Automated Safety Warning Controller–
Controller Message Protocol Definition Document

Prepared by

Kelvin Bateman
Research Associate

Dan Richter
Research Associate

and

Douglas Galarus
Program Manager: System Engineering, Integration and Development

Western Transportation Institute
Montana State University
PO Box 174250
Bozeman, MT 59717-4250
Phone: (406) 994-6320

for the

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

March 22nd, 2013

REVISION HISTORY

Version	Date	Description	Description of Changes
0.9	7/11/2011	Baseline controller message protocol definition document	Preliminary controller message protocol definition for submission to Caltrans
0.91	12/22/2011	Baseline controller message protocol definition document	Added module status to GET OUTPUTELEMENTS response and changed AuthStatusReply to return authentication level.
1.0	3/22/2013	Baseline controller message protocol definition document	Draft version promoted to version 1.

Scope

The intent of this document is to define the network protocol used for administration and configuration of the Automated Safety Warning Controller (ASWC). The most recent version of this document will serve as the authoritative standard for implementation of the protocol on both the server (the ASWC device) and the client. The use of Satellite Operations Center Command System (SOCCS) Automated Controller as the client is assumed, though any client may implement this protocol. This document assumes a working knowledge of the ASWC and its features and operation. For further information about the ASWC see Automated Safety Warning Controller System Concept and Requirements Specification, Automated Safety Warning Controller Testing and Development Lab Summary and System Development Summary, Controller Installation Guide (1,2,3).

Background

The ASWC interfaces with roadside devices such as sensors and signs. The ASWC allows for automated data collection and application of best practice algorithms to analyze sensor data and to actuate related warning messages and signals. For instance, wind warning messages might be actuated on a changeable message sign (CMS) when wind speed, as read from a sensor, exceeds a given threshold.

Remote access for administration and monitoring is crucial for successful deployment, particularly in isolated rural areas. It is neither feasible nor cost-effective to require on-site access to the ASWC for such tasks. ASWCs and related roadside devices will be deployed in areas that are hours away from the nearest technician. If such “service calls” were necessary to administer the ASWC, warning systems would be left inoperable for hours if not days.

Document Organization

The remaining five sections of this document provide a detailed description of the protocol used by SOCCS to configure and manage the ASWC. The *Communication Overview* section describes the network configuration and base protocols used to communicate with the ASWC. The *Message Sequence* section establishes the expected order of messages for authentication, data retrieval, and error notification. The *Message Overview* section describes the list of defined messages, their intended purpose, and data expected from the client and returned by the ASWC. The *Detailed Message Syntax Definition* section provides a comprehensive description of the protocol message formats using Backus-Naur Form. The *Examples* section contains eight sample messages showing the authentication sequence, a status request, and a notification sent to the ASWC of a message that has been put on the sign.

Communication Overview

Each ASWC will be uniquely addressable by IP address. Communication will occur over a TCP socket; the ASWC will listen on TCP port 6467. All ASWC data, including authentication data, transmitted over this TCP socket will be encrypted using Transport Layer Security (TLS) version 1.2 or greater. The ASWC may optionally provide generic status information over an unencrypted, unauthenticated UDP channel on UDP port 6467. Only the following GET Commands may be handled by the unencrypted UDP channel: `SIMPLESTATUS`, `INPUTELEMENTS`, `OUTPUTELEMENTS`, and `OUTPUTELEMENTMSG`.

Message Sequence

This section will outline the basic sequences for establishing a connection with the ASWC and sending command messages to the ASWC.

Connection Establishment

Upon connection to the ASWC, the client should immediately initiate a TLS handshake; all protocol data will be encrypted. At the successful completion of the TLS handshake the client should issue an `AUTHINIT` command to the ASWC. The ASWC will reply with an `AuthChallenge` response to the client, to which the client should respond with an `AUTH` command. The ASWC will then respond with an `AuthStatusReply` message indicating successful or failed authentication. If the response is positive then the client may immediately issue a `Command` message. If the response is negative the client should restart the authentication process by sending another `AUTHINIT` message. Immediately after the third negative `AuthStatusReply` message the ASWC will close the connection and log the IP address of the client with a message indicating the three failed attempts to connect.

Command Messages

After successful authentication, the ASWC is ready to accept and execute command messages, and will continue to listen for commands until the client closes the connection. The ASWC operates in a polled mode, responding to messages sent from a client. All messages successfully received are replied to with either the appropriate data or an acknowledgement of successful receipt. The command messages fall into two categories: GET messages that are used to receive information from the ASWC and PUT messages that are used to send data or configuration settings to the ASWC.

