Automated Safety Warning Controller Management and Maintenance Guide

By

Daniel Richter, Research Associate
Kelvin Bateman, Research Associate
Douglas E. Galarus, Program Manager, Senior Research Associate

Western Transportation Institute
College of Engineering
Montana State University

Prepared for

Caltrans
State of California Department of Transportation
Division of Research and Innovation

Attention: Sean Campbell
Attention: Ian Turnbull

March 22nd, 2013

REVISION HISTORY

Version	Date	Description	Description of Changes
1.0	09/21/2009	Phase 1 Installation guide	Preliminary Summary and Recommendations for Submission to Caltrans
2.0		Phase 2	Added changes to for the Phase 2 ASWC

Table of Contents

Introduction

This document describes the basic structure and setup of the Automated Safety Warning Controller (ASWC) developed by Western Transportation Institute under contract with the California Department of Transportation's Division of Research and Innovation. The controller interfaces with roadside sensors and signs. This allows for automated data collection and application of user defined scripts for placing alert messages on changeable message signs.

The controller system for Phase 1 ran on a Moxa UC-7420 RISC based embedded computer running Linux kernel version 2.4.18. The controller system for Phase 2 added the ability to run on a Moxa DA-661 rack mount RISC based embedded computer running Linux kernel version 2.6.10. All controller modules were written using the Python scripting language.

This document contains information about the security and logins used by the controller application; the directory structure of controller; information about the key files that control the behavior of controller; and step by step instructions for setting up a controller unit.

Security and Logins

There are four levels of permissions associated with the Controller device: An operator level which allows the user to view logs and status information, but not change anything; a slightly elevated supervisor level that permits impermanent changes to be made to a running Controller, a technician level that allows changes to be made to the Controller software installation, and an administrator level that allows full control over the device.

An operator accesses the device either through the TMC login over SSH, or through the Satellite Operations Center Command System (SOCCS) ASWC web interface. The SSH TMC login goes directly to the Controller command line interface. The operator can execute any commands that do not require the user to have first executed the enable command (see the command line interface documentation). The preferred method for operator access is through the SOCCS ASWC interface.

A supervisor accesses the system in the same way as an operator either through the TMC login over SSH, or through the SOCCS ASWC web interface, and all the same commands are available to a supervisor. The supervisor, however, if using the SSH command line interface can issue the enable command and be able to execute commands that change the state of a running Controller. The commands that can be executed after the enable command are documented in the command line interface documentation. The preferred method for supervisor access is through the SOCCS ASWC interface.

The SOCCS ASWC system allows logins to be setup for users with either operator or supervisor access levels.

Technicians access the device through the controller login over SSH or SFTP. They have full access to the Controller software installation (usually at directory: "/Controller"), but may not make operating system changes and do not have access to the " /etc" directory. Technicians may set the password for the controller login and the enable password for supervisor access.

System administrators have full control over the device. They have read and write permissions on the whole file system and may make changes to the operating system, Controller, or any of the logins. The system administrator does initial device setup, and any device level maintenance including setting passwords for any of the other permissions levels.

The passwords for the command line and SOCCS ASWC interface are stored in a file called passwd in the manager directory of Controller and can be set up using a simple script as described below under Controller Software Installation. All other permissions levels are handled as Linux users and their passwords may be set with the Linux `passwd` utility.

Directory Structure

The Moxa device stores the operating system, related binaries, configuration files, and the rest of the root file system on a flash ROM in the enclosure. The Controller file system is stored on a CompactFlash card that is inserted in device, which the Moxa software automatically mounts at `/var/hda`. The base path of the Controller application is configurable, but this document assumes paths starting at `/Controller/`. DO NOT copy the Controller directories to the Moxa's flash ROM, the flash ROM is not large enough for Controller to operate correctly.

/Controller/alert_logic
This directory contains the alert logic scripts. After a script is developed and tested the script file is placed in this directory. The script names must end in `.py` and otherwise be consistent with the name of the alert logic script in the configuration file.

/Controller/data
This directory stores all the data files used by field elements. The files store current and past data, for persistence if the Controller application is restarted, and for history. Data is currently stored in csv format. There will be a data file for each field element configured in the system including CMS signs, RWIS, MVDS, and Loop sensors. Note that the Controller system archives data files by appending a date to the end of the filename.

