off the road (road splatter). Additionally, winds may blow the chemical off the pavement before it has had a chance to stick to the road. Once the chemical is applied, routine road traffic may also scatter the chemical off the roadway (traffic splatter).
Figure B3. Overview of anti-icing (de-icing) chemical application and dilution process.

As precipitation begins to occur, the chemical mixes with the available surface water to form a chemical solution. Some of the solution is lost as the liquid drains from the roadway. The strength of this solution is directly calculated from the amount of chemicals dropped and the precipitation that falls on the road surface. The anti-icing and de-icing effectiveness of the solution is determined by knowing the concentration of the solution and the temperature of the solution (pavement temperature).

As more precipitation falls, the chemical concentration continues to decrease. In addition, even without additional precipitation, the solution will slowly evaporate from or drain off the road surface. Eventually, the chemical concentration drops to a level that is insufficient to prevent ice/snow build-up (below the solution point in the phase diagram). Once precipitation stops, water or solution remains on the road surface but continues to drain from the road surface or evaporate (process not shown in figure). The ChemConc algorithm has been designed to capture the essential elements of the chemical application/dilution process.

B.5. Algorithm Description

The ChemConc algorithm is one of several components within the RCTM. The purpose of the ChemConc algorithm is to track the forecasted level of anti-icing chemicals (dry or in solution) and water on the road surface. The ChemConc routine can be triggered in three ways. First, the RCTM may recommend a new treatment (in this case ChemConc begins
processing at the treatment timestep), calculates the net rate of chemical delivery at the treatment hours and then reduces the available chemical and water over time based on dilution, traffic, runoff and evaporation. Similarly, ChemConc may also be triggered by a user selected treatment requiring the same steps as above. Finally, ChemConc will be called each time the weather forecast variables are updated. In this case, ChemConc starts at the beginning timestep and applies chemicals based on the current treatment plan. Any time the ChemConc algorithm is applied the pavement temperature model is also re-run because the effectiveness of chemicals impacts the level of snow or ice on the road surface, thereby changing the thermal exchange at the road surface.

Figure B4. Road condition treatment module (RCTM) component algorithms.

There are a number of user parameters that may be used to adjust the sensitivity and thresholds of the algorithm for specific operations. These parameters are described in detail in section B.5.c and are referred to as necessary in the following discussion.

The overall functional flow of the algorithm is shown in Figure B5. The algorithm is currently triggered by the RCTM when a user requests treatment options for a specific route and chemical type (UserChemType). The necessary weather variables, road conditions, and treatment recommendations are then sent to ChemConc. If a new treatment is to be applied, then the process starts at the treatment application time as opposed to the start of the forecast period. ChemConc then loops over each forecast time step calculating the chemical concentration at each successive hour by performing the steps described below.

First, the amount of available water is calculated by summing the water left on the road surface in the previous hour (even if it is already in solution) with the precipitation that fell
in the current hour. Parameters are used to control the rate at which new water (Road-WaterImmediateRunoff) and old solution (RoadChemSolutionRunoffPerHour) drain from the road.

Next, there is a series of steps to calculate the amount of available chemicals on the pavement. If this time step is the treatment hour then the ChemRate is set to the recommended treatment application (otherwise it is zero). ChemRate is then adjusted to account for the fraction of the chemical that will be lost due to road splatter and wind effects (RoadChemAppInitialLossRate). This rate of loss is different for liquid (P_InitialChemAppLossRateLiquid) vs. dry and pre-wet (P_InitialChemAppLossRatePrewet) applications. If a previous application of chemical was already made then the process determines the amount of residual chemicals - (ResidualChemicals: chemicals that were not dissolved previously because the chemical concentration was above saturation). Finally, the process determines the amount of chemicals already in solution from previous applications (ChemInSolution). The amount of chemicals available for anti-icing operations is defined as:

$$\text{AvailableChemical} = (1-\text{tfactor})*(\text{ChemRate} + \text{ResidualChemical} + \text{ChemInSolution})$$

where tfactor represents the fraction of chemicals lost from the road surface due to transport from automobiles and trucks on the road (as calculated in the routine CalcTrafficFactor). The nominal impact of traffic on available chemicals is controlled by the parameter NominalTrafficDilution. Increases in traffic levels (Low=1, Medium=2, High=3) result in exponential increases from the nominal dilution value (e.g., a high traffic count is nine times more diluting than a low traffic count).

If there is still residual chemical on the roadway, but the pavement is dry, then the residual chemical is dissipated at a faster rate (25% per hour).

The next series of steps is used to calculate the final concentration level of the solution remaining on the road surface. The nominal chemical concentration is determined by simply dividing the available chemicals by the sum of the available water and chemicals. However, chemicals can only dissolve into water up to their saturation level. Therefore, the nominal concentration is clipped to the chemical saturation point (as calculated in CalcCriticalChemSaturationPoint) and used as the final chemical concentration value. Once precipitation has ceased, the available water is allowed to evaporate from the road surface, eventually the road surface will be considered dry and, therefore safe from re-freezing.

Any surface water remaining on the road is reduced by evaporation if the water is in liquid form (when chemicals are effective or no chemicals are needed). The routine CalcEvaporationRate estimates the evaporation rate from relative humidity and wind speed factors.

One last step is to determine at what time step the chemical concentration will become ineffective. The algorithm currently determines this by finding the first time step where the final chemical concentration is at or below the chemical solution point (as calculated in
The forecasted chemical concentration levels and failure time step are then passed back to the RCTM.

Figure B5. Functional flow diagram of ChemConc algorithm.
B.5.a. Algorithm Inputs

The *ChemConc* algorithm requires a forecast of the precipitation rate expressed as a liquid water equivalent (for example 10” of snow may equal 1.0” of water). It also requires relative humidity and wind speed values. Pavement temperature is another necessary input. If the user is applying a new treatment, then the algorithm needs to know the type, rate and application time of the chemical. The only other input needed is a prediction of the traffic intensity over the forecast period (low, medium or high).

B.5.b. Algorithm Outputs

The *ChemConc* algorithm outputs an estimate of the predicted chemical concentration over the forecast period. In addition, the algorithm predicts the time step where the chemical concentration will become ineffective at preventing ice build-up.

B.5.c. Algorithm Structures

The ChemicalType structure defined in RulesOfPractice.hh holds the parameters/settings applicable to the ChemConc algorithm. The following members are in the structure:

- **UserChemType**: Specify the type of anti-icing (de-icing) chemical to calculate concentrations.
- **UserChemForm**: The application form of the chemical (dry, pre-wet, or liquid).
- **ChemName**: The name of the de-icing chemical (eg. NaCl).
- **NominalTrafficDilution**: The fractional loss of chemicals (off the roadway) caused by automobile and truck traffic on the road. This nominal value is applied to the lowest level of traffic rates (1=low). Increasing traffic (2=med, 3 =high) results in a squared increase of the traffic impact. So a high traffic roadway experiences 9 times the loss of a low traffic roadway. (nominally set to \( P_{\text{NominalTrafficDilution}} \)).
- **RoadChemAppInitialLossRate**: The fractional loss rate of chemicals as they are being applied. This is primarily due to chemicals bouncing off the road surface, but may also be caused by high winds. (nominally set to \( P_{\text{InitialChemAppLoss}} \)).
- **RoadWaterImmediateRunoff**: The fractional loss rate of liquid water (runoff) from the road surface in the hour that the precipitation falls (nominally set to \( P_{\text{RoadwayImmediateRunoff}} \)).
- **RoadChemSolutionRunoffPerHour**: The fractional loss rate of chemical solution (runoff) at each time step. An assumption is made that the water and chemicals in the solution runoff or dissipate at the same rate. (nominally set to \( P_{\text{ChemSolutionRunoff}} \)).
- **MinApplicationRate**: The minimum rate at which the chemical should be applied (nominally set to \( P_{\text{MIN_TREAT}} \)).
MaxApplicationRate: The maximum rate at which the chemical should be applied (nominally set to \texttt{P\_MAX\_TREAT}).

