# CTNET Field Protocol Specification

*November 19, 1997*

*DRAFT*

## Introduction

Version 1.0 of CTNET will support the AB3418 protocol for communication to field controllers. AB3418 is a point-to-point protocol which supports only master-to-local communication. AB3418 does not support central-to-master communication or central-master-local communication (message routing). Because AB3418 does not support central-to-master communication and because there is currently no other standard protocol for communicating between central and master this document defines a system for CTNET-to-master communications.

In addition to the problem of central-master communications, AB3418's message set is extremely limited. For example, yellow times, overlaps, peds, and phase calls are all missing from AB3418. In order for CTNET to display more than just green times, pattern number, and alarms the AB3418 message set will be extended. These "AB3418 Extended" (AB3418E) messages are defined in this document.

## Automatic Polling Sequence

In order to maximize the use of the limited field communications bandwidth, the master will continuously poll local controllers in a round-robin, preemptable fashion and send the local's response frame back to CTNET. Preemption is implemented by insertion of a message at the beginning of the master's outgoing message queue.

Define three (3) priority levels, with Priority #1 defined as highest priority:

- Priority #1 – Master to Local signal coordination messages.
- Priority #2 – CTNET to Master to Local (on-demand) messages.
- Priority #3 – GetStatus8 master polling cycle messages.

In the absence of priority 1 & 2 messages, the master will poll each local controller with an AB3418E GetStatus8 command/response message sequence. The resulting response from each local will be echoed to the central CTNET system. The polling will occur as often as possible given the existing master-local communication bandwidth limits.

CTNET can send any AB3418 or AB3418E message to the master at any time, which the master will forward to the local. The local's response will be forwarded back to CTNET. These CTNET on-demand messages have priority #2.

The master's signal coordination control routine can send coordination messages (SetTime, SetPattern, GetLongStatus8) at any time. These messages are priority #1, the highest priority. This insures the integrity & timeliness of master to local signal coordination.

The automatic polling sequence shall be enabled & disabled by DIP feature switch on the 170.

## AB3418E Message Definitions

All AB3418E messages use standard AB3418 framing. The definitions below cover only the protocol data unit (PDU) portion of a standard AB3418 message. More information on AB3418 framing can be found in *Standard Communications Protocol for Traffic Signals in California, Specification and Implementation Requirements*. (http://www.dot.ca.gov/hq/traffops/elecsys/ab3418/abhome.htm).

The following messages are currently defined for AB3418:

- GetControllerID
- SetTime
- SetPattern
- GetShortStatus
- GetSystemDetectorData

The following messages are defined in this document for AB3418E:

- GetStatus8 -- 8 phase operation; includes presence.
- GetLongStatus8 – 8 phase operation; includes presence, volume and occupancy for 8 system detectors.
- SetLoginAccess – Initiates login sequence.
- SetMasterPolling – Sets the polling mode of the Field Master
- SetMasterTrafficResponsive – Enables or disables the Traffic Responsive Routine in the Field Master.
- GetControllerTimingData – Gets Controller Timing Data in 32 byte increments
- SetControllerTimingData -- Sets the Controller Timing Data
- SetControllerTimingDataOffset – Sets the offset memory location for GetControllerTimingData message.
- GetStatus16 (future) – 16 phase operation; includes presence.

Message Type numbers for AB3418 & AB3418E are assigned as follows.