/Controller/field_elements
This directory contains the modules needed to communicate with different field elements. The files in this directory should only be changed when updating software or adding the capability to communicate with a new type of field element. Two files that may need to be viewed/modified are: ConversionDict.py and OIDDict.py. ConversionDict.py contains a dictionary of conversions to be applied to rwis sensor values, for instance to convert from degrees Fahrenheit to Celsius. OIDDict.py contains a dictionary of OID values for NTCIP sensor names, this can be referenced to get valid sensor names but shouldn't have to be changed. Examples of both of these files are listed at the end of this document.

/Controller/log
This directory stores the log files generated by Controller. `CMS.log` contains errors and informational messages produced by the CMS field elements, such as when a message was placed on the sign, or why one wasn't. It is also where CMS messages generated by alert scripts are written.
`QC.log` contains all the sensor readings that fail quality control checks, as well as the value that failed the check.
`system.log` stores all other messages generated by the Controller application: program errors, failures in communication with field elements, errors or bugs in alert logic scripts, etc.

/Controller/manager
This directory contains the heart of the application code, as well as the configuration files. Generally the only files in this directory that need to be changed are `AlertLogic.ini`, `FieldElement.ini`, `Messages.ini` and `QualityControl.ini`.
`AlertLogic.ini` contains configuration variables for alert logic scripts.

`FieldElement.ini` contains configuration variables for the field elements. `QualityControl.ini` contains minimum and maximum values for every unit that field element data is measured in. `Messages.ini` contains a list of alert messages that can be used by the alert scripts.

Controller Software Installation

The Controller software can be placed on a CompactFlash card, and the card can simply be inserted into the Moxa enclosure. The Controller directories can be placed at the root level on the card, or any level of directories deep, as long as the controller.ini file (discussed later) has the absolute path of the Controller directories in it. Note that the CompactFlash card needs to be formatted with the ext2fs file system, so to install the Controller software on the card using a PC, the PC must support ext2fs.

An automated script has been created for installing the Controller software on a Moxa device. Follow these installation instructions:

1. Put the Controller directory and install_controller.sh script on the controller device (putting them on a usb key and plugging it in to the DA-661 works pretty well, alternatively you can type upramdisk on the device and ftp the files to /mnt/ramdisk)
    a. Note that the Controller directory and its sub-directories will be copied to the Moxa device as part of the install procedure. You can create your own version of Controller to be installed on devices by adding your custom files to the appropriate location on the installation media. For instance you may want to include the same alert scripts or the same Messages.ini files on all your installations.
2. Run the install script
    a. The script will ask a series of questions and then setup the appropriate directories and system settings.
3. Set up users for SOCCS:
    a. python /var/hda/Controller/manager/controllerpasswd.py adduser USERNAME PASSWORD OPERATOR|SUPERVISOR
    b. the last argument must be either OPERATOR or SUPERVISOR in all caps
4. Set up user for enable password in command line interface:
    a. python /var/hda/Controller/manager/controllerpasswd.py adduser ThreadMonitor PASSWORD
    b. The username must be ThreadMonitor for it to be recognized as the enable password
5. Start Controller: /etc/init.d/controller start

Controller Passwords

The SOCCS and enable passwords are kept in a file named passwd in the “Controller/manager directory”. This file contains the username, hashed password, and OPERATOR or SUPERVISOR level of access. The fields are separated with a 0xff character (value 255).

Editing Configuration Files

There are 6 configuration files that control the behavior of Controller and need to be edited for a new installation. Five of them are contained in the “`/Controller/manager`” directory, the fifth is in `/etc/controller`.

AlertLogic.ini

The AlertLogic.ini file contains a section for each alert logic script with values that control the behavior of the script. A sample section would look like this:

```
windwarning:
{
    Status : 1
    InitialDelay : 120
    Interval : 60
    Timeout : 2
    Threshold:
    {
        highwindwarning : { type: "int", value: 56 }
        windwarning : { type: "int", value: 41 }
        windgustwarning : { type: "int", value: 46 }
        windadvisory : { type: "int", value: 26 }
        NextMilesMessage : { type: "str", value: "NEXT 5 MILES" }
        highwindwarningmessage : { type: "messagename", value:
"HighWindWarningMessage" }
        windwarningmessage : { type: "messagename", value:
"WindWarningMessage" }
        windadvisorymessage: { type: "messagename", value:
"WindAdvisoryMessage" }
    }
}
```

This example demonstrates the configuration for an alert script called windwarning. Note that there must be a corresponding script called `windwarning.py` in `/controller/alert_logic/`.

The **Status** variable is used to turn scripts on and off for the duration of the running of the application. A value of 1 means this script will be started, a value of 0 means it will not be started. For example, there may be a standard installation that is placed on all the Controller devices which includes a congestionwarning script that utilizes a loop detector. On installations that aren't connected to a loop detector it would be pointless to run this script; on those installations the status could be set to 0, so the script never runs.