RoundTreatment: When recommending treatment rates, round to this nearest interval (nominally set to \texttt{P\_ROUND\_TREAT}).

Units: The units of the application chemical (\texttt{C\_LBS\_SQFT} [dry or pre-wet] or \texttt{C\_GALS\_LANEMILE} [liquid]).

MinTemp: The minimum temperature at which this chemical should be applied (nominally set to the \(\frac{1}{2}\) point temperature of the chemical \texttt{C\_NACL\_HALFPOINT\_TEMP} for example).

MaxTemp: The maximum temperature at which this chemical should be applied (nominally set to \texttt{P\_MAX\_TEMP}).

ChemSafeTempDelta: A sensitivity adjustment (deg C) for the eutectic curve (nominally set to \texttt{P\_CHEM\_SAFE\_DELTA}).

\textbf{B.5.d. Algorithm Parameters}

\texttt{P\_user\_chemtype}: Specify the type of anti-icing (de-icing) chemical to calculate concentrations. (See Table C1).

\texttt{P\_TREAT\_UNITS}: The name of the units being used within the algorithm (e.g. “Lbs/lane-mile”).

\texttt{P\_MIN\_TREAT}: The minimum rate at which the chemical should be applied (in \texttt{P\_TREAT\_UNITS}).

\texttt{P\_MAX\_TREAT}: The maximum rate at which the chemical should be applied (in \texttt{P\_TREAT\_UNITS}).

\texttt{P\_ROUND\_TREAT}: When recommending treatment rates, round to this nearest interval (in \texttt{P\_TREAT\_UNITS}).

\texttt{P\_InitialChemAppLossRateXXX}: The fractional loss rate of chemicals as they are being applied. This is primarily due to chemicals bouncing off the road surface, but may also be caused by high winds. Where “XXX” is “Dry”, “Prewet” or “Liquid”.

\texttt{P\_RoadwayImmediateRunoff}: The fractional loss rate of liquid water (runoff) from the road surface in the hour that the precipitation falls.

\texttt{P\_RoadwayMinDepthforRunoff}: <Not used> Minimal water/solution depth for runoff to occur (mm).

\texttt{P\_ChemSolutionRunoff}: The fractional loss rate of chemical solution (runoff) at each time step. An assumption is made that the water and chemicals in the solution runoff or dissipate at the same rate.

\texttt{P\_NominalTrafficDilution}: The fractional loss of chemicals (off the roadway) caused by automobile and truck traffic on the road. This nominal value is applied to the lowest level of traffic rates (1=low). Increasing traffic (2=med, 3
=high) results in a squared increase of the traffic impact. So a high traffic roadway experiences 9 times the loss of a low traffic roadway.

P_EvaporationPerHour: The nominal expected fractional rate of evaporation per time step.

P_DryChem_LossRate: The fractional rate of loss of dry chemicals on a dry road per time step.

P_MIN_TEMP: The minimum temperature at which this chemical should be applied (nominally estimated to the ½ point temperature of the chemical). C_NACL_HALFPOINT_TEMP for example).

P_MAX_TEMP: The maximum temperature at which this chemical should be applied (deg C).

P_CHEM_SAFE_DELTA A sensitivity adjustment (deg C) for the eutectic curve.

Table B2. Nominal algorithm parameter values and ranges.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Nominal Value</th>
<th>Units</th>
<th>Expected Range</th>
<th>Precision</th>
</tr>
</thead>
<tbody>
<tr>
<td>P_user_chemtype</td>
<td>1</td>
<td>Unitless</td>
<td>1,2,3</td>
<td>1</td>
</tr>
<tr>
<td>P_TREAT_UNITS</td>
<td>“lbs/lane mile”</td>
<td>Character</td>
<td>30 characters</td>
<td>--</td>
</tr>
<tr>
<td>P_MIN_TREAT</td>
<td>100</td>
<td>lbs/lanemile</td>
<td>0-9999</td>
<td>1</td>
</tr>
<tr>
<td>P_MAX_TREAT</td>
<td>2000</td>
<td>lbs/lanemile</td>
<td>0-9999</td>
<td>1</td>
</tr>
<tr>
<td>P_ROUND_TREAT</td>
<td>25</td>
<td>lbs/lanemile</td>
<td>0-500</td>
<td>1</td>
</tr>
<tr>
<td>P_InitialChemAppLossRateDry</td>
<td>0.20</td>
<td>Fraction</td>
<td>0.0-1.0</td>
<td>0.01</td>
</tr>
<tr>
<td>P_InitialChemAppLossRatePrewet</td>
<td>0.10</td>
<td>Fraction</td>
<td>0.0-1.0</td>
<td>0.01</td>
</tr>
<tr>
<td>P_InitialChemAppLossRateLiquid</td>
<td>0.05</td>
<td>Fraction</td>
<td>0.0-1.0</td>
<td>0.01</td>
</tr>
<tr>
<td>P_RoadwayImmediateRunoff</td>
<td>0.98</td>
<td>Fraction</td>
<td>0.0-1.0</td>
<td>0.01</td>
</tr>
<tr>
<td>P_ChemSolutionRunoff</td>
<td>0.15</td>
<td>Fraction</td>
<td>0.0-1.0</td>
<td>0.01</td>
</tr>
<tr>
<td>P_RoadwayMinDepthforRunoff</td>
<td>0.05</td>
<td>Fraction</td>
<td>0.0-1.0</td>
<td>0.01</td>
</tr>
<tr>
<td>P_DryChem_LossRate</td>
<td>0.25</td>
<td>Fraction</td>
<td>0.0-1.0</td>
<td>0.01</td>
</tr>
<tr>
<td>P_NominalTrafficDilution</td>
<td>0.01</td>
<td>Fraction</td>
<td>0.0-1.0</td>
<td>0.01</td>
</tr>
<tr>
<td>P_MIN_TEMP</td>
<td>-10.0</td>
<td>Degrees C</td>
<td>-100 to 0</td>
<td>0.1</td>
</tr>
<tr>
<td>P_MAX_TEMP</td>
<td>2.0</td>
<td>Degrees C</td>
<td>-10 to 10</td>
<td>0.1</td>
</tr>
<tr>
<td>P_CHEM_SAFE_DELTA</td>
<td>2.0</td>
<td>Degrees C</td>
<td>-5 to 5</td>
<td>0.01</td>
</tr>
</tbody>
</table>

B.5.e. Algorithm Constants

C_BAD_TEMP, C_BAD_CONC: Bad value flags for concentrations, temperatures and other values. [unitless][-999].
C_mmHr_to_LbSf: Conversion factor for rain from mm/hour to lbs/square-foot. \([(\text{mm/hr})/(\text{lb/sf})]\times0.2047].

