UECU Surface Stimulation
Setup and Use

UECU commands/Amulet commands

- The Amulet command set supports a CRC UART protocol. One incoming and one outgoing command from this command set have been selected to implement the UECU command set.
- Output (Amulet to UECU)
  - The outgoing message is implemented using the "streamOut" command from the Amulet. This command simply sends out bytes without any manipulation. These messages simply use the UECU protocol (to be explained later in this document)
  - Example (Create Schedule):
    - Amulet:UART.streamOut(0x04+0x80+0x10+0x03+0xaa+0x00+0x64+0x59),
- Input (UECU to Amulet)
  - The UECU sends a message back which places the UECU message in a predefined location in the Amulet internal RAM. Currently, this location is 0xAA (170dec). This is the "Amulet Set Byte Variable Array" command. This command will automatically be processed by the Amulet, and the UECU message will be placed in memory.
  - The Amulet can read the memory, using the InternalRAM commands, where the value in parenthesis after "byte" is the predefined address:
    - Amulet:internalRAM.byte(170).value()

- UECU message structure:
  - Each message contains a UECU header, data and a checksum byte
  - UECU header:
    - Destination Address: 0x04 (fixed)
    - Source Address: 0x80 (fixed)
    - Message Type: see message commands
    - Length: Length of the message without the header or checksum

  - Checksum Calculation (one's complement)
    - Add up each of the bytes in the message
    - Mask out the lower byte of the sum and add the carry byte to it (ie: shift the upper byte right 8 bits and add it to the lower byte)
    - Invert the sum
    - Example:
      - Message: 0x04+0x80+0x10+0x03+0xaa+0x00+0x64
      - Checksum = 0x59

Channel Setup:

- The ChannelSetup command is used to setup each channel with the correct waveform shape and polarity. Also, the amplitude and pulse width limits for each channel are configured using this command.
    - 1 byte: PortChannel
        - The PortChannel parameter designates the port and channel that you are configuring.
        - The lower 4 bits are the channel and the upper 4 bits are the port
        - On the surface board, there is only port = 0
    - 1 byte: Amplitude Limit
        - Maximum amplitude that this channel will output.
    - 1 byte: Pulse Width Limit
        - Maximum pulse width that this channel will output.
    - 2 bytes: Interphase Delay in usec
        - The Interphase Delay range is 10-65535.
        - The interphase delay is the space between the phases of the waveform. Figure 1 shows a Interphase Delay of 100us (red arrow)

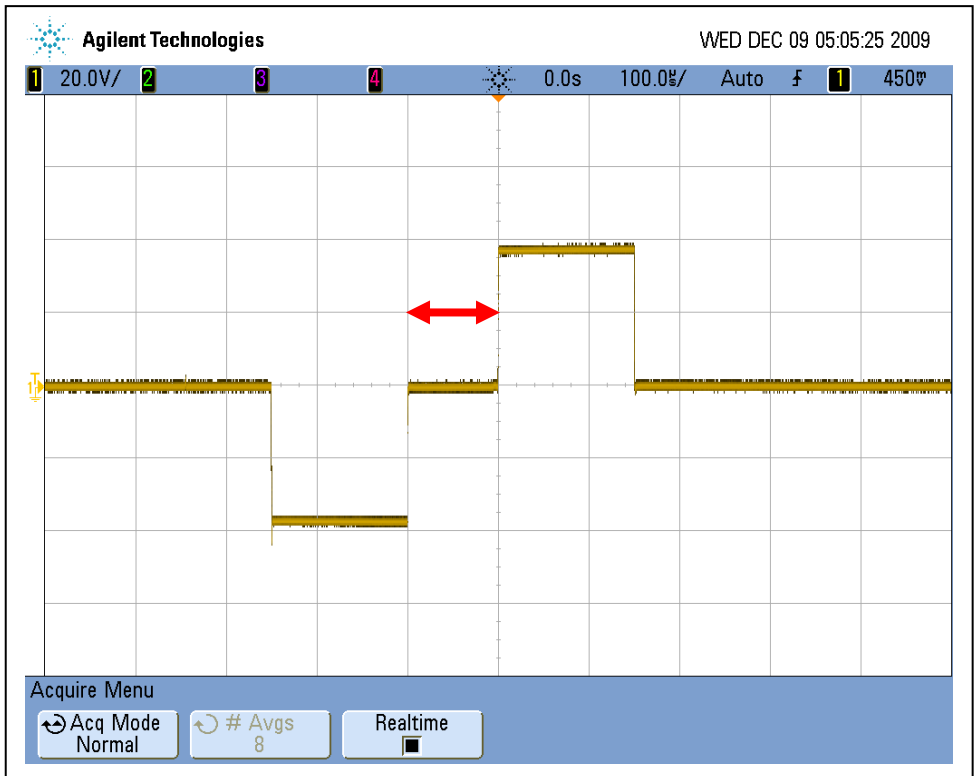

Figure 1: Interphase Delay is 100 usec, Aspect Ratio set to 0x11

    - 1 byte: AspectRatio

- The aspect ratio designates the proportion of the amplitude of the first phase to the second phase.
- The lower 4 bits represent the first phase and the upper 4 bits represent the second phase.
- Figure 1 (above) shows an aspect ratio of 0x11 (default)
- When the ratio is calculated, the phase that is represented by the smaller number in the aspect ratio does not change in amplitude or duration. The adjustments for the ratio are all applied to the phase represented by the larger number in the aspect ratio.
    - To cause the amplitude of the second phase (recharge phase) to be twice the amplitude of the first phase (stim phase), the Aspect Ratio is set to 0x21. In order to remain charge balanced, the pulse width of the second pulse (recharge phase) is automatically cut in half. (Figure 2)