The **InitialDelay** variable sets the amount of time the script waits before its first run after the application starts. This allows the field element scripts to gather some data prior to the alert script running. This value is in seconds.

The **Interval** variable is the amount of time the script pauses between subsequent runs. This value is in seconds.

The **Threshold** variable defines threshold values that are accessible by alert logic scripts. Defining a threshold in this configuration file, as opposed to hard coding them in a script, allows the variable to be viewed/changed from the command line interface. Specified with each threshold value is its type and value. Valid types are; "str" for strings, "int" for integers, "float" for floating point and "messagename" for messages configured in the Messages.ini file. For example, the windwarning script configured by the preceding example may contain a condition like this: "`if RWISStation.essAvgWindSpeed() > highwindwarning:`". The

highwindwarning variable is defined in the configuration file and could be changed from location to location by changing the value in the configuration file. No modifications to the script would need to be made. The values of threshold variables can also be changed temporarily through the command line or web interface, although the change is lost if the device reboots or the application is restarted.

FieldElement.ini

This file contains a section for every field element, and they're formatted similar to the sections in AlertLogic.ini. The name for each section can be arbitrary; it doesn't need to correspond to anything else like the section names in AlertLogic.ini do. The FieldElement names are what are recorded in the logfile, used by alert logic scripts to access sensor data, and used for interaction in the command line and web interfaces, so they should be descriptive. This is an example of a typical section in FieldElement.ini:

```
RWIS:
{
      Status : 1
      Type : "rwis"
      IPAddress : "10.10.31.20"
      Port : 161
      RetryNumber : 3
      RetryInterval : 5
      CommunicationTimeout : 30
      SNMPCommunity : "private"
      Interval : 60
      MaxFileLength : 2000
      SavePeriod : 30
      DataValidTimeout : 340
      Location : { 'Lat' : 40.0024, 'Long' : 0-120.9579}

      County : "Plumas"
      Highway : "70"
      PostMile : "50.86"
      Sensors:
      {
            "essAirTemperature1"  : "degF"
            "essRelativeHumidity" : "pct"
            "essAvgWindDirection" : "degrees"
            "essAvgWindSpeed"     : "mph"
            "essMaxWindGustSpeed" : "mph"
            "essDewpointTemp"     : "degF"
            "essVisibility"       : ""
            "essSubSurfaceTemperature" : "degF"
            "essPrecipitationTwelveHours" : ""
            "essPrecipitationSixHours" : ""
            "essPrecipitation24Hours" : ""
            "essPrecipitationThreeHours" : ""
            "essPrecipitationOneHour" : ""
            "essSurfaceStatus1" : ""
            "essSurfaceStatus2" : ""
            "essSurfaceStatus3" : ""
            "essSurfaceStatus4" : ""
            "essSurfaceTemperature1" : ""
            "essSurfaceTemperature2" : ""
```

```
            "essSurfaceTemperature3" : ""
            "essSurfaceTemperature4" : ""
      }
      Fieldnames:
      {

            "essAirTemperature1"
            "essRelativeHumidity"
            "essAvgWindDirection"
            "essAvgWindSpeed"
            "essMaxWindGustSpeed"
            "essDewpointTemp"
            "essVisibility"
            "essSubSurfaceTemperature"
            "essPrecipitationTwelveHours"
            "essPrecipitationSixHours"
            "essPrecipitation24Hours"
            "essPrecipitationThreeHours"
            "essPrecipitationOneHour"
            "essSurfaceStatus1"
            "essSurfaceStatus2"
            "essSurfaceStatus3"
            "essSurfaceStatus4"
            "essSurfaceTemperature1"
            "essSurfaceTemperature2"
            "essSurfaceTemperature3"
            "essSurfaceTemperature4"
      }
}
```

The **Status** variable is used to determine if this field element will be activated. A value of 1 means this field element will be started, a value of 0 means it will not be started.

The **Type** variable is used by the program to determine what type of field element to associate with this particular field element. For each type of field element there are two files in `/Controller/field_elements/`: `type_module.py` and `FieldElement_type.py`. For this reason **Type** always needs to be one of cms, loop, or rwis. Other types may be added in the future such as rtms).

IPAddress and **Port** are used for communication with the field element. All communication between the Controller device and field elements utilizes IP, so these variables are essential.