C NOMINAL_LANEWIDTH: The width of a single lane of roadway. [feet][12].

C MILES_TO_FEET: The number of feet in a mile. [feet][5280].

C CHEMTYPE_NACL: Internal identifier for salt. [unitless][1].
C CHEMTYPE_CACL2: Internal identifier for calcium chloride. [unitless][2].
C CHEMTYPE_MGCL2: Internal identifier for magnesium chloride. [unitless][3].
C CHEMTYPE_CMA: Internal identifier for calcium magnesium acetate. [unitless][4].
C CHEMTYPE_KAC: Internal identifier for potassium acetate. [unitless][5].
C CHEMTYPE_CALIBER: Internal identifier for Caliber (brand name). [unitless][6].
C CHEMTYPE_ICESLICER: Internal identifier for IceSlicer (brand name). [unitless][7].

C_NACL_EUTECTIC_TEMP: The eutectic temperature for salt. [degC][-21.6].
C_CACL2_EUTECTIC_TEMP: The eutectic temperature for calcium chloride. [degC][-51].
C_MGCL2_EUTECTIC_TEMP: The eutectic temperature for magnesium chloride. [degC][-33].
C_CMA_EUTECTIC_TEMP: The eutectic temperature for calcium magnesium acetate. [degC][-27.5].
C_KAC_EUTECTIC_TEMP: The eutectic temperature for potassium acetate. [degC][-60].
C_CALIBER_EUTECTIC_TEMP: The eutectic temperature for Caliber (brand name). [degC][-60].
C_ICECLICER_EUTECTIC_TEMP: The eutectic temperature for Ice Slicer (brand name). [degC][-60].

C_NACL_EUTECTIC_CONC: The eutectic chemical concentration for salt. [%][23.3].
C_CACL2_EUTECTIC_CONC: The eutectic chemical concentration for calcium chloride. [%][29.8].
C_MGCL2_EUTECTIC_CONC: The eutectic chemical concentration for magnesium chloride. [%][21.6].
C_CMA_EUTECTIC_CONC: The eutectic chemical concentration for calcium magnesium acetate. [%][32.5].
C_KAC_EUTECTIC_CONC: The eutectic chemical concentration for potassium acetate. [%][49.0].
C_CALIBER_EUTECTIC_CONC: The eutectic chemical concentration for Caliber. [%][32.0].
C_ICESLICER_EUTECTIC_CONC: The eutectic chemical concentration for Ice Slicer. [%][32.0].

C_NACL_HALFPOINT_TEMP: The temperature on the solution point curve that corresponds to 50% of the eutectic concentration. Used as a measure of the minimum effective temperature of NaCl. [degC][-10.0].

C_CACL2_HALFPOINT_TEMP: The temperature on the solution point curve that corresponds to 50% of the eutectic concentration. Used as a measure of the minimum effective temperature of CaCl2. [degC][-12.0].

C_MGCL2_HALFPOINT_TEMP: The temperature on the solution point curve that corresponds to 50% of the eutectic concentration. Used as a measure of the minimum effective temperature of MgCl2. [degC][-9.5].

C_CMA_HALFPOINT_TEMP: The temperature on the solution point curve that corresponds to 50% of the eutectic concentration. Used as a measure of the minimum effective temperature of CMA. [degC][-7.5].

C_KAC_HALFPOINT_TEMP: The temperature on the solution point curve that corresponds to 50% of the eutectic concentration. Used as a measure of the minimum effective temperature of KAC. [degC][-16.0].

C_CALIBER_HALFPOINT_TEMP: The temperature on the solution point curve that corresponds to 50% of the eutectic concentration. Used as a measure of the minimum effective temperature of Caliber. [degC][-30.0].

C_ICESLICER_HALFPOINT_TEMP: The temperature on the solution point curve that corresponds to 50% of the eutectic concentration. Used as a measure of the minimum effective temperature of Ice Slicer. [degC][-10.0].

C_MIN_ACCEPTABLE_CONC: Used as a minimal chemical concentration value for a chemical to be effective when the temperature is at or above 0 deg C (used in CalcCriticalChemSolutionPoint).

C_CHEM_CHK: Threshold used in RulesOfPractice.cc in situations where plow treatments may be recommended. (P_TREAT_UNITS).

C_NOCHEM_THRESH: Threshold for zeroing out chemical concentration. (lbs/sf).

C_AVAILH20_THRESH: Threshold for zeroing out available roadway water. (lbs/sf).

C_MAX_CHEMNAME: Maximum length of the character string containing the name of the chemical being used (e.g. “Magnesium Chloride”).

C_MAX_TREATNAME: Maximum length of the character string containing the chemical rate units (e.g. “Lbs/lane-mile”).
C_STATUS_OK: Return code that the algorithm is working properly.
C_STATUS_BAD: Return code that the algorithm is not working properly.

References


Appendix C: 
Net Mobility Module

The DOT users indicated a desire to have a single (non-dimensional) metric to identify the predicted state of the roadway relative to winter road conditions. For demonstration purposes, a simple mobility metric has been developed that takes into account pavement condition (wet, dry, snow, snow depth, ice, etc.).

The Net Mobility Module reads in the meteorological and road surface conditions and outputs an index describing the amount of mobility a vehicle could encounter on the road. This index ranges from 0 (no mobility) to 1 (optimal road conditions). A number of tables exist which describe the mobility in certain conditions. A decision tree leads to finding the proper table. The mobility index is found by finding the proper cell in the table that fits the existing environmental and traffic conditions.

Development Status: This metric needs additional development, as it does not currently take into account some of the subtle factors (e.g., wet snow, dry snow, snow on ice, etc.) that impact mobility. A simple algorithm has been implemented as a placeholder. The algorithm uses the following general scheme:

<table>
<thead>
<tr>
<th>Pavement Condition</th>
<th>Mobility Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dry</td>
<td>1.0</td>
</tr>
<tr>
<td>Wet</td>
<td>0.7</td>
</tr>
<tr>
<td>Snow &lt; 4 inches</td>
<td>0.6</td>
</tr>
<tr>
<td>Snow 4-6 inches</td>
<td>0.4</td>
</tr>
<tr>
<td>Snow &gt; 6 inches</td>
<td>0.3</td>
</tr>
<tr>
<td>Ice</td>
<td>0.2</td>
</tr>
</tbody>
</table>

The utility of using a mobility metric must still be fully investigated. It is anticipated that several iterations with the users and experience will be required to firm up this concept.

The net mobility code is part of the Road Condition and Treatment Module. The code can be found on the MDSS CD in the directory /src/apps/roadcond_mdss. The file of interest is calc_mobility.cc.
Appendix D:  
Rules of Practice Module 

The “rules of practice” code is part of the Road Condition and Treatment Module. The code can be found on the MDSS CD in the directory /src/apps/roadcond_mdss. The files of interest are RulesOfPractice.cc and RulesOfPractice.hh.