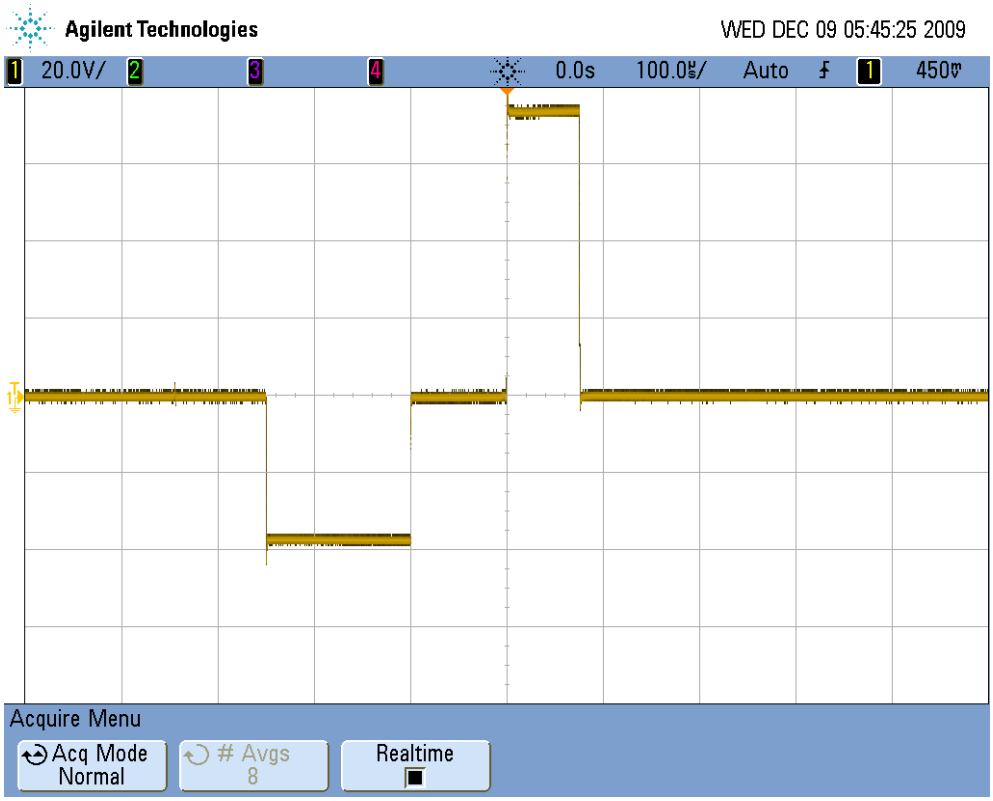

Figure 1: Interphase Delay is 100 usec, Aspect Ratio set to 0x21

- Any fraction can be selected by changing the aspect ratio as long as the numerator and the denominator are between 1 and 0xF (15 dec).
- When the ratio is calculated, the phase that is represented by the smaller number in the aspect ratio does not change in amplitude or duration. The adjustments for the ratio are all applied to the phase represented by the larger number in the aspect ratio. For an aspect ratio of 0x35, the recharge phase is represented by the smaller number (3) and therefore has an amplitude and duration equal to the amplitude and duration that is set in the event structure (40mA, 150usec). The stim phase has an amplitude equal to 5/3 times the recharge phase amplitude (5/3 * 40mA = 66.7mA) and a pulse width equal to 3/5 times the recharge phase pulse width (3/5 * 150usec = 90usec).