RetryNumber and RetryInterval are used when communicating with the field element. They specify the number of times to retry communications prior to reporting an error and the interval, in seconds, to wait between retry attempts.

SignAddress is only utilized by CMS field elements.

CommunicationTimeout sets how long the application tries to communicate with the physical field element before giving up. This can be adjusted to accommodate the lag time to establish a dialup connection.

The **Interval** variable is the amount of time the script pauses between subsequent runs (this value is only the pause interval, so actual time from one run to the next will be this value PLUS the amount of time it takes to run). This value is in seconds. Note that for a CMS module this will also be the minimum amount of time a message will remain on the CMS before possibly being superseded by a higher priority message so be careful about making it too short.

The **MaxFileLength**, and **SavePeriod** variables are used for the archive feature. **AutoArchive** turns the feature on (1) or off (0), **MaxFileLength** sets how large the data file can get (number of lines) before it's archived. **SavePeriod** specifies the maximum age of the data (in minutes) that is kept in the original file after the archive process runs; this assures that there is sufficient data available for alert scripts. In the example, when a data file gets more than 2000 records, the file is archived to a backup file, and everything newer than 30 minutes is kept in the current data file. Note that since we are using a simple, flat file structure that archiving is recommended to increase performance.

The **Location**, **County**, **Highway** and **PostMile** variables are merely informational.

The **DataValidTimeout** variable specifies how old the current data can be before it's considered expired or invalid. Once the current data is expired it is no longer used by the alert logic scripts.

The **Sensors** variable contains a list of the sensor names that should be read, and data stored, for the particular field element. In the case of an RWIS these would be standard NTCIP variable names. The RWIS field element automatically converts temperatures to Fahrenheit and wind speeds to MPH. The unit names are used for quality control, and must match the section names in QualityControl.ini. The Sensor variable, like SignAddress, is not used by all field element modules.

The **Fieldnames** variable contains a list of fieldnames that will be stored in the appropriate data file. The fields will be stored in the file in the order they are listed within the Fieldnames variable. This can help make the data files more readable.

Messages.ini

This file is used to define messages that can then be used by alert scripts. Defining the sign messages in the Messages.ini file can help make the alert scripts shorter as well as keep the sign messages more consistent.

```
IceWarningMessage:
{
    MessageType: "Page1Normal"
    FontPage1: "SingleStroke"
    FontPage2: "SingleStroke"
    DisplayTime: 50
    Priority: 10
    Expiration: 600
    MessagePage1Line1: "ICY CURVES"
    MessagePage1Line2: "AHEAD"
    MessagePage1Line3: ""
    MessagePage2Line1: ""
```

```
    MessagePage2Line2: ""
    MessagePage2Line3: ""
}
```

The following values should be listed for each message:
MessageType: PageBlank,Page1Normal, Page1Flash, Pages2Extended
FontPage1: SingleStroke, DoubleStroke
FontPage2: SingleStroke, DoubleStroke
DisplayTime: integer value in tenths of a second
Priority: integer value higher numbers are higher priority
Expiration: integer time until message is expired, in seconds
MessagePage1Line1: ""
MessagePage1Line2: ""
MessagePage1Line3: ""
MessagePage2Line1: ""
MessagePage2Line2: ""
MessagePage2Line3: ""

QualityControl.ini
This file is used to define min and max values for each unit that field elements return data in. These can probably be defined once and never need to change for any other installation. A single section is fairly simple:

```
degF:
{
      min: 0-70.0
      max: 100.0
}
```

The name of the section must match the unit as it appears in the Sensors variable of the field element that gets the data. Also note that in the .ini files negative numbers must be subtracted from zero, a negative sign before a number causes an error.

passwd
This file contains the enable password in the command line interface and the passwords for the SOCCS ASWC admin screen. It's located in `/controller/manager/`. It can have comments; any line that begins with a # is ignored, and the first line that doesn't begin with a # is the enable password. The password is hashed so not easily readable. A python script has been created to maintain this file: controllerpasswd.py. The following commands are available:

python controllerpassword.py adduser username password [OPERATOR|SUPERVISOR]
python controllerpassword.py removeuser username
python controllerpassword.py changepassword username password

config.ini
This is the only file that isn't located in `/Controller/manager/`, it's found in `/etc/controller/` (though a symbolic link could be used to locate it anywhere). This configuration file has no separate sections, and only four variables.

```
RootPath :
"/var/hda/Controller"
LogRotateIntervalDays : 3
NumSavedLogs: 5
UseWatchdog: no
```

The **RootPath** variable sets the base path for Controller, meaning it doesn't need to be at `/Controller`.