D.1. Algorithm Identifier

RulesOfPractice

D.2. Type

Process

D.3. Software

RulesOfPractice.cc
RulesOfPractice.hh

D.4. Conceptual Overview

The RulesOfPractice algorithm is designed to recommend appropriate road treatment actions for winter storm maintenance crews during winter storms. The initial guidance for anti-icing operations were detailed in the FHWA “Manual of Practice for an Effective Anti-icing program: A guide for Highway Maintenance Personnel”. The series of FHWA look-up tables (example shown in Figure D1), however, were based on the eutectic curves of anti-icing chemicals (for further discussion on eutectic curves see the ChemConc algorithm description Appendix B). Earlier versions of the RulesOfPractice used simplified curves to essentially automate the treatment look-up from the FHWA tables. This latest version for Release 5, however, utilizes both the eutectic curves directly and the dilution algorithm from the ChemConc algorithm to more accurately estimate the amount of chemicals needed on the road surface to keep snow and ice from bonding.
Maintenance personnel typically gauge the amount of chemical needed during a storm based on past experience and forecasted road and weather conditions. Anti-icing refers to a snow and ice control practice of applying chemicals to prevent the formation or bonding of snow and ice to road surfaces. Treatments are often applied in advance of the actual storm event so that the initial snow/ice does not form a pack on the road surface. The RulesOfPractice algorithm embraces the concept of anti-icing, but also recognizes that some storm conditions (overwhelming snow) or circumstances (equipment breakdown, inadequate crew availability) may necessitate de-icing (the practice of combating the storm as the storm happens -- plowing and applying chemicals as possible to minimize snow and ice build-up). This algorithm is only a first step towards a fully automated guidance system; many simplifications have been made to make this initial task more manageable.

As shown in the component algorithms diagram for the RCTM system (Figure B4), the RulesOfPractice module (labeled “Determine Recommended Treatment”) ingests the ambient weather forecast, the forecasted road conditions and the storm characterization (detailed in Appendix E). The algorithm iterates through the forecast period and identifies the point at which the road surface will become unsafe (called the trigger point). The rules (detailed below) are used to determine the number and type of treatments needed to protect the road surface both during the storm and until the road surface is considered “dry”. Often, treatments will be recommended before the trigger point to protect the road from freezing as the precipitation starts and/or after the precipitation ends to protect the road surface from re-freezing. The algorithm outputs a structure containing the treatments that the system has determined are needed to protect the road surface from re-freezing. Users may select these recommended treatments, or input their own user-selected treatments. Once a user selects a treatment plan, the RulesOfPractice will only recommend additional treatments if the
chosen treatments do not appear (to the automated system) to be sufficient to protect the road surface.

D.5. Algorithm Description

The *RulesOfPractice* algorithm is one of several key components within the RCTM. The RCTM delivers forecasts of the necessary weather variables (precipitation rate and type), road condition variables (pavement temperature and snow depth from the pavement temperature model), and storm characterization. The user selects the type of chemical to use to treat the road (*UserChemType*). The various inputs are passed into the *RulesOfPractice* algorithm, which in turn looks at the road condition data to determine if treatment may be needed (a build-up of snow or freezing rain on the road for example).

There are a number of user parameters that may be used to adjust the sensitivity and thresholds of the algorithm for specific operations. These parameters are described in detail in section **D.5.c** and are referred to as necessary in the following discussion.

The overall functional flow of the algorithm is shown in Figure D2.

![Figure D2. Functional flow diagram for Rules of Practice algorithm.](image)

The algorithm is currently triggered by the RCTM when a user requests treatment options for a specific route, chemical type (*UserChemType*) and form (*UserChemForm*). The necessary weather variables, road conditions and a storm characterization are then sent to the *RulesOfPractice*. Currently, five chemicals are supported as shown in Table D1. Each chemical can be specified as Dry, Pre-wet, or Liquid forms.
Next, the algorithm loops over all the forecast hours looking for a treatment trigger. The first trigger looks to see if the water or chemical solution that is on the road surface, or forecasted to be on the road surface is expected to freeze over the expected route travel time. Freezing rain (at any level) is also a trigger point. A routine called DeterminePrecipType is used to determine the current precipitation type (This process is run as part of the CharacterizeStorm module). The weather forecast may indicate rain or freezing rain, but the road surface must also be cold enough to support a freezing rain declaration (the parameters P_FRZRAIN_FRZRAIN_TROAD and P_FRZRAIN_RAIN_TROAD control the threshold points for freezing rain). In addition, if snow is building up on the road, but no chemical action is appropriate (roads temperatures are too cold for example), then the system will trigger a plow-only treatment once the plow snow-depth threshold is exceeded (P_SNOW_PLOW_THRESH). Once a trigger is found, the algorithm attempts to determine the next appropriate treatment. When this first treatment is determined the algorithm applies this single treatment and invokes the ChemConc and Pavement Temperature algorithms to update the road conditions. The algorithm then iterates to find the next treatment (if necessary) until the entire storm has been treated.

The first step in assessing treatment options is to determine if the storm conditions are such that only plowing should be performed. There are several storm criteria that are used to determine if plow-only treatments should be applied, including:

- In-storm road temperatures are Cold or In-range (for chemicals) and becoming too Cold for chemicals.
- Road temperatures are In-range or Cold and becoming In-range and significant amounts of blowing snow are forecasted.
- Post-storm road temperatures turn too Cold for chemicals.

Next, if anti-icing chemicals are warranted, the algorithm determines if a pre-treatment strategy should be undertaken. Pre-treatments are performed if the following criteria are met:

- Storm phase types that start as freezing rain.
- Storm starts as snow and:
  - Road temperatures are NOT warm prior to the start of storm, or at the beginning of the storm (road trend type W or WI).
  - Road temperatures are expected to be In-range (I or IW).
  - Road temperatures are cold (C, IC, CI) and significant amounts of blowing snow are NOT forecasted.
- Storm phase type does NOT begin as Rain.
Finally, if pre-treatment isn’t needed (or has already been applied in a past treatment), the RulesOfPractice utilizes the routine CalcChemicalRate() to directly calculate the amount of anti-icing chemicals needed to protect the road surface over the route time (which includes the time to travel the route and re-supply). In short storms (or near the end of storms), if possible, the chemical rate is increased to cover the remaining storm length.

Once a treatment type and level has been determined, the algorithm must determine the appropriate time to apply the treatment.

For plow-only operations, plowing is recommended at the point where plowable snow is on the road (P_SNOW_PLOW_THRESH). If the threshold is never met, no treatment will be recommended for this route. There is also a user adjustable parameter, P_PLOW_OFFSET, that allows the user to subtract an offset from this hour.

Pre-treatment operations are given a specific time in the MDSS, although this time generally refers to starting the operation on or before the recommended time. For pre-treatments the estimated treatment is changed to a brine recommendation (P_LIQUID_BRINE_EQUIV lbs/lane-mile) and the recommended treatment time is set to the trigger time minus P_PRETREAT_OFFSET (hours).

For normal anti-icing operations, the application rate is first checked to ensure that it is within limits. If the treatment rate is below the user’s specified minimum (MinApplicationRate), then the MinApplicationRate is recommended instead. Conversely, if the treatment rate is above the user’s specified maximum (MaxApplicationRate) then the recommended rate is capped at MaxApplicationRate. In addition, the recommended start of treatment time is offset backwards from the trigger point by the route trip time (RouteRoundtripTiming). If, however, rain is falling prior to the treatment trigger time, then the offset is set by the parameter PreRainDelta. This offset allows the application to be applied to most of the route before conditions deteriorate.