Upon receipt of a Message, the ASWC validates the Message using both the length and checksum fields of the message. If the message fails either check the ASWC will reply with an `INVALIDCOMMAND` error message. If the client receives an `INVALIDCOMMAND` error message it should resend the last message and/or notify the the user that the command could not be executed. Typical practice is to retry the message up to three times prior to notifying the user however this is a client side decision. If the message passes both length and checksum checks, the command is parsed and executed. If the command fails during execution it returns a `COMMANDEXECFAILED` error message. If the client receives a `COMMANDEXECFAILED` error message it should immediately notify the user, without resending the command.

The response message of a PUT command is always the old value and the new value, unless the command is `LOG` or `OUTPUTELEMENTNOTIFY` wherein the response will simply be the new value. The response of a GET command is the information requested. If a parameter passed to a GET or PUT command is invalid, such as an undefined field element name, the ASWC will respond with an `INVALIDPARAM` error message with the invalid parameter listed in the error message detail.

Upon receipt of a response from the ASWC, the client should validate the message using the length, checksum, and message number fields of the message. Each command message sent from the client contains a message number that is returned in the response from the ASWC; the client should use the message number to verify that the response corresponds to the request. If the client does not receive a response within a configurable amount of time, or the response fails validation, the client should send the command again and/or notify the user that the request could not be completed. If the failed request is a PUT request, the client has no way of ascertaining whether the command was executed and should execute the corresponding GET command to make sure SOCCS and the ASWC have the same value for that setting. If the GET command corresponding to a failed PUT command fails, the user should be notified of the network failure and the possibility that the ASWC and the client may be out of synchronization on the requested setting. For example, if the client issues a `PUT ALERTSCRIPTPARAM` command to change an Alert script parameter value, but receives no acknowledgement from the ASWC, then it is unclear if the ASWC received and executed the command but the response to the client failed, or if the ASWC never received the command. At this point the client does not know if the ASWC has the original value or the new value and should issue a `GET ALERTSCRIPTPARAMS` command to determine which value the ASWC is using. If the original failed command is a GET command, the client should simply notify the user that the ASWC did not respond with the appropriate data within the specified amount of time; since the failed GET command would not change anything, there is no risk of the client and server being out of synchronization.

Connection Termination

Disconnection is handled by closing the TLS socket.

Message Overview

This section outlines the defined commands available to a client wishing to authenticate with the ASWC and get information (GET) from the ASWC or send information (PUT) to the ASWC and their designed purpose. Additional commands may be implemented and added to this list as the functionality of the ASWC is enhanced.

`AUTH` Commands:

- `AUTHINIT`: This command is issued by the client after connecting and successfully completing the TLS handshake, and is used to initiate authentication with the ASWC. Upon receipt of an `AuthChallenge` response from the ASWC, the `AUTH` command must be used to complete authentication.
  - Parameters: none
  - ASWC returns: <`AuthChallenge`> - indicates that the `AUTHINIT` command has been received and that authentication credentials are required.

- `AUTH`: This command is used to attempt to authenticate following authentication initiation via the `AUTHINIT` command.
  - Parameter: <`Username`>, <`Password`> - a username and password for system access.
  - ASWC returns: <`AuthStatusReply`> - indicates if authentication succeeded or failed and the level of privilege granted.

`GET` Commands:

- `GET CONTROLLERACTIVE`: This command is used to determine if the ASWC is currently running in normal (active) mode and sending commands to the information output field elements (CMS for instance). See the requirement for an "intermediate state" in which the ASWC is running, but will not place messages on signs, etc. (1.)
    - Parameters: none
    - ASWC returns: <`GetControllerActiveReply`> - indicates whether or not the ASWC is currently running in normal (active) mode and sending commands to the information output field elements (e.g. CMS), or <`ErrorReply`> if the command could not be executed.

- `GET SIMPLESTATUS`: This command is used to get a quick health check of the ASWC.
    - Parameters: none
    - ASWC returns: <`GetSimpleStatusReply`> - indicates `'OK'` if all modules and scripts finished successfully during the last run interval, `'ERROR'` if any module or script finished with an error status, or <`ErrorReply`> if the command could not be executed.

- `GET VERBOSESTATUS`: This command is used to get a detailed health check for all modules and scripts running on the ASWC. This command should be used to determine the active state of a module or script or to determine which module(s) or script(s) finished with an error in the case that the ASWC replies to a `GET SIMPLESTATUS` command with an error.
    - Parameters: none
    - ASWC returns: <`GetVerboseStatusReply`> - a list of modules and scripts, current active state (running or paused), status of last run interval, the time last ran, and the last run duration, or <`ErrorReply`> if the command could not be executed.