The **LogRotateIntervalDays** sets how often the log files are rotated out, in the example it's every 3 days. When a log file is rotated out the old log file has the current date appended to the file name and a new empty file takes its place.

If **NumSavedLogs** is nonzero, at most NumSavedLogs files will be kept, and if more would be created when rollover occurs, the oldest one is deleted.

The **UseWatchdog** sets whether or not to use an internal watchdog to prevent the controller application from hanging. If set to **yes** the system will reboot if a signal isn't received from the Contoller application every 30 seconds.

Install Script Information

The following outlines the information that will be requested by the ASWC install script:

1. Defaults file found, use values in defaults file? (y/n)
    a. If the values from a previous run of the installation script were stored you will have the option of loading and using the default values.
2. Choose a timezone
    a. You will be prompted to choose between Eastern, Central, Mountain, or Pacific timezones.
3. Please enter the current date
    a. You will be prompted for MM, DD, YYYY
4. Set time periodically on this device with NTP? (y/n)
    a. If yes you will be prompted for the IP address of the NTP server and how often (in seconds) to update the clock
5. Please insert the Compact Flash card and press enter. Note that contents of the card will be lost. To exit press Ctrl-c
6. How often should the log files be rotated (in days)?
    a. A new log will be started each interval, the previous log will be saved with the date appended to the name. As currently configured the last 5 log files will be saved with older ones deleted.
7. Will the network be configured automatically with dhcp (y/n)?
    a. If you answer “n” the system will prompt for the IP Address, Subnet Mask, Network, and Gateway addresses.
8. Store answers in default file? (y/n)
    a. If “y” the values previously entered will be saved for future use.

SOCCS Client Installation

The following outlines the installation instructions for the SOCCS ASWC client software:
Any machine running apache and mod_php will do. The Moxa DA-661s come set up like this by default. The same device that is running controller may also serve up the SOCCS client software although for best performance the SOCCS client software should be installed on a machine that is local.

1. Copy the *contents* of the SOCCSClient directory from the installation media to /home/httpd/html/ (on the DA-661, on any other machine copy it to whatever directory is designated as the DocumentRoot in the apache httpd.conf file).
2. Verify that apache is running.
    a. You can use the "ps" command and look for apache/httpd.conf process.
3. Check the changereasons.txt file located with the SOCCS client files and edit as necessary.
    a. This is a simple text file that contains the reasons, one per line, which will populate the dropdown box for the CMS message chooser window of the SOCCS client.
4. Set up the list of Controllers that will be monitored by the SOCCS client:
    a. On a client machine, browse to http://*serveraddr*/config.php
    b. Click "Add"
    c. Enter:
        i. Location: Location is just a descriptive name of the Controller installation, and may be anything.
        ii. IP Address: the ip address of the Controller.
        iii. Update Interval: Update interval is how often the list should retrieve status information from the Controller device in minutes.
        iv. SOCCS Timeout: How the SOCCS interface will wait for a connection with the Controller before timing out.
        v. Admin Update Interval: Admin update interval is how often the admin window, if open, should retrieve status information from the Controller device in minutes. Note that since the Admin window tries to keep an authenticated connection with the Controller open that in the case of a dialup connection this interval should be less than the dialup inactivity hang-up interval.
    d. Click "Save"
5. Create the documentation directory.
    a. The SOCCS ASWC documentation must be stored on a compact flash card installed in the Moxa as there is not enough room for the files on the built-in memory.
    b. Move the SOCCSDocumentation from the installation media to a directory on the compact flash card. Be sure the images subdirectory gets copied (the –r option on the "cp" command will copy the subdirectory also)
    c. Create a symbolic link for the documentation directory.
        i. From the /home/httpd/html directory (or the directory designated by the DocumentRoot if different) use the following command to create a link: "ln –s /var/hda/documentation/ documentation"

ii. Note that if necessary replace "/var/hda/documentation" with the actual location of the documentation files.