| Message | Request ID | Response ID | Error Response ID |
|---|---|---|---|
| AB3418 | | | |
| GetControllerID | 0x81 | 0xC1 | 0xE1 |
| SetTime | 0x92 | 0xD2 | 0xF2 |
| SetPattern | 0x93 | 0xD3 | 0xF3 |
| GetShortStatus | 0x84 | 0xC4 | 0xE4 |
| GetSystemDetectorData | 0x85 | 0xC5 | 0xE5 |
| AB3418E: | | | |
| GetStatus8 | 0x86 | 0xC6 | 0xE6 |
| SetLoginAccess | 0x97 | 0xD7 | 0xF7 |
| SetMasterPolling | 0x98 | 0xD8 | 0xF8 |
| GetControllerTimingData | 0x89 | 0xC9 | 0xE9 |
| SetControllerTimingData | 0x99 | 0xD9 | 0xF9 |
| GetStatus16 (future) | 0x8A | 0xCA | 0xEA |
| SetControllerTimingDataOffset | 0x9B | 0xDB | 0xFB |
| GetLongStatus8 | 0x8C | 0xCC | 0xEC |
| SetMasterTrafficResponsive | 0x9D | 0xDD | 0xFD |
| Unused Message Types: | | | |
| Reserved (Set) | 0x96 | 0xD6 | 0xF6 |
| Reserved (Get) | 0x87 | 0xC7 | 0xE7 |
| Reserved (Get) | 0x88 | 0xC8 | 0xE8 |
| Reserved (Set) | 0x9A | 0xDA | 0xFA |
| Reserved (Get) | 0x8B | 0xCB | 0xEB |
| Reserved (Set) | 0x9C | 0xDC | 0xFC |
| Reserved (Get) | 0x8D | 0xCD | 0xED |
| Reserved (Get) | 0x8E | 0xCE | 0xEE |
| Reserved (Set) | 0x9E | 0xDE | 0xFE |
| Reserved (Get) | 0x8F | 0xCF | 0xEF |
| Reserved (Set) | 0x9F | 0xDF | 0xFF |

All message PDU's are expressed in standard ANSI C notation for ease of readability. It should also be noted that the *BYTE* data type is defined as an *unsigned char*.

## GetStatus8 Message

```
struct GetStatus8Request
{
        BYTE    0x86;                   // Message Type - Request
};

struct GetStatus8Response
{
        BYTE    0xC6;                   // Message Type - Response
        BYTE    flags;                  // additional flags; Bit 0 ⇔ focus (default 0 - no focus).
                                        //  Bits 1-7 ⇔ reserved unused.
        BYTE    status;                 // format identical to AB3418 GetShortStatus “status” byte.
        BYTE    pattern;                // format identical to AB3418 GetShortStatus “pattern” byte.
        BYTE    green_yellow_overlap;   // Bits 0-3 ⇔ Green overlaps A-D.  Bits 4-7 ⇔ Yellow overlaps A-D.
        BYTE    preemption;             // Bits 0-3 ⇔ EV A-D.  Bits 4-5 ⇔ RR 1-2.  Bits 6-7 unused.
        BYTE    phase_call;             // format identical to AB3418 GetShortStatus “green” byte.
        BYTE    ped_call;               // format identical to AB3418 GetShortStatus “green” byte.
        BYTE    active_phase;           // Bits 0-7 ⇔Phases 1-8.  Bit set true for phase active.
        BYTE    interval;               // Bits 0-3 ⇔Ring 0 interval; Bits 4-7 ⇔Ring 1 interval.
                                        // Interval encoding is as follows:
                                        // 0x00 = Walk               0x01 = Don’t Walk        0x02 = Min Green
                                        // 0x03 = (Unused)           0x04 = Added Initial     0x05 = Passage - Resting
                                        // 0x06 = Max Gap            0x07 = Min Gap           0x08 = Red Rest
                                        // 0x09 = Preemption         0x0A = Stop Time         0x0B = Red Revert
                                        // 0x0C = Max Termination    0x0D = Gap Termination   0x0E = Force Off
                                        // 0x0F = Red Clearance
        BYTE    presence1;              // Bits 0-7 ⇔Detector 1-8.  Presence bits set true for positive presence.
        BYTE    presence2;              // Bits 0-7 ⇔Detector 9-16.
        BYTE    presence3;              // Bits 0-7 ⇔Detector 17-24.
        BYTE    presence4;              // Bits 0-3 ⇔Detector 25-28.  Bits 4-7 unused.
};

struct GetStatus8ErrorResponse
{
        BYTE    0xE6;                   // Message Type - Error Response
        BYTE    error;                  // Error number
        BYTE    index;                  // Index number
};
```