- `GET INPUTELEMENTS`: This command is used to get a list of all data collection (input) field elements monitored by the ASWC. This list can then be used to query the ASWC for sensor values from a given field element (see `GET INPUTELEMENTDATA` command).
    - Parameters: none
    - ASWC returns: <`GetInputElementsReply`> - a list of attached field elements, their types, and status, or <`ErrorReply`> if the command could not be executed.

- `GET INPUTELEMENTDATA`: This command is used to get a list of all the current sensor values for a given data collection field element.

    - Parameter: <`InputElementName`> `- the` field element name, which can be retrieved by using the `GET INPUTELEMENTS` command.
    - ASWC returns: <`GetInputElementDataReply`> - a list of all variables for the specified data collection (input) field element and their values or <`ErrorReply`> if the command could not be executed.

- `GET OUTPUTELEMENTS`: This command is used to get a list of all information output field elements (such as CMS or flashing beacon) attached to the ASWC. This list can then be used to identify a sign and subsequently query the ASWC for the message currently on that sign, or to change the message on the sign. (See the `GET OUTPUTELEMENTMSG` and `PUT OUTPUTELEMENTMSG` commands).

    - Parameters: none
    - ASWC returns: <`GetOutputElementsReply`> - a list of attached output field elements, their types and status, or <`ErrorReply`> if the command could not be executed.

- `GET OUTPUTELEMENTMSG`: This command is used to get the current message displayed on a given sign.

    - Parameter: <`OutputElementName`> - the output element name, which can be retrieved by using the `GET OUTPUTELEMENTS` command.
    - ASWC returns: <`GetOutputElementMsgReply`> - the sign type (e.g. FLASHINGBEACON or 170/500CMS) and the message currently on the sign (e.g. `'ON'` or `'OFF'` for FLASHINGBEACON or raw pixel data for 170/500CMS) or <`ErrorReply`> if the command could not be executed.

- `GET OUTPUTELEMENTMSGLIST`: This command is used to get the current message list (queue) pending for a given sign. Note that multiple messages may be pending for a sign if message, depending on message priorities.

    - Parameter: <`OutputElementName`> `- the output element` name, which can be retrieved using the `GET OUTPUTELEMENTS` command.
    - ASWC returns: <`GetOutputElementMsgListReply`> - all messages currently in the list (queue) to be applied to the specified output field element (sign) or <`ErrorReply`> if the command could not be executed.

- `GET LOG`: This command is used to get the list of entries in a given log file.
    - Parameters: <`LogName`> and (optional) <`NumLines`> - the log name and an optional specification of the number of lines.
    - ASWC returns: <`GetLogReply`> - a list of the lines from the (QC, CMS, or System) log file or <`ErrorReply`> if the command could not be executed.

- `GET ALERTSCRIPTSTATUS`: This command is used to get a list of all alert scripts and whether they are active, inactive, or in test mode.
    - Parameters: none
    - ASWC returns: <`GetAlertScriptStatusReply`> - a list of all alert scripts with a status for each script (active, inactive, or testmode), or <`ErrorReply`> if the command could not be executed.

- `GET ALERTSCRIPTPARAMS`: This command is used to get the parameter names and values for an alert script.
    - Parameter: <`AlertScriptName`> - the script name (from `GET ALERTSCRIPTSTATUS` command).
    - ASWC returns: <`GetAlertScriptParamsReply`> - all parameter values and types associated with the alert script, or <`ErrorReply`> if the command could not be executed.

PUT Commands:

- `PUT CONTROLLERACTIVE`: This command is used to pause (off) / resume (on) all output field element modules so the TMC can put a message on a sign. While inactive, the ASWC will continue to read from configured data collection field elements but will not change output field elements such as CMS or flashing beacon.
    - Parameters: <`OnOrOff`> - indicates whether output field element modules will be paused (off) or resume (on).
    - ASWC returns: <`PutControllerActiveReply`> - indicates whether the ASWC is currently `'ON'` (active) or `'OFF'` (inactive), or <`ErrorReply`> if the command could not be executed.

- `PUT OUTPUTELEMENTMSG`: This command is used to put a specified message on an information output field element, overriding any message currently on it.
    - Parameters: <`OutputElementName`>, <`OutputElementMsgType`>, <`OutputElementMsgPriority`>, <`OutputElementMsg`> - the field element name, the type of message, the message priority and the message text. Note that the type is needed to distinguish between different formats and devices.
    - ASWC returns: <`PutOutputElementMsgReply`> - indicates the old message and new message or <`ErrorReply`> if the command could not be executed.