In addition, there is a special case where the system starts up with snow already on the road. This can happen when users have not treated the road, or have, but have not entered the treatment into the system or when the storm is so strong that no level of acceptable treatments will keep up with the snow. In this case, the existing snow is converted to available water (essentially the liquid water equivalent with adjustments for runoff). This water is mixed with any available chemical and then treated as any other available water as above (plow-only or chemical application).

D.5.a Algorithm Inputs

The RulesOfPractice algorithm ingests forecasts of precipitation rate and type, and pavement temperature that are used to determine the correct level of chemicals needed to keep the roads ice free (CalcChemicalRate). Snow depth is used in both the chemical treatment investigation and plowing recommendations. The user must supply the type and form (liquid, pre-wet or dry) of the chemical they will be using so that the algorithm will
only recommend chemical treatments in the appropriate temperature ranges and with reasonable application rates.

**D.5.b. Algorithm Outputs**

The **RulesOfPractice** algorithm outputs a structure detailing the recommended treatment plan. The structure indicates whether chemical treatment and/or plowing are necessary and at what hour. If chemicals are recommended, the algorithm also returns the chemical type and application rate.

**D.5.c. Algorithm Structures**

There are four structures that are used within the Rules of Practice algorithm: ChemicalType, ROPParamsStruct, TreatmentCurves, and Treatment structures. The ChemicalType structure is used to define the characteristics of the user’s preferred anti-icing chemical. This structure is detailed in Appendix B.

The ROPParamsStruct structure contains parameters that help to define the trigger points and route timing for Rules of Practice decisions. The structure contains:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Bare_Pavement_Thresh:</strong></td>
<td>The chemical treatment threshold trigger for snow on the road. When a snow depth above the minimum is found, chemical treatment options are investigated.</td>
</tr>
<tr>
<td><strong>Snow_Plow_Thresh:</strong></td>
<td>The snow plow treatment threshold for snow on the road. Plowing is recommended when snow depths meet or exceed this threshold.</td>
</tr>
<tr>
<td><strong>RouteRoundTripTiming:</strong></td>
<td>Estimated time it takes to traverse the specified service route. This time delta is used both to determine the length of time to investigate treatments and to estimate the proper pre-treatment time.</td>
</tr>
<tr>
<td><strong>TruckTurnAroundTime:</strong></td>
<td>Estimated time (in minutes) it takes for a truck to load or reload the anti-icing chemicals.</td>
</tr>
<tr>
<td><strong>PreTreatOffset:</strong></td>
<td>The time in minutes to offset pretreatment operations from the trigger point.</td>
</tr>
<tr>
<td><strong>PreRainDelta:</strong></td>
<td>The time in minutes to offset chemical treatment if rain is falling prior to the trigger time.</td>
</tr>
<tr>
<td><strong>ChemRateSensitivity:</strong></td>
<td>A sensitivity value used to buffer the minimum allowable chemical concentration (% points).</td>
</tr>
<tr>
<td><strong>TreatmentStrategy:</strong></td>
<td>The treatment strategy to be used in determining how often to check for a new treatment. A &quot;C_ONTRIGGER&quot; strategy will search for the next treatment and if chemicals are used it will not look for another treatment until the chemicals fail -- this would be an Iowa strategy (and to some extent CDOT in their less traffic impacted routes). The &quot;C_CONTINUOUS&quot; strategy determines the next treatment and assumes that the chemicals will fail after the treatment is applied. So, if we have a route that takes 2 hours to treat and an application of chemicals is put down at 1200, then</td>
</tr>
</tbody>
</table>
the next treatment will be triggered at 1500 (1200+2hour treatment + next hour). It still needs to find a reason for a new application, but often it will in heavy storms. This strategy is similar to the E-470 strategy.

Missing TreatmentCurves

Finally, the Treatment structure holds all the information about a recommended treatment. The members are:

StartTime: The recommended start time of the plowing or application (in hours from the start of the forecast).

EndTime: The expected end time of the treatment given the route time and truck turn around estimate (in hours from the start of the forecast).

EffectiveTime: The last time period covered by this treatment (in hours). This would be the same as EndTime for a plowing operation. In the case of chemical application, EffectiveTime represents the last time period that the chemicals will be effective as a deicing agent.

DoPlowing: Flag indicating whether the roads should be plowed (1) or not (0).

ApplyChemicals: Flag indicating whether chemicals should be applied (1) or not (0).

Chemicals: The type of chemical to apply (ChemicalType).

ApplicationRate: The rate (in P_TREAT_UNITS) at which to spread the recommended chemical.

TreatType: Holds the type of treatment based on the treatment flags (C_NO_TREAT, C_PRETREAT, C_TREAT, or C_PLOWONLY).

Explanation: A text string describing the type of treatment being employed and why (or in the case of no treatment, why not).

D.5.d. Algorithm Parameters

P_USER_CHEMTYPE: Specify the type of anti-icing (de-icing) chemical to calculate concentrations. There are currently five types supported: 1 – NaCl (Salt), 2- CaCl₂ (calcium chloride) and 3 – MgCl₂ (magnesium chloride), 6 – Caliber, 7 – Ice Slicer. Other chemicals are not yet fully functional.

P_USER_CHEMFORM: The form of the chemical to be used (C_DRY, C_PREWET, C_LIQUID).

P_TREAT_UNITS: The name of the units being used within the algorithm (e.g. “Lbs/lane-mile”, “Gals/lane-mile”).

P_MIN_TREAT: The minimum rate at which the chemical should be applied (in P_TREAT_UNITS).
P_MAX_TREAT: The maximum rate at which the chemical should be applied (in P_TREAT_UNITS).

P_ROUND_TREAT: When recommending treatment rates, round to this nearest interval (in P_TREAT_UNITS).

P_TURNAROUND_TIME: The time it takes to reload a truck (minutes).

P_MIN_TEMP: The minimum temperature at which this chemical should be applied (nominally est to the ½ point temperature of the chemical C_NACL_HALFPOINT_TEMP for example).

P_MAX_TEMP: The maximum temperature at which this chemical should be applied (deg C).

P_CHEM_SAFE_DELTA: A sensitivity adjustment (deg C) for the eutectic curve.

P_CHEMRATE_SENSITIVITY_PCT: A sensitivity value based on a percentage point offset from the eutectic curve. (this replaces the CHEMRATE_SENSITIVITY parameter available in earlier versions of the software).

P_BARE_PAVEMENT_THRESH: The chemical treatment threshold trigger for snow on the road. When a snow depth above the minimum is found, chemical treatment options are investigated (inches).

P_SNOW_PLOW_THRESH: The snow plow treatment threshold for snow on the road. Plowing is recommended when snow depths meet or exceed this threshold (inches).

P_PLOW_OFFSET: The time (in hours) to offset a plow-only treatment.

P_FRZRAIN_FRZRAIN_TROAD: The pavement temperature necessary to declare freezing rain on the road surface if the MDSS forecast indicating a precipitation type of freezing rain (deg F).

P_FRZRAIN_RAIN_TROAD: The pavement temperature necessary to declare freezing rain on the road surface if the MDSS forecast is indicating a precipitation type of rain (deg F).

P_LIQUID_BRINE_EQUIV: Current system only allows one type of chemical application (lbs/lane-mile). In order to represent the a typical 50 gal/lane-mile liquid brine treatment we must use a dry salt lbs/lane-mile equivalent of 110 lbs/lane-mile.