## GetLongStatus8 Message

```
struct GetLongStatus8Request
{
        BYTE    0x8C;                   // Message Type - Request
};

struct GetLongStatus8Response
{
        BYTE    0xCC;                   // Message Type - Response
        BYTE    flags;                  // additional flags; Bit 0 ⬄ focus (default 0 - no focus).
                                        //  Bits 1-7 ⬄ reserved unused.
        BYTE    status;                 // format identical to AB3418 GetShortStatus “status” byte.
        BYTE    pattern;                // format identical to AB3418 GetShortStatus “pattern” byte.
        BYTE    green_yellow_overlap;   // Bits 0-3 ⬄ Green overlaps A-D.  Bits 4-7 ⬄ Yellow overlaps A-D.
        BYTE    preemption;             // Bits 0-3 ⬄ EV A-D.  Bits 4-5 ⬄ RR 1-2.  Bits 6-7 unused.
        BYTE    phase_call;             // format identical to AB3418 GetShortStatus “green” byte.
        BYTE    ped_call;               // format identical to AB3418 GetShortStatus “green” byte.
        BYTE    active_phase;           // Bits 0-7 ⬄Phases 1-8.  Bit set true for phase active.
        BYTE    interval;               // Bits 0-3 ⬄Ring 0 interval; Bits 4-7 ⬄Ring 1 interval.
                                        // Interval encoding is as follows:
                                        // 0x00 = Walk              0x01 = Don’t Walk       0x02 = Min Green
                                        // 0x03 = (Unused)          0x04 = Added Initial    0x05 = Passage - Resting
                                        // 0x06 = Max Gap           0x07 = Min Gap          0x08 = Red Rest
                                        // 0x09 = Preemption        0x0A = Stop Time        0x0B = Red Revert
                                        // 0x0C = Max Termination   0x0D = Gap Termination  0x0E = Force Off
                                        // 0x0F = Red Clearance
        BYTE    presence1;              // Bits 0-7 ⬄Detector 1-8.  Presence bits set true for positive presence.
        BYTE    presence2;              // Bits 0-7 ⬄Detector 9-16.
        BYTE    presence3;              // Bits 0-7 ⬄Detector 17-24.
        BYTE    presence4;              // Bits 0-3 ⬄Detector 25-28.  Bits 4-7 unused.
        BYTE    sequence_number;        // sample sequence number
        BYTE    volume1;                // System detector 1
        BYTE    occupancy1;             // System detector 1, See AB3418 for description of the Occupancy Byte.
        BYTE    volume2;                // System detector 2
        BYTE    occupancy2;             // System detector 2
        BYTE    volume3;                // System detector 3
        BYTE    occupancy3;             // System detector 3
        BYTE    volume4;                // System detector 4
        BYTE    occupancy4;             // System detector 4
        BYTE    volume5;                // System detector 5
        BYTE    occupancy5;             // System detector 5
        BYTE    volume6;                // System detector 6
        BYTE    occupancy6;             // System detector 6
        BYTE    volume7;                // System detector 7
        BYTE    occupancy7;             // System detector 7
        BYTE    volume8;                // System detector 8
        BYTE    occupancy8;             // System detector 8
};

struct GetLongStatus8ErrorResponse
{
        BYTE    0xEC;                   // Message Type - Error Response
        BYTE    error;                  // Error number
        BYTE    index;                  // Index number
};
```