- `PUT OUTPUTELEMENTNOTIFY`: This command is used to send notification that a message has been put on a sign by a source other than the ASWC (such as SOCCS CMS) and indicates whether the message can be overridden by the ASWC. This will be logged by the ASWC and used to keep track of the state of the sign.
    - Parameters: <`OutputElementName`>, <`OutputElementMsgType`>, <`OutputElementMsgPriority`>, <`OutputElementMsg`> - the field element name, the type of message, the message priority and the message text. Note that the type is needed to distinguish between different formats and devices.
    - ASWC returns: <`PutOutputElementNotifyReply`> - the message on the sign or <`ErrorReply`> if the command could not be executed.

- `PUT MODULESTATUS`: This command is used to place a field element module into an inactive state where it will not communicate with the associated field element. The module may still operate in a restricted fashion (e.g. information output modules may still delete expired messages from the associated message queue), but will not send data to or received data from the physical field element.
  - Parameters: <`ModuleName`>, <`OnOrOff`>- the name of the output field element module or input field element module of which the state should be changed and the state to change to the module to.
  - ASWC returns: <`PutModuleStatusReply`> - the old status and the new status of the module, or <`ErrorReply`> if the command could not be executed.

- `PUT LOG`: This command is used to place a message in the system log. The ASWC will also log the timestamp of when the message was received.

  - Parameters: <`LogName`>, <`LogMessage`> - the name of the log and the text of the message to be placed in the log.
  - ASWC returns: <`PutLogReply`> - Log name and log message just added to the ASWC's log or <`ErrorReply`> if the command could not be executed.

- `PUT ALERTSCRIPTSTATUS`: This command is used to change the status of an alert script. Valid states for alert scripts are `'ACTIVE'`, `'INACTIVE'`, and `'TESTMODE'`. When inactive, the alert script will not run. In test mode, scripts will run but won't commit a change to a sign or any other field element.

  Parameters: <`AlertScriptName`>, <`AlertScriptStatus`> - the alert script name and the new status (`'ACTIVE'`, `'INACTIVE'`, `'TESTMODE'`).

  - ASWC returns: <`PutAlertScriptStatusReply`> - script name, previous status, and current status, or <`ErrorReply`> if the command could not be executed.

- `PUT ALERTSCRIPTPARAM`: This command is used to set new a value for alert script parameter.

  - Parameters: <`AlertScriptName`>, <`AlertScriptParamName`>, <`AlertScriptParamValue`> - the alert script name, the parameter name, and the value
  - ASWC returns: <`PutAlertScriptParamReply`> - the alert script name, parameter name, the old value, and the new value or <`ErrorReply`> if the command could not be executed

Detailed Message Syntax Definition

The message syntax is described using Backus-Naur Form..

ASWC General Purpose Symbols

```
<Message> ::=
               <MessageLength> <MessageNumber> <Command> <Checksum>
            |  <MessageLength> <MessageNumber> <Response> <Checksum>
```

```
<MessageLength> ::= 16-bit unsigned integer in network byte (Big-
endian) order[1]
```

```
<MessageNumber> ::= 16-bit unsigned integer in network byte (Big-
endian) order[2]
```

```
<Checksum> ::= 16-bit integer in network byte order[3]

<LogName> ::= <Name>

<LogMessage> ::= <String>

<LogContents> ::=
               <emptystring>
            |  <String> <ff> <LogContents>


<InputElementName> ::= <Name>

<InputElementType> ::= <Name>[4]

<InputElementStatus> ::=
               'ACTIVE'
            |  'INACTIVE'

<InputElementList> ::=
               <emptystring>
            |  <InputElementName> <ff> <InputElementType> <ff>
                  <InputElementStatus> <ff> <InputElementList>

<InputElementVariableList> ::=
               <emptystring>
            |  <VariableName> <ff> <VariableValue> <ff>
                  <InputElementVariableList>

<OutputElementStatus> ::=
               'ACTIVE'

            |  'INACTIVE'
```

[1] The number of bytes in the message after MessageLength, including the 2-byte checksum
[2] An arbitrary number used for matching a server response with the original request. It should be unique for each message.
[3] The two least significant bytes of the sum of all bytes between MessageLength and Checksum, exclusive
[4] Type of input field element as defined by ASWC; e.g. "RWIS", "Loop", "MVDS", etc.

```
<OutputElement> ::=
               <OutputElementName> <ff> <OutputElementType> <ff>
                  <OutputElementStatus>

<OutputElementType> ::= <Name>[5]

<OutputElementName> ::= <Name>

<OutputElementMsg> ::=
               <FlashingBeaconMessage>
            |  <m170_500SignMsg>
            |  <m170_500SignMsgData>
            |  <additional message placeholder>[6]

<OutputElementMsgType> ::= <Name>[7]

<OutputElementMsgPriority> ::= <Integer>[8]

<OutputElementMsgList> ::=
               <emptystring>
            |  <OutputElementMsgType> <ff> <OutputElementMsgPriority>
                  <ff> <OutputElementMsg>[9] <ff> <OutputElementMsgList>

<OutputElementList> ::=
               <emptystring>
            |  <OutputElement> <ff> <OutputElementList>

<NewOutputElementMsg> ::= <OutputElementMsg>[10]

<OldOutputElementMsg> ::= <OutputElementMsg>[11]

<NewModuleStatus> ::= <ModuleStatus>

<OldModuleStatus> ::= <ModuleStatus>
```