6. Browse to http://*serveraddr*/SOCCSAutomatedController.html to access the SOCCS client.

Writing Alert Scripts

The /Controller/alert_logic directory contains a template directory with a basic alert script template. The alert scripts are written in the Python programming language, and all facilities available in Python are available to the alert scripts; however, much of what is needed to integrate into the Controller application is provided, so the scripts can be very simple. Field elements can be accessed directly by name, as they appear in FieldElement.ini. Alert scripts will generally include a condition with actions to be taken if the condition is true, usually putting a message into the sign queue.

```
#Script Template
"""
Variables          possible values (values between asterisks represent the
default values)
These are the components of a message in the Messages.ini file, any component
can be overridden using the syntax: msg.component = xxx
messageType        *PageBlank*,Page1Normal, Page1Flash, Pages2Extended
fontPage1          *SingleStroke*, DoubleStroke
fontPage2          *SingleStroke*, DoubleStroke

displayTime        *50* integer value in 1/10 seconds
priority           *10* integer value, higher number is higher priority
expiration         *10 Minutes* time until message is expired, in seconds
messagePage1Line1  *empty string* string value
messagePage1Line2  *empty string* string value
messagePage1Line3  *empty string* string value
messagePage2Line1  *empty string* string value
messagePage2Line2  *empty string* string value
messagePage2Line3  *empty string* string value

"""
# grab a message from the Messages.ini file
msg = Messages[predefinedmessagename]

if FieldElement.sensorname(optional time span) [>, >=, <, <=, ==, !=]
threshold:
    # Example of overriding  part of the message
    msg.messagePage1Line1 = "Custom first line"
       msg.priority = 5
    CMSSignName.WriteLog("Put this message in the log")
    CMSSignName.AddMessageToQueue(msg)
```

FieldElementName can be any field element that was listed in FieldElement.ini, and **sensorname** can be anything listed in the **sensornames** variable of that field element. The parentheses after the sensor name are required by Python, though in most cases they will be empty, as in the example. Optionally a time span may be put inside the parentheses, measured in minutes, which will return a list of all the sensor readings found in that time span. Members of the list are accessed with square brackets, starting at 0, so for example: `FieldElementName.sensorname(60)[0]` will return the earliest sensor reading within the last hour. Python provides a few built in operations that can be done on a list, namely min, max, and

avg. For example, the average of all the readings over the last hour is given by `avg(FieldElementName.sensorname(60))`.

Comparison can be any of the python comparison operators <, <=, >, >=, ==, !=, which are, respectively, less than, less than or equal to, greater than, greater than or equal to, equal to, and not equal to. **Threshold** is the name of any of the thresholds listed in the alert script's section of AlertLogic.ini or alternatively a hard coded value. Note that if you hard code a threshold value it can only be changed by changing the script. Multiple conditions can be combined with "`and`" or "`or`" between them, a condition consisting of two or more conditions separated with an "`and`" will be true only if all the conditions are true. A condition consisting of two or more conditions separated by an "`or`" will be true if any one of the conditions are true.

CMSSignName can be any sign listed in FieldElement.ini, namely, a FieldElement in which the variable `Type` is equal to cms. Several parameters are automatically provided for setting the message on a sign. **messageType** can be any of: PageBlank, which is the default; Page1Normal; Page1Flash; or Pages2Extended. **fontPage1** and **fontPage2** can be set individually, and be either SingleStroke or DoubleStroke. Up to 6 message lines are available: two pages of three lines each, though for the second page to display messageType must be set to Pages2Extended. If the Message type is Pages2Extended, **displayTime** is the time in tenths of a second between page switches. It defaults to the maximum value of 30, or 3 seconds.

The **priority** and **expiration values** are both used to control the behavior of the CMS field element as it determines what message to put on the sign next. Priority is an integer from 0 to 32767, higher numbers get precedence. If a message in the queue has a higher priority than the message currently on the sign, the sign will be set with the higher priority message, even if the other message hasn't yet expired. While the message currently on the sign may be preempted by another message, it isn't deleted until it expires, so if the high priority message expires before the lower priority message, the lower priority message will be placed back on the sign. **expiration** is a number in seconds, representing how long the message should stay in the queue. Long expiration periods mixed with short run intervals are not recommended as the message queue may grow too large.

WriteLog("Put this message in the log") adds a message to the log for the sign. Note that Python formatting and variable substitution may be used for example: **WriteLog("Icy Curve Warning. Values: %d, %d" %(RWIS.essSurfaceStatus1(),RWIS.essSurfaceStatus2()))** will replace the two %d (for decimal values) with the two RWIS values in the parenthesis.

AddMessageToQueue(msg) adds the message to the message queue for the designated sign. In this example we assigned an entry from the Messages.ini file to the variable "msg" so we could modify compnents of the message. An alternative is no modifications are necessary is **AddMessageToQueue(Messages[predefinedmessagename])**

Python Syntax

Indentation matters. A conditional statement always has one or more statements following it that should only be done if the condition is true. The statements associated with a condition are indented one tab further than the conditional statement. The first line that is not indented will be executed regardless of whether or not the condition is true.