P_LIQUID_BRINE_OFFSET: The time offset for pre-treatments is generally different than that of regular applications (it is generally applied much earlier) (hours). This is also used to determine the minimum number of hours of “dry” roads needed to declare a pre-treatment.

P_PRERAIN_DELTA: The time (in minutes) to offset a treatment when rain precedes the triggering event.
P_BLOWINGSNOWCHEMADJUSTMENT:
The nominal fractional increase in chemical rate when blowing snow is present (multiplied by the square of the blowing snow potential value).

Table D2. Nominal algorithm parameter values and ranges.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Nominal Value</th>
<th>Units</th>
<th>Expected Range</th>
<th>Precision</th>
</tr>
</thead>
<tbody>
<tr>
<td>P_USER_CHEM_TYPE</td>
<td>1</td>
<td>Unitless</td>
<td>1,2,3</td>
<td>1</td>
</tr>
<tr>
<td>P_USER_CHEM_FORM</td>
<td>C_PREWET</td>
<td>Index</td>
<td>C_DRY, C_PREWET,</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>C_LIQUID</td>
<td></td>
</tr>
<tr>
<td>P_TREAT_UNITS</td>
<td>“lbs/lanemile”</td>
<td>Character</td>
<td>30 characters</td>
<td>--</td>
</tr>
<tr>
<td>P_MIN_TREAT</td>
<td>100</td>
<td>lbs/lanemile</td>
<td>0-9999</td>
<td>1</td>
</tr>
<tr>
<td>P_MAX_TREAT</td>
<td>2000</td>
<td>lbs/lanemile</td>
<td>0-9999</td>
<td>1</td>
</tr>
<tr>
<td>P_ROUND_TREAT</td>
<td>25</td>
<td>lbs/lanemile</td>
<td>0-500</td>
<td>1</td>
</tr>
<tr>
<td>P_TURNAROUND_TIME</td>
<td>30</td>
<td>Minutes</td>
<td>0-60</td>
<td>1</td>
</tr>
<tr>
<td>P_MIN_TEMP</td>
<td>-10.0</td>
<td>Degrees C</td>
<td>-100 to 0</td>
<td>0.1</td>
</tr>
<tr>
<td>P_MAX_TEMP</td>
<td>2.0</td>
<td>Degrees C</td>
<td>-10 to 10</td>
<td>0.1</td>
</tr>
<tr>
<td>P_CHEM_SAFE_DELTA</td>
<td>2.0</td>
<td>Degrees C</td>
<td>-5 to 5</td>
<td>0.01</td>
</tr>
<tr>
<td>P_CHEM_RATE_SENSITIVITY_PCT</td>
<td>2.0</td>
<td>% points</td>
<td>-5 to 5</td>
<td>0.1</td>
</tr>
<tr>
<td>P_BARE_PAVEMENT_SENSITIVITY_THRESH</td>
<td>0.005</td>
<td>Inches</td>
<td>0.001-0.5</td>
<td>0.001</td>
</tr>
<tr>
<td>P_SNOW_PLOW_THRESH</td>
<td>3.0</td>
<td>Inches</td>
<td>0.05-10.0</td>
<td>0.01</td>
</tr>
<tr>
<td>P_FRZRAIN_FRZRAIN_TROAD</td>
<td>35.0</td>
<td>Degrees F</td>
<td>25-40</td>
<td>0.1</td>
</tr>
<tr>
<td>P_FRZRAIN_RAIN_TROAD</td>
<td>35.0</td>
<td>Degrees F</td>
<td>25-40</td>
<td>0.1</td>
</tr>
<tr>
<td>P_LIQUID_BRINE_EQUIV</td>
<td>110.0</td>
<td>lbs/lanemile</td>
<td>110</td>
<td>1</td>
</tr>
<tr>
<td>P_LIQUID_BRINE_OFFSET(old)</td>
<td>6</td>
<td>Hours</td>
<td>0-24</td>
<td>1</td>
</tr>
<tr>
<td>P_PRETREAT_OFFSET(new)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

D.5.e. Algorithm Constants

Following are general constants:

C_BAD_VAL: Bad value flags for concentrations, temperatures and other values. [unitless][-999].

C_InHr_to_LbSf: Conversion factor for rain from inches/hour to lbs/square-foot. [(in/hr)/(lb/sf)] [5.2].

C_NOMINAL_LANEWIDTH: The width of a single lane of roadway. [feet][12].

C_MILES_TO_FEET: The number of feet in a mile. [feet][5280].
The following are used to determine user preferred treatment strategy:

C_ONTRIGGER: Recommend a new treatment only when the algorithm finds a new treatment trigger (icy or snow covered roads).
C_CONTINUOUS: Begin looking for new treatments as soon as the last treatment was completed.

The following constants are used to index the type of chemical:
C_CHEMTYPE_NACL: Internal identifier for salt. [unitless][1].
C_CHEMTYPE_CACL2: Internal identifier for calcium chloride. [unitless][2].
C_CHEMTYPE_MGCL2: Internal identifier for magnesium chloride. [unitless][3].
C_CHEMTYPE_CMA: Internal identifier for salt. [unitless][1].
C_CHEMTYPE_KAC: Internal identifier for calcium chloride. [unitless][2].
C_CHEMTYPE_CALIBER: Internal identifier for magnesium chloride. [unitless][3].
C_CHEMTYPE_ICESLICER: Internal identifier for salt. [unitless][1].

The following constants are used to identify the chemical form:
C_CHEMFORM_DRY: Dry chemical. [unitless][0].
C_CHEMFORM_PREWET: Pre-wet (with any liquid). [unitless][1].
C_CHEMFORM_LIQUID: Liquid [unitless][2].

The following constants are used to index the type of treatment recommendations:
C_PRETREAT: Apply a pre-treatment application (4).
C_TREAT: Apply chemicals (2).
C_TREAT_PLOWONLY: Plow only (1).
C_TREAT_THRESH: Threshold between treatment and no treatment (used internally) (0).
C_NOTREAT_NONEED: No chemical treatment needed because there is no snow on road (-1).
C_NOTREAT_TOOCOLD: No chemical treatment needed because it is too cold (-2).
C_NOTREAT_TOOWARM: No chemical treatment needed because it is too warm (-3).
C_NOTREAT_1515_RULE: No chemical treatment needed because combinations of winds and road temperature will make chemical application hazardous due to blowing snow (-4).
C_NOTREAT_UNDEF: Error in treatment recommendation (-9).

The following constants are used to index the type of precipitation:
C_PTYPE_NONE: No precipitation (0).
C_PTYPE_RAIN: Rain (1).
C_PTYPE_SNOW: Snow (2).
C_PTYPE_MIXED: Snow/rain mix (3).
C_PTYPE_SLEET: Sleet (4).
C_PTYPE_FRZRRAIN: Freezing rain (5).

The following constants are used to index the road water phase:
C_PTYPE_DRY: The road is dry. (0).
C_PTYPE_WET: The road is wet. (1).
C_PTYPE_CHEMWET: The water is in a chemical solution that is keeping it from
freezing. (2).

C_PTYPE_CHEMICE: The water is in a chemical solution but it is too weak to keep from freezing (3).

C_PTYPE_SLUSH: Slush on road (4).

C_PTYPE_SNOW: Snow on road (5).

C_RTYPE_ICE: Water is frozen (6).