[5] Type of output field element as defined by ASWC; e.g. "CMS", "FLASHINGBEACON", etc.
[6] The value of `OutputElementMsg` is parsed according to the value of `OutputElementMsgType`, other types may be added as Controller is expanded.
[7] Used to distinguish how to parse `OutputElementMsg`. May be values like OnOrOff, m170_500SignMsgData, or m170_500SignMsg.
[8] Priority is used to determine if the controller is allowed to overwrite the message with its own messages. At present, lesser integers denote lower priority and greater integers denote higher priority.
[9] The format of `OutputElementMsg` is dependent on the value of `OutputElementMsgType`; e.g. "ON" or "OFF" for FLASHINGBEACON, binary bulb data for m170_500SignMsgData, etc.
[10] See **Error! Bookmark not defined.**
[11] See **Error! Bookmark not defined.**

```
<AlertScriptName> ::= <Name>

<AlertScriptStatus> ::=

               'ACTIVE'
             | 'INACTIVE'
             | 'TESTMODE'

<AlertScriptParamName> ::= <Name>

<AlertScriptParamValue> ::= <String>

<AlertScriptParamType> ::= 'INT' | 'FLOAT' | 'STR' | 'MESSAGENAME'[12]

<AlertScriptParamList> ::=

               <emptystring>
             | <AlertScriptParamName> <ff> <AlertScriptParamValue> <ff>
                  <AlertScriptParamType> <ff> <AlertScriptParamList>

<OldAlertScriptParamValue> ::= <AlertScriptParamValue>

<NewAlertScriptParamValue> ::= <AlertScriptParamValue>

<AlertScriptAndStatusList> ::=

               <emptystring>
             | <AlertScriptName> <ff> <AlertScriptStatus> <ff>
                  <AlertScriptAndStatusList>

<OldAlertScriptStatusValue> ::= <AlertScriptStatus>

<NewAlertScriptStatusValue> ::= <AlertScriptStatus>


<ModuleName> ::= <Name>

<ModuleStatus> ::=

               <ModuleName> <ff> <RunStatus> <ff> <LastRunDateTime>
                  <ff> <SecondsToComplete>[13] <ff> <OkOrError>

<ModuleAndStatusList> ::=

               <emptystring>
             | <ModuleStatus> <ff> <ModuleAndStatusList>

<FlashingBeaconMessage> ::= <OnOrOff>

<NumLines> ::= <Integer>

<VariableName> ::= <Name>

<VariableValue> ::= <String>


<CurrentDateTime> ::= <DateTime>

<LastRunDateTime> ::= <DateTime>

<SecondsToComplete> ::= <String>

<OldControllerActiveValue> ::= <OnOrOff>
```

[12] Represents the type of the parameter, which is specified when the parameter is defined in an ASWC configuration file. It is provided for parameter verification on the client program.
[13] Amount of time taken by module or script to run

```
<NewControllerActiveValue> ::= <OnOrOff>
```

Common Symbols

```
<char> ::= any printable 7-bit ascii character
<upper_case_char> ::= [A-Z]
<lower_case_char> ::= [a-z]
<ff> ::= ASCII character 12: form feed
<underscore> ::= '_'
<emptystring> ::= ''[14]
<String> ::=
                <emptystring>
            |   <char> <String>
<DateTime> ::= date and time in YYYYMMDDHHmmss format
<Integer> ::=
                <emptystring>
            |   <digit> <Integer>
<digit> ::= [0123456789]
<NameChar> ::=
                <digit>
            |   <upper_case_char>
            |   <lower_case_char>
            |   <underscore>
<Name> ::=
                <NameChar>
            |   <NameChar> <Name>
<OnOrOff> ::=
                'ON'
            |   'OFF'

<OkOrError> ::=
                'OK'
            |   'ERROR'
<RunStatus> ::=
                'Running'
            |   'Paused'
```

[14] This represents a zero-length string.