## GetStatus16 Message (future)

```
struct GetStatus16Request
{
        BYTE    0x8A;                   // Message Type - Request
};

struct GetStatus16Response
{
        BYTE    0xCA;                   // Message Type - Response
        BYTE    flags;                  // additional flags; Bit 0 ⇔ focus (default 0 - no focus).
                                        //  Bits 1-7 ⇔ reserved unused.
        BYTE    status;                 // format identical to AB3418 GetShortStatus “status” byte.
        BYTE    pattern;                // format identical to AB3418 GetShortStatus “pattern” byte.
        BYTE    green_overlaps;         // Bits 0-7 ⇔ Green overlaps A-H.
        BYTE    yellow_overlaps;        // Bits 0-7 ⇔ Yellow overlaps A-H.
        BYTE    preemption;             // Bits 0-3 ⇔ EV A-D.  Bits 4-5 ⇔ RR 1-2.  Bits 6-7 unused.
        BYTE    phase_callAB;           // Ring A & B;  format identical to AB3418 GetShortStatus “green” byte.
        BYTE    phase_callCD;           // Ring C & D;  format identical to AB3418 GetShortStatus “green” byte.
        BYTE    ped_callAB;             // Ring A & B;  format identical to AB3418 GetShortStatus “green” byte.
        BYTE    ped_callCD;             // Ring C & D;  format identical to AB3418 GetShortStatus “green” byte.
        BYTE    active_phaseAB;         // Bits 0-7 ⇔Phases 1-8.  Bit set true for phase active.
        BYTE    active_phaseCD;         // Bits 0-7 ⇔Phases 9-16.  Bit set true for phase active.
        BYTE    intervalAB;             // Bits 0-3 ⇔Ring A interval; Bits 4-7 ⇔Ring B interval.
        BYTE    intervalCD;             // Bits 0-3 ⇔Ring C interval; Bits 4-7 ⇔Ring D interval.
                                        // Interval encoding is as follows:
                                        // 0x00 = Walk              0x01 = Don’t Walk        0x02 = Min Green
                                        // 0x03 = (Unused)          0x04 = Added Initial     0x05 = Passage - Resting
                                        // 0x06 = Max Gap           0x07 = Min Gap           0x08 = Red Rest
                                        // 0x09 = Preemption        0x0A = Stop Time         0x0B = Red Revert
                                        // 0x0C = Max Termination   0x0D = Gap Termination  0x0E = Force Off
                                        // 0x0F = Red Clearance
        BYTE    presence1;              // Bits 0-7 ⇔Detector 1-8.  Presence bits set true for positive presence.
        BYTE    presence2;              // Bits 0-7 ⇔Detector 9-16.
        BYTE    presence3;              // Bits 0-7 ⇔Detector 17-24.
        BYTE    presence4;              // Bits 0-7 ⇔Detector 25-32.
};

struct GetStatus16ErrorResponse
{
        BYTE    0xEA;                   // Message Type - Error Response
        BYTE    error;                  // Error number
        BYTE    index;                  // Index number
};
```

```
};
```

## SetLoginAccess Message

This message establishes the Station ID of the caller. This ID can optionally be used by the master in determining which phone number to use during dial-back. This message initiates the connection sequence.

```
struct SetLoginAccessRequest
{
        BYTE    0x97;                   // Message Type -- Request
        BYTE    station_id;             // unique number identifying which phone number to call back
};

struct SetLoginAccessResponse
{
        BYTE    0xD7;                   // Message Type -- Response
};

struct SetLoginAccessErrorResponse
{
        BYTE    0xF7;                   // Message Type -- Error Response
        BYTE    error;                  // Error number
        BYTE    index;                  // Index number
};
```