References


Appendix E:
Characterize Storm Module

The storm characterization algorithm is part of the Road Condition and Treatment Module. The code can be found on the MDSS CD in the directory /src/apps/roadcond_mdss. The files of interest are CharacterizeStorm.cc and RulesOfPractice.hh.

E.1. Algorithm Identifier

CharacterizeStorm

E.2. Type

Process

E.3. Software

CharacterizeStorm.cc

RulesOfPractice.hh

E.4. Conceptual Overview

Understanding the overall structure, extent and type of winter storm is an important factor in determining the type and extent (if any) of the treatment required. For example, knowing that a storm will start as rain indicates that pre-treatment should be suppressed. Knowing that post-storm temperatures will be below the range chemical effectiveness may indicate that only plowing should be performed. In support of this goal, the CharacterizeStorm algorithm is designed to summarize the characteristics of the overall storm (prior, during, and after precipitation) that are relevant to the type and extent of treatment that may be required. Figure B4 shows the overall structure of the RCTM. The CharacterizeStorm algorithm is invoked after the initialization of the road conditions based on the current weather forecast, previous road conditions, and operational characteristics of the route. The routine outputs a structure containing information on the pre-, in-, and post-storm environment. Figure E1 illustrates the breakdown of storm characteristics gathered for the pre-, in-, and post-storm environment.
Figure E1. Illustration of components captured for storm characterization.

The road temperature trends for assessing chemical effectiveness are represented by “W” for too warm, “I” for In-range, and “C” for too cold. Precipitation types are represented by “R” for rain, “Z” for ice, and “S” for snow. Combinations of types are also shown.

Starting from the right, the pre-storm environment is captured by measuring the road temperature trends as warm, in-range, or cold. A simple fuzzy logic template is applied to the time period 6 hours prior to the start of measurable precipitation (above 0.01 inches liquid water equivalent) on the road surface. The template applied is shown in Figure E2 (a) warm, (b) in-range, and (c) cold. In-range measurements represent road temperatures that are within the effective range of the anti-icing chemical to be used (refer to the ChemConc algorithm description in Appendix B). The road temperature characterization of warm or cold environments indicates time segments where the road temperatures are above or below the effective chemical range, respectively. The fuzzy scoring function is used to reduce the impact of borderline or spurious measurements on the system. Similarly, the road temperature trend is captured for the post-storm environment shown on the left-hand side of Figure E1.
There are several in-storm environment characteristics that are captured by the algorithm. Overall precipitation characteristics, such as total amount and timing (stop, start, and duration) of both liquid and frozen precipitation are captured first. Road temperature trends are also captured, but up to two phases of temperatures are allowed. For example, roads may start out as warm and transition to in-range, or start in-range and turn to cold. There are 7 combinations of road temperatures: W, I, C, WI, IW, IC, CI. Transitions that skip a range are unlikely and are not captured by the system (e.g. WC). Finally, the phase type of the storm is captured. Like the road temperature trends a scoring function is applied to determine the type of storm. Since many storms have multiple phases, up to 3 phase changes are allowed per storm. Therefore the system should be able to identify a storm that starts as rain, turns briefly to freezing rain and then to all snow (RZS). In the case of phase change, skipping a phase is allowed so that a storm may be classified as rain changing to snow (RS). There are 23 classifications of in-storm precipitation phase, as shown in Table E1.

Table E1. Storm phase type identifier and description.

<table>
<thead>
<tr>
<th>Storm phase type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>All rain.</td>
</tr>
<tr>
<td>Z</td>
<td>All freezing rain.</td>
</tr>
<tr>
<td>S</td>
<td>All snow.</td>
</tr>
<tr>
<td>RS</td>
<td>Rain changing to snow.</td>
</tr>
<tr>
<td>RZ</td>
<td>Rain changing to freezing rain.</td>
</tr>
<tr>
<td>SZ</td>
<td>Snow changing to freezing rain.</td>
</tr>
<tr>
<td>SR</td>
<td>Snow changing to rain.</td>
</tr>
<tr>
<td>ZR</td>
<td>Freezing rain changing to rain.</td>
</tr>
<tr>
<td>ZS</td>
<td>Freezing rain changing to snow.</td>
</tr>
<tr>
<td>RRZ</td>
<td>Rain ending as freezing rain.</td>
</tr>
<tr>
<td>RZZ</td>
<td>Starting as rain turning to freezing rain.</td>
</tr>
<tr>
<td>RZR</td>
<td>Rain with a brief period of freezing rain.</td>
</tr>
<tr>
<td>------</td>
<td>------------------------------------------</td>
</tr>
<tr>
<td>RZS</td>
<td>Starting as rain, turning to freezing rain and ending as snow.</td>
</tr>
<tr>
<td>ZZR</td>
<td>Freezing rain ending as rain.</td>
</tr>
<tr>
<td>ZZS</td>
<td>Freezing rain ending as snow.</td>
</tr>
<tr>
<td>ZRR</td>
<td>Starting as freezing rain and turning to rain.</td>
</tr>
<tr>
<td>ZRZ</td>
<td>Freezing rain with a brief period of rain.</td>
</tr>
<tr>
<td>ZSS</td>
<td>Starting as freezing rain and turning to snow.</td>
</tr>
<tr>
<td>ZSZ</td>
<td>Freezing rain with a brief period of snow.</td>
</tr>
<tr>
<td>SSZ</td>
<td>Snow ending as freezing rain.</td>
</tr>
<tr>
<td>SZZ</td>
<td>Starting as snow and turning to freezing rain.</td>
</tr>
<tr>
<td>SZS</td>
<td>Snow with a brief period of freezing rain.</td>
</tr>
<tr>
<td>SZR</td>
<td>Starting as snow, turning to freezing rain and ending as rain.</td>
</tr>
</tbody>
</table>

One final derived variable is a measurement of the likelihood that road temperatures will be cold and blowing snow will be present. This measure of blowing is not the same as that captured in the new MDSS blowing snow algorithm, but it will be in the future. The measure is designed to flag storms where treatments should be suppressed in lieu of plow-only operations because the snow will simply blow across the road without sticking to the surface. A redesign of this parameter is likely so a detailed description is not given here.

**E.5. Algorithm Description**

The *StormCharacterization* algorithm, as described above, a key component of the RCTM, is designed to capture the overall storm characteristics (prior, during and after precipitation) that are relevant to the type and extent of treatment that may be required.

There are a number of user parameters that may be used to adjust the sensitivity and thresholds of the algorithm for specific operations. These parameters are described in detail in section E.5.c and are referred to as necessary in the following discussion.

The overall functional flow of the algorithm is shown in Figure E3. The algorithm is currently triggered by the RCTM whenever a new set of forecasted weather conditions is received. The algorithm ingests current weather forecast, previous road conditions and operational characteristics of the route, and outputs a structure containing information on the pre-, in-, and post-storm environment.
E.5.a Algorithm Inputs

The CharacterizeStorm algorithm ingests forecasts of precipitation rate and type, wind speed and pavement temperature. Forecasted precipitation type and pavement temperature are used to determine the expected phase of any moisture on the road and to capture the hourly extent of the storm event. Pavement temperature alone is used to determine the road temperature trends within the storm, likewise precipitation type and storm type (rain, snow, rain-snow, etc). The blowing snow potential index is also input.