CMS 170/500 Symbols

```
<m170_500SignMsg> ::=
               <m170_500DisplayTime> <ff> <m170_500MessageType> <ff>
                  <m170_500Page1Font> <ff> <m170_500Page2Font> <ff>
                  <m170_500Page1Line1> <ff> <m170_500Page1Line2>  <ff>
                  <m170_500Page1Line3> <ff> <m170_500Page2Line1> <ff>
                  <m170_500Page2Line2>  <ff> <m170_500Page2Line3>

<m170_500DisplayTime> ::= <Integer>[15]

<m170_500MessageType> ::=
               <m170_500normal>
             | <m170_500extended>
             | <m170_500flashing>
             | <m170_500blank>

<m170_500normal> ::= '0'

<m170_500flashing> ::= '1'

<m170_500extended> ::= '2'

<m170_500blank> ::= '8'

<m170_500Page1Font> ::=
               <m170_500singlestroke>
             | <m170_500doublestroke>

<m170_500Page2Font> ::=
               <m170_500singlestroke>
             | <m170_500doublestroke>

<m170_500singlestroke> ::= '1'

<m170_500doublestroke> ::= '2'

<m170_500Page1Line1> ::= <m170_500MessageLine>

<m170_500Page1Line2> ::= <m170_500MessageLine>

<m170_500Page1Line3> ::= <m170_500MessageLine>

<m170_500Page2Line1> ::= <m170_500MessageLine>

<m170_500Page2Line2> ::= <m170_500MessageLine>

<m170_500Page2Line3> ::= <m170_500MessageLine>

<m170_500MessageLine> ::=
               <emptystring>
             | <m170_500MessageChar> <m170_500MessageLine>

<m170_500MessageChar> ::= [ !\#$%&'()*+,-./0-9:;<=>?A-Z^_`~][16]
```

[15] Amount of time a flashing or multi page message will stay on, in 1/10ths of a second.
[16] These characters are from the CMS 170/500 protocol as implemented in ASWC and SOCCS CMS as implemented

```
<m170_500SignMsgData> ::= <raw_data_from_sign>
```
[17]

Client → ASWC Device Communication Symbols

```
<Command> ::=

               <AuthCommand>
            |  'GET' <ff> <GetCommand>
            |  'PUT' <ff> <PutCommand>

<AuthCommand> ::=

            |  'AUTH' <ff> <Username> <ff> <Password>
               'AUTHINIT'

<Username> ::= <String>

<Password> ::= <String>

<GetCommand> ::=

            |  'ALERTSCRIPSTATUS'
            |  'ALERTSCRIPTPARAMS' <AlertScriptName>
            |  'CONTROLLERACTIVE'
            |  'INPUTELEMENTS'
            |  'INPUTELEMENTDATA' <ff> <InputElementName>
            |  'LOG' <ff> <LogName>
            |  'LOG' <ff> <LogName> <ff> <NumLines>
            |  'OUTPUTELEMENTMSGLIST' <ff> <OutputElementName>
            |  'OUTPUTELEMENTMSG' <ff> <OutputElementName>
            |  'OUTPUTELEMENTS'
            |  'SIMPLESTATUS'
               'VERBOSESTATUS'

<PutCommand> ::=

               'ALERTSCRIPTSTATUS' <ff> <AlertScriptName> <ff>
                  <AlertScriptStatus>
            |  'ALERTSCRIPTPARAM' <ff> <AlertScriptName> <ff>
                  <AlertScriptParamName> <ff> <AlertScriptParamValue>
            |  'CONTROLLERACTIVE' <ff> <OnOrOff>
            |  'LOG' <ff> <LogName> <ff> <LogMessage>
            |  'OUTPUTELEMENTMSG' <ff> <OutputElementName> <ff>
                  <OutputElementMsgType> <ff>
                  <OutputElementMsgPriority> <ff> <OutputElementMsg>
            |  'OUTPUTELEMENTNOTIFY' <ff> <OutputElementName> <ff>
                  <OutputElementMsgType> <ff>
                  <OutputElementMsgPriority> <ff> <OutputElementMsg>

            | 'MODULESTATUS' <ff> <ModuleName> <ff> < OnOrOff>
```

[17] Some Information Output Elements return binary data that ASWC does not parse, in these instances ASWC just passes the binary data through to the client which may represent it in a way that is understandable to the operator.