Conditional statements are ended by a colon.

Every statement must be on a line by itself, and only use one line. If a statement is too long to fit on one line, or is more aesthetically appealing or more readable when divided between lines, a \ can be used at the end of a line to indicate the statement is continued on the next.

Regular Maintenance

Weekly

Check for errors that have occurred and address them if necessary. Use the grep utility to search the week's log files for the word ERROR: `grep ERROR /controller/log/system.log`. Repeat for each log file that has been generated over the week.

Check the amount of free space on the CompactFlash card with the command `df -h` (the card will be the one labeled `/dev/hda1`). If the space available is less than 200M then some log files and data files should be moved off the device, which can be done easily with SFTP. Both log and data files have the date appended to the end of the file name after they have been archived, signifying they are just historical and are no longer used by the system. If the free space on the card is getting low then every file ending with a date should be copied to an external system and erased on the Controller device.

Monthly

Copy all archived log and data files from the Controller device to an external device and delete them off the Controller CompactFlash card.

Reference

The following pages are for reference to sensor names when writing alert logic scripts, or, if the need arises, to change the Controller application itself. The two python files quoted are stored in the field_elements directory and are used for Controller's interface with the outside world. If the unit conversion for a sensor needs to be added or changed the ConversionDict.py file may be modified. If an RWIS sensor needs to be accessed but is not in OIDDict.py it may be added. In light of these possible modifications the following are for reference only, the final authoritative source of the behavior of a device is the code on the device itself.

ConversionDict.py

This file is used by the program to make automatic unit conversions after reading from an RWIS sensor. After a sensor is read, if this dictionary contains the sensor, then the formula will be applied to the value read. Quality Control checks occur after the conversion, so they should be done in the unit that is being converted to.

```
ConvDict = {
        'essAirTemperature1': '32+1.8*float(value)/10.0',  #Temp. F = (1.8)*Temp C+32
        'essAirTemperature2': '32+1.8*float(value)/10.0',
        'essAirTemperature3': '32+1.8*float(value)/10.0',
        #convert wind speed from tenths of meters per second; 1m/s=2.2369362920544mph
        'essAvgWindSpeed': 'float(value)/10*2.24',
        'essMaxWindGustSpeed':'float(value)/10*2.24',
        'essRelativeHumidity':'float(value)/100.00',  # convert % to decimal
        'essDewpointTemp':'32+1.8*float(value)/10.0',
        'essSurfaceTemperature1':'32+1.8*float(value)/10.0',
        'essSurfaceTemperature2':'32+1.8float(value)/10.0',
        'essSurfaceTemperature3':'32+1.8*float(value)/10.0',
        'essSurfaceTemperature4':'32+1.8*float(value)/10.0',
        'essSurfaceTemperature5':'32+1.8*float(value)/10.0',
        'essSurfaceTemperature6':'32+1.8*float(value)/10.0',
        }
```