E.5.b. Algorithm Outputs

The CharacterizeStorm algorithm outputs a structure containing variables that describe the expected type of storm. Pre-, In-, and Post-storm codes for road temperature trends and In-storm characterization of the storm precipitation phase changes. In addition, summary statistics are also saved on storm start and stop and total liquid and frozen precipitation. The structure is detailed in section E5.c.

E.5.c. Algorithm Structures

There is one primary structure used within the CharacterizeStorm algorithm: StormType. The StormType structure holds all the storm summary information gathered in CharacterizeStorm, as follows:

BeginLookout: First hour to look for trigger points and pretreat options based on storm characteristics and current road water/snow conditions.
EndLookout: The last hour that protection is needed (dry road time).
StartWetRoad: First hour that the road has water on it above threshold (C_AVAILH20_THRESH).
EndWetRoad: Last timestep that the road has water on it.
StartAnyPrecip: The first hour that any type of precipitation is forecasted (in hours from the start of the forecast).
EndAnyPrecip: The last hour that any type of precipitation is forecasted (in hours from the start of the forecast).
TotalLiquidPrecip: The total liquid water equivalent (mm).
TotalFrozenPrecip: The total depth of frozen precipitation (mm).
StartFrozenPrecip: The first hour that frozen precipitation is forecasted (in hours from the start of the forecast).
EndFrozenPrecip: The last hour that frozen precipitation is forecasted (in hours from the start of the forecast).
RoadTrend_PrestormType: The road temperature trend category prior to the start of storm precipitation (Warm, In-range, or Cold).
RoadTrend_PoststormType: The road temperature trend category after the end of storm precipitation (Warm, In-range, or Cold).
RoadTrend_StormType: The road temperature trend category during the storm. (Warm, In-range, Cold with up to 2 tiers, e.g. Cold to Warm).
PrecipType_StormType: The phase of precipitation during the storm (Snow, Freezing Rain, or Rain, with up to 3 tiers, e.g. Snow to Rain to Snow).
PrecipRate_StormType: Not Used. Characterize the change in storm intensity.
ColdBS_score: Score to indicate the likelihood that blowing snow will likely fall on cold roads.
StormTypeText: A text string describing the storm type.

E.5.d. Algorithm Parameters

The following three parameters control the fuzzy function for determining road temperature conditions and trends. They are meant to represent values at which we are 100% certain that the roads are indeed warm, in range, or too cold – not exact thresholds.

P_ROADT_WARM: The pavement temperature at which roads are considered warm. [Celsius] [10].
P_ROADT_INRANGE: The pavement temperature at which roads are considered appropriate for chemical treatment [Celsius] [-5].
P_ROADT_TOOCOLD: The pavement temperature at which roads are considered too cold for chemical treatment [Celsius] [-15].
P_ROADT_NUM_PRE: The number of hours prior to the start of precipitation that constitutes the pre-storm environment (hours).
P_ROADT_NUM_POST: The number of hours after the end of precipitation that constitutes the post-storm environment (hours).
Table E2. Nominal algorithm parameter values and ranges.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Nominal Value</th>
<th>Units</th>
<th>Expected Range</th>
<th>Precision</th>
</tr>
</thead>
<tbody>
<tr>
<td>P_ROADT_WARM</td>
<td>10.0</td>
<td>Degrees C</td>
<td>0 to 40</td>
<td>0.1</td>
</tr>
<tr>
<td>P_ROADT_INRANGE</td>
<td>-5.0</td>
<td>Degrees C</td>
<td>-10 to 0</td>
<td>0.1</td>
</tr>
<tr>
<td>P_ROADT_TOOCOLD</td>
<td>-15.0</td>
<td>Degrees C</td>
<td>-25 to 0</td>
<td>0.1</td>
</tr>
<tr>
<td>P_ROADT_NUM_PRE</td>
<td>6</td>
<td>Hours</td>
<td>1-12</td>
<td>1</td>
</tr>
<tr>
<td>P_ROADT_NUM_POST</td>
<td>6</td>
<td>Hours</td>
<td>1-12</td>
<td>1</td>
</tr>
</tbody>
</table>

E.5.e. Algorithm Constants

The following constants are used to index the type of precipitation:
- C_PTYPE_NONE: No precipitation (0).
- C_PTYPE_RAIN: Rain (1).
- C_PTYPE_SNOW: Snow (2).
- C_PTYPE_MIXED: Snow/rain mix (3).
- C_PTYPE_SLEET: Sleet (4).
- C_PTYPE_FRZRAIN: Freezing rain (5).

References


Appendix F: Technical Points of Contact

The primary technical points of contact for items contained in the MDSS Release-5 package are listed below.

<table>
<thead>
<tr>
<th>MDSS Technical Component</th>
<th>Source Lab</th>
<th>Technical Point of Contact</th>
</tr>
</thead>
<tbody>
<tr>
<td>METRo</td>
<td>Environment Canada</td>
<td><a href="http://home.gna.org/metro/">http://home.gna.org/metro/</a></td>
</tr>
<tr>
<td>Chemical Concentration Algorithms</td>
<td>LL</td>
<td>Robert G. Hallowell</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MIT Lincoln Laboratory</td>
</tr>
<tr>
<td></td>
<td></td>
<td>244 Wood Street</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Lexington MA 02420-9180</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ph: 781-981-3645</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Fax: 781-981-0632</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Email: <a href="mailto:bobh@ll.mit.edu">bobh@ll.mit.edu</a></td>
</tr>
<tr>
<td>Storm Characterization</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Road Condition and</td>
<td>NCAR</td>
<td>Bill Myers, Lead Software Engineer</td>
</tr>
<tr>
<td>Treatment Module</td>
<td></td>
<td>NCAR</td>
</tr>
<tr>
<td>Blowing Snow Potential</td>
<td></td>
<td>3450 Mitchell Lane</td>
</tr>
<tr>
<td>Algorithm</td>
<td></td>
<td>Boulder CO 80301</td>
</tr>
<tr>
<td>Precipitation Type</td>
<td></td>
<td>Ph: 303-497-8412</td>
</tr>
<tr>
<td>Algorithms</td>
<td></td>
<td>Fax: 303-497-8401</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Email: <a href="mailto:myers@ucar.edu">myers@ucar.edu</a></td>
</tr>
<tr>
<td>System Integration</td>
<td>Jim Cowie</td>
<td>NCAR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3450 Mitchell Lane</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Boulder, CO 80301</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ph: 303-497-2831</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Fax: 303-497-8401</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Email: <a href="mailto:cowie@ucar.edu">cowie@ucar.edu</a></td>
</tr>
<tr>
<td>MDSS Display Application</td>
<td>NCAR</td>
<td>Arnaud Dumont</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NCAR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3450 Mitchell Lane</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Boulder CO 80301</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ph: 303-497-8434</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Fax: 303-497-8401</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Email: <a href="mailto:dumont@ucar.edu">dumont@ucar.edu</a></td>
</tr>
<tr>
<td>Position</td>
<td>Organization</td>
<td>Name</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>MDSS Project Manager</td>
<td>NCAR</td>
<td>Kevin Petty</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Bill Mahoney</td>
</tr>
<tr>
<td>MDSS Program Leader Road Weather Management Program</td>
<td>FHWA</td>
<td>Paul Pisano</td>
</tr>
<tr>
<td>MDSS Program Support for the FHWA</td>
<td>Noblis</td>
<td>Andy Stern</td>
</tr>
</tbody>
</table>