ASWC Device → Client Communication Symbols

```
<Response> ::=

               <AuthChallenge>
             | <AuthStatusReply>
             | <GetAlertScriptParamsReply>
             | <GetAlertScriptStatusReply>
             | <GetControllerActiveReply>
             | <GetInputElementsReply>
             | <GetInputElementDataReply>
             | <GetLogReply>
             | <GetOutputElementsReply>
             | <GetOutputElementMsgReply>
             | <GetOutputElementMsgListReply>
             | <GetSimpleStatusReply>
             | <GetVerboseStatusReply>
             | <PutAlertScriptParamReply>
             | <PutAlertScriptStatusReply>
             | <PutControllerActiveReply>
             | <PutLogReply>
             | <PutOutputElementMsgReply>
             | <PutOutputElementNotifyReply>
             | <ErrorReply>

<AuthChallenge> ::= 'AUTHREQ'

<AuthStatusReply> ::=

               'AUTHOPERATOR'
             | 'AUTHSUPERVISOR'
             | 'AUTHFAIL'

<GetAlertScriptParamsReply> ::= <AlertScriptParamList>

<GetAlertScriptStatusReply> ::= <AlertScriptAndStatusList>

<GetControllerActiveReply> ::= <OnOrOff>

<GetInputElementsReply> ::= <InputElementList>

<GetInputElementDataReply> ::= <InputElementVariableList>

<GetLogReply> ::= <CurrentDateTime> <ff> <LogContents>

<GetOutputElementsReply > ::= <OutputElementList>

<GetOutputElementMsgReply> ::=

               <OutputElementMsgType> <ff> <OutputElementMsg>[18]

<GetOutputElementMsgListReply> ::= <OutputElementMsgList>

<GetSimpleStatusReply> ::= <OkOrError>

<GetVerboseStatusReply> ::=

               <CurrentDateTime> <ff> <ModuleAndStatusList>

<PutAlertScriptParamReply> ::=
```

[18] The format of `OutputElementMsg` is dependent on the value of `OutputElementMsgType`; e.g. "ON" or "OFF" for FLASHINGBEACON, binary bulb data for m170_500SignMsgData, etc.

```
               <AlertScriptName> <ff> <AlertScriptParamName> <ff>
                  <OldAlertScriptParamValue> <ff>
                  <NewAlertScriptParamValue>

<PutAlertScriptStatusReply> ::=

               <AlertScriptName> <ff> <OldAlertScriptStatusValue> <ff>
                  <NewAlertScriptStatusValue>

<PutControllerActiveReply> ::=

               <OldControllerActiveValue> <ff>
                  <NewControllerActiveValue>

<PutLogReply> ::= <LogName> <ff> <LogMessage>

<PutOutputElementMsgReply> ::=

               <OutputElementName> <ff> <OutputElementMsgType> <ff>
                  <OldOutputElementMsg> <ff> <NewOutputElementMsg>

<PutOutputElementNotifyReply> ::= <OutputElementName> <ff>
                <OutputElementMsg>

<PutModuleStatusReply> ::=

               <ModuleName> <ff> <OldModuleStatus> <ff>
                  <NewModuleStatus>

<ErrorReply> ::=

               'ERROR' <ff> <ErrorType> <ff> <ErrorReplyDetails>

<ErrorType> ::=

               'INVALIDCOMMAND'
            |  'INVALIDPARAM'
            |  'COMMANDEXECFAILED'

<ErrorReplyDetails> ::= <String>
```

Examples

Establishing a connection:

Upon successful establishment of a TLS connection:

Client sends:

Message Length	Message Number	Message Content	Checksum
Bytes 0-1	**Bytes 2-3**	**Bytes 4-11**	**Bytes 12-13**
0x00- 0x0c	0x00- 0x01	0x41- 0x55- 0x54- 0x48- 0x49- 0x4e- 0x49- 0x54 'A' 'U' 'T' 'H' 'I' 'N' 'I' 'T'	0x02- 0x67

ASWC responds:

Message Length	Message Number	Message Content	Checksum
Bytes 0-1	**Bytes 2-3**	**Bytes 4-10**	**Bytes 11-12**
0x00- 0x0b	0x00- 0x01	0x41- 0x55- 0x54- 0x48- 0x52- 0x45- 0x51 'A' 'U' 'T' 'H' 'R' 'E' 'Q'	0x02- 0x1b

Client sends:

Message Length	Message Number	Message Content	Checksum
Bytes 0-1	**Bytes 2-3**	**Bytes 4-18**	**Bytes 19-20**
0x00- 0x13	0x00- 0x02	0x41- 0x55- 0x54- 0x48- 0x0c- 0x75- 0x6E- 0x61- 0x6D- 0x65- 0x0c- 0x70- 0x73- 0x77- 0x64 'A' 'U' 'T' 'H' <ff> 'u' 'n' 'a' 'm' 'e' <ff> 'p' 's' 'w' 'd'	0x05- 0x20

ASWC responds:

Message Length	Message Number	Message Content	Checksum
Bytes 0-1	**Bytes 2-3**	**Bytes 4-9**	**Bytes 10-11**
0x00- 0x0a	0x00- 0x02	0x41- 0x55- 0x54- 0x48- 0x4f- 0x4b 'A' 'U' 'T' 'H' 'O' 'K'	0x01- 0xce

ASWC is now ready to accept commands.

Client requests a Simple status to get overall health of the ASWC.