OIDDict.py

This file maps sensor names to OIDs for the RWIS variables. If an OID is wrong or a sensor is excluded this dictionary should be changed and Controller restarted. The OIDs are all the digits that occur after the common initial 1.3.6.1.4.1.1206.

```
OIDDict = {
        'essAirTemperature1': '4.2.5.2.5.2.1.3.1,-1000..1001',
        'essAirTemperature2': '4.2.5.2.5.2.1.3.2,-1000..1001',
        'essAirTemperature3': '4.2.5.2.5.2.1.3.3,-1000..1001',
        'essWindSituation0':  '4.2.5.2.4.3.0',
        'essAvgWindDirection': '4.2.5.1.11.1.0,0..361',
        'essAvgWindSpeed': '4.2.5.1.11.2.0,0..65535',
        'essMaxWindGustSpeed':'4.2.5.1.11.41.0,0..65535',
        'essMaxWindGustDir':'4.2.5.1.11.43.0,0..361',
        'essRelativeHumidity':'4.2.5.1.13.3.0,0..101',
        'essDewpointTemp':'4.2.5.2.5.4.0,-1000..1001',
        'essSurfaceStatus1':'4.2.5.2.9.2.1.7.1',
        'essSurfaceStatus2':'4.2.5.2.9.2.1.7.2',
        'essSurfaceStatus3':'4.2.5.2.9.2.1.7.3',
        'essSurfaceStatus4':'4.2.5.2.9.2.1.7.4',
        'essSurfaceStatus5':'4.2.5.2.9.2.1.7.5',
        'essSurfaceStatus6':'4.2.5.2.9.2.1.7.6',
        'essNtcipCategory':'4.2.5.2.1.2.0',
        'essSurfaceWaterDepth':'4.2.5.2.9.2.1.10.1',
        'essSurfaceConductivity':'4.2.5.2.9.2.1.12.1',
        'essPressureHeight':'4.2.5.2.3.2.0',
        'essPrecipitationEndTime':'4.2.5.2.6.9.0',
        'essSpotWindDirection':'4.2.5.2.4.1.0',
        'essMinTemp':'4.2.5.2.5.6.0',
        'essPavementTemperature':'4.2.5.2.9.2.1.9.1',
        'essPavementSensorError':'4.2.5.2.9.2.1.15.1',
        'Globaltime':'4.2.6.3.1.0',
        'essPrecipitationOneHour':'4.2.5.1.13.19.0',
        'essVisibility':'4.2.5.2.8.1.0',
        'numEssPavementSensors':'4.2.5.2.9.1.0',
        'essSurfaceFreezePoint':'4.2.5.2.9.2.1.13.1',
        'essPrecipitationTwelveHours':'4.2.5.1.13.22.0',
        'essMaxWindGustDir':'4.2.5.1.11.43.0',
        'essSubSurfaceTemperature':'4.2.5.2.9.4.1.5.1',
        'essPrecipYesNo':'4.2.5.2.6.5.0',
        'essPrecipitationSixHours':'4.2.5.1.13.21.0',
        'essAtmosphericPressure':'4.2.5.1.7.4.0',
        'essNumTemperatureSensors':'4.2.5.2.5.1.0',
        'essPrecipitation24Hours':'4.2.5.1.13.23.0',
        'essSurfaceSalinity':'4.2.5.2.9.2.1.11.1',
        'essPrecipitationThreeHours':'4.2.5.1.13.20.0',
```

```
'essSurfaceTemperature1':'4.2.5.2.9.2.1.8.1',
'essSurfaceTemperature2':'4.2.5.2.9.2.1.8.2',
'essSurfaceTemperature3':'4.2.5.2.9.2.1.8.3',
'essSurfaceTemperature4':'4.2.5.2.9.2.1.8.4',
'essSurfaceTemperature5':'4.2.5.2.9.2.1.8.5',
'essSurfaceTemperature6':'4.2.5.2.9.2.1.8.6',
'globalDaylightSaving':'4.2.6.3.2.0',
'essSpotWindSpeed':'4.2.5.2.4.2.0',
'essNtcipTypeofStation':'4.2.5.1.2.1.0',
'essPrecipitationStartTime':'4.2.5.2.6.8.0',
'essPrecipRate':'4.2.5.1.13.14.0',
'essPrecipSituation':'4.2.5.2.6.6.0',
'essMaxTemp':'4.2.5.2.5.5.0',
'essSurfaceBlackIceSignal':'4.2.5.2.9.2.1.14.1',
'essWetBulbTemp':'4.2.5.2.5.3.0',
'essReferenceHeight':'4.2.5.2.3.1.0',
'essWindSensorHeight':'4.2.5.2.3.3.0',
'essWaterDepth':'4.2.5.2.6.1.0'
}
```

Loop Module Sensors

The following is a list of available loop module variables:

'Headway1'
'Headway2'
'Headway3'
'Headway4'
'Headway5'
'Headway6'
'Headway1_avg'
'Headway2_avg'
'Headway3_avg'
'Headway4_avg'
'Headway5_avg'
'Headway6_avg'
'Speed1' #current speeds range from
'Speed2' #5 to 82 miles per hour
'Speed3' #if > 82mph then zero
'Speed4' #unit is mph
'Speed5'
'Speed6'
'Speed1_avg'
'Speed2_avg'
'Speed3_avg'
'Speed4_avg'
'Speed5_avg'
'Speed6_avg'
'Occupancy_A'
'Volume_A'
'Occupancy_B'
'Volume_B'
'Occupancy_C'
'Volume_C'
'Occupancy_D'
'Volume_D'
'Occupancy_E'
'Volume_E'
'Occupancy_F'
'Volume_F'
'Occupancy_G'
'Volume_G'
'Occupancy_H'
'Volume_H'
'Occupancy_I'
'Volume_I'
'Occupancy_J'
'Volume_J'
'Occupancy_K'
'Volume_K'
'Occupancy_L'
'Volume_L'
'Occupancy_M'
'Volume_M'
'Occupancy_N'
'Volume_N'
'Occupancy_O'
'Volume_O'
'Occupancy_P'
'Volume_P'