## SetMasterPolling Message

This message toggles between full corridor mode (normal polling) and focus mode. To get detailed information on a single local controller, including accurate presence information, this message should be sent. This message will alter the master’s automatic polling routine from sequentially polling all locals to polling one local at a higher priority. The other locals will still be polled but at a lower rate. This will effectively boost the resolution of the presence information at the designated local by an order of magnitude (assuming 10 locals per master, the practical limit of the 170 master software using this protocol).

```
struct SetMasterPollingRequest
{
        BYTE    0x98;                   // Message Type -- Request
        BYTE    local_addr;             // address of local controller to poll at higher priority
        BYTE    timeout;                // number of minutes that local contrller will have higher priority.  Set to 0 for
                                        // immediate time-out.
        BYTE    msg_type;               // message type with which to poll single local
};

struct SetMasterPollingResponse
{
        BYTE    0xD8;                   // Message Type -- Response
};

struct SetMasterPollingErrorResponse
{
        BYTE    0xF8;                   // Message Type -- Error Response
        BYTE    error;                  // Error number
        BYTE    index;                  // Index number
};
```

## SetMasterTrafficResponsive Message

This message enables or disables the traffic responsive routine in the field master.

```
struct SetMasterTrafficResponsiveRequest
{
	BYTE	0x9D;			// Message Type -- Request
	BYTE	flags;			// Bit 0 ⬄  0 = Traffic Responsive Disabled;  1 = Traffic Responsive Enabled;
					// Bits 1 - 7 ⬄ reserved unused.
};

struct SetMasterTrafficResponsiveResponse
{
	BYTE	0xDD;			// Message Type -- Response
};

struct SetMasterTrafficResponsiveErrorResponse
{
	BYTE	0xFD;			// Message Type -- Error Response
	BYTE	error;			// Error number
	BYTE	index;			// Index number
};
```

## GetControllerTimingData Message

This message retrieves the local controller’s timing data in 32 byte increments.   The memory offset is set by calling the SetControllerTimingDataOffset message.

```
struct GetControllerTimingDataRequest
{
	BYTE	0x89;			// Message Type – Request
};

struct GetControllerTimingDataResponse
{
	BYTE	0xC9;			// Message Type -- Response
	short	offset;			// Cell address (0x0000 - 0xFFFF).  MSB = Page, LSB = Cell.
	BYTE	cell_contents0;		// Contents of cell at offset + 0 bytes
	…
	BYTE	cell_contents31;	// Contents of cell at offset + 31 bytes
};

struct GetControllerTimingDataErrorResponse
{
	BYTE	0xE9;			// Message Type -- Error Response
	BYTE	error;			// Error number
	BYTE	index;			// Index number
};
```

## SetControllerTimingData Message

This variable length message sets the local controller’s timing data.

```
struct SetControllerTimingDataRequest
{
        BYTE      0x99;                   // Message Type -- Request
        BYTE      number_of_cells;        // 1 - 16 max.  The number of cells that this message contains
        short     cell_address1;          // Cell address (0x0000 - 0xFFFF).  MSB = Page, LSB = Cell.
        BYTE      cell_contents1;         // Contents of cell at cell_address1
        …
        short     cell_addressN;          // Cell address (0x0000 - 0xFFFF).  MSB = Page, LSB = Cell.
        BYTE      cell_contentsN;         // Contents of cell at cell_addressN

};

struct SetControllerTimingDataResponse
{
        BYTE      0xD9;                   // Message Type -- Response
};

struct SetControllerTimingDataErrorResponse
{
        BYTE      0xF9;                   // Message Type -- Error Response
        BYTE      error;                  // Error number
        BYTE      index;                  // Index number
};
```

## SetControllerTimingDataOffset Message

This message sets the local controller’s timing data offset memory location.   This value is used when the GetControllerTimingData message is called.

```
struct SetControllerTimingDataOffsetRequest
{
        BYTE      0x9B;                   // Message Type -- Request
        short     offset;                 // Cell address (0x0000 - 0xFFFF). MSB = Page, LSB = Cell.
};

struct SetControllerTimingDataOffsetResponse
{
        BYTE      0xDB;                   // Message Type -- Response
};

struct SetControllerTimingDataErrorOffsetResponse
{
        BYTE      0xFB;                   // Message Type -- Error Response
        BYTE      error;                  // Error number
        BYTE      index;                  // Index number
};
```