Client sends:

Message Length	Message Number	Message Content	Checksum
Bytes 0-1	**Bytes 2-3**	**Bytes 4-19**	**Bytes 20-21**
0x00- 0x14	0x00- 0x03	0x47- 0x45- 0x54- 0x0c- 0x53- 0x49- 0x4d- 0x50- 0x4c- 0x45- 0x53- 0x54- 0x41- 0x54- 0x55- 0x53 'G' 'E' 'T' <ff> 'S' 'I' 'M' 'P' 'L' 'E' 'S' 'T' 'A' 'T' 'U' 'S'	0x04- 0x9d

ASWC responds:

Message Length	Message Number	Message Content	Checksum
Bytes 0-1	**Bytes 2-3**	**Bytes 4-5**	**Bytes 6-7**
0x00- 0x06	0x00- 0x03	0x4f- 0x4b 'O' 'K'	0x00- 0x9d

Client tells the ASWC that it put a message (“TEST MESSAGE”) on page 1 lines 1 and 2 of the sign named “CMSEAST”, which is a sign type of m170_500SignMsg, with a display time of 600 (tenths of a second), LOW priority, Normal message type, and Single Stroke font

Client sends:

Message Length	**Message Number**	**Message Content**	**Checksum**
Bytes 0-1	**Bytes 2-3**	**Bytes 4-81**	**Bytes 82-83**
`0x00- 0x52`	`0x00- 0x04`	`0x50- 0x55- 0x54- 0x0c- 0x4f- 0x55- 0x54- 0x50- 0x55- 0x54- 0x45- 0x4c- 0x45- 0x4d- 0x45- 0x4e- 0x54 0x4e- 0x4f- 0x54- 0x49- 0x46- 0x59- 0x0c- 0x43- 0x4d- 0x53- 0x45- 0x41- 0x53- 0x54- 0x0c- 0x6d- 0x31- 0x37- 0x30- 0x5f- 0x35- 0x30- 0x30- 0x53- 0x69- 0x67- 0x6e- 0x4d- 0x73- 0x67- 0x0c- 0x4c- 0x4f- 0x57- 0x0c- 0x36- 0x30- 0x30- 0x0c- 0x30- 0x0c- 0x31- 0x0c- 0x31- 0x0c- 0x54- 0x45- 0x53- 0x54- 0x0c- 0x4d- 0x45- 0x53- 0x53- 0x41- 0x47- 0x45- 0x0c- 0x0c- 0x0c- 0x0c 'P' 'U' 'T' <ff> 'O' 'U' 'T' 'P' 'U' 'T' 'E' 'L' 'E' 'M' 'E' 'N' 'T' 'N' 'O' 'T' 'I' 'F' 'Y' <ff> 'C' 'M' 'S' 'E' 'A' 'S' 'T' <ff> 'm' '1' '7' '0' '_' '5' '0' '0' 'S' 'i' 'g' 'n' 'M' 's' 'g' <ff> 'L' 'O' 'W' <ff> '6' '0' '0' <ff> '0' <ff> '1' <ff> '1' <ff> 'T' 'E' 'S' 'T' <ff> 'M' 'E' 'S' 'S' 'A' 'G' 'E' <ff> <ff> <ff> <ff>`	0x13- 0x99

ASWC responds:

Message Length	**Message Number**	**Message Content**	**Checksum**
Bytes 0-1	**Bytes 2-3**	**Bytes 4-37**	**Bytes 38-39**
`0x00- 0x26`	`0x00- 0x04`	`0x43- 0x4d- 0x53- 0x45- 0x41- 0x53- 0x54- 0x0c- 0x36- 0x30- 0x30- 0x0c- 0x30- 0x0c- 0x31- 0x0c- 0x31- 0x0c- 0x54- 0x45- 0x53- 0x54- 0x0c- 0x4d- 0x45- 0x53- 0x53- 0x41- 0x47- 0x45- 0x0c- 0x0c- 0x0c- 0x0c 'C' 'M' 'S' 'E' 'A' 'S' 'T' <ff> '6' '0' '0' <ff>`	`0x06- 0xF9`

		`'0'   <ff>   '1'  <ff>  '1'   <ff>`
		`'T'   'E'   'S'   'T'   <ff>  'M'`
		`'E'   'S'   'S'   'A'   'G'   'E'`
		`<ff>  <ff>  <ff>  <ff>`	

References

1. **Western Transportation Institute, September 2007,** *Automated Safety Warning Controller System Concept and Requirements Specification*.
2. **Western Transportation Institute, December 2009,** *Automated Safety Warning Controller Testing and Development Lab Summary and System Development Summary*
3. **Western Transportation Institute, September 2009,** *Controller Installation Guide*