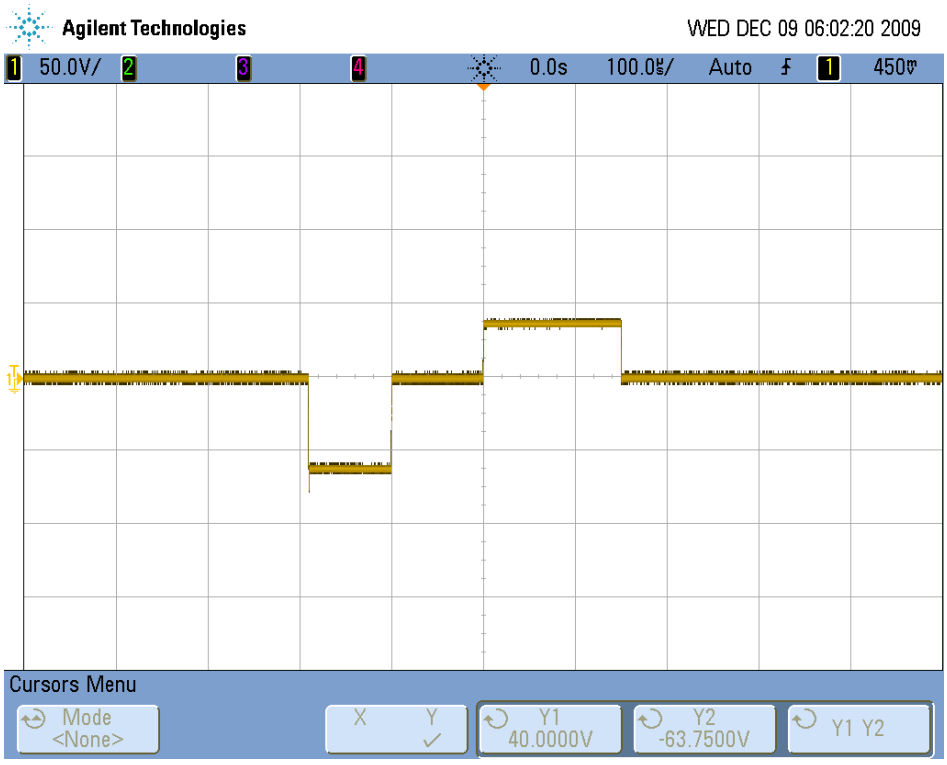

Figure 3: Interphase Delay is 100 usec, Aspect Ratio set to 0x35

- 1 byte: AnodeCathode
  - The lower 4 bits specify which physical channel is the cathode and the upper 4 bits specify which physical channel is the anode.
  - For 7 monopolar channels, the AnodeCathode setting for each channel would be: 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07
  - For 4 bipolar channels, the AnodeCathode setting for each channel would be: 0x01, 0x23, 0x45, 0x67

Schedule Setup:

- The UECU uses a quasi-operating system construct referred to as the “scheduler”. The scheduler controls timing of different input/output “events” that the board is capable of producing. Each event is assigned to a schedule and the schedule determines when the event occurs.
- Each schedule is specified by a “duration” and a “sync signal”. The duration tells the schedule how often it should repeat. If the duration is set to zero, the schedule runs once and then stops. The schedule is initiated when a sync command is received with a matching sync signal. The schedule is stopped, when a Halt command is received.
- Each event is created during setup to allow the most flexibility. The events have function specific parameters such as amplitude and pulse width for a stimulus event that are adjustable on the fly, using the change event parameter command
- In order for the events to run, they must be assigned to an existing schedule. They are assigned by specifying the schedule ID and a Delay from the start of the schedule. So for example:
    - Create a schedule with a Duration of 50 msec and a sync signal of 0xAA
    - Create a stimulation event on channel 1 and assign it to schedule 1 with a delay of 10 msec
    - Create a stimulation event on channel 2 and assign it to schedule 1 with a delay of 20 msec
    - Send a sync signal of 0xAA
    - The schedule will run every 50 msec, until a Halt command is received.
    - Stimulation will occur on channel 1 10 msec after the sync is received, then stimulation will occur on channel 2 10 msec after channel 1. This sequence will repeat every 50 msec.
- The schedule duration essentially controls the frequency of the events. The schedule duration can be changed on the fly to alter the stimulus frequency using the Change Schedule command. Care has to be taken to make sure that the duration is not set to a value that is lower that the highest delay for any events that are assigned to this schedule. An error message will be returned.

List of Message Codes:

// Message Codes

TRIGGER_SETUP_MSG	0x03
HALT_MSG	0x04
ERROR_REPORT_MSG	0x05
EVENT_ERROR_MSG	0x06
CREATE_SCHEDULE_MSG	0x10
CREATE_SCHEDULE_REPLY_MSG	0x11
DELETE_SCHEDULE_MSG	0x12
CHANGE_SCHEDULE_MSG	0x13
CHANGE_SCHEDULE_STATE_MSG	0x14
CREATE_EVENT_MSG	0x15
CREATE_EVENT_REPLY_MSG	0x16
DELETE_EVENT_MSG	0x17
CHANGE_EVENT_SCHED_MSG	0x18
CHANGE_EVENT_PARAMS_MSG	0x19
SYNC_MSG	0x1B
EVENT_COMMAND_MSG	0x1C
CHANNEL_SETUP_MSG	0x47
EVENT_COMMAND_REPLY	0x49

Event Parameters for Stimulus Event:

```
struct StimEvent
{
   unsigned char PulseWidth;        // pulse width in usec
   unsigned char Amplitude;         // amplitude in msec
   unsigned char Zone;              // unused
};
```

Message: DeleteScheduleMsg

Description: The DeleteSchedule message causes a UECU module to remove a schedule from its Scheduler and release all internal resources associated with that schedule and its events. A schedule may be removed when it is in any state, but the removal may not be immediate if the schedule is Active.

Size: 0x01

Structure:

```
struct DeleteScheduleMsg
{
   unsigned char ScheduleID;          // Schedule ID value
};
```

Message: ChangeScheduleMsg

Description: The ChangeSchedule message is sent to a UECU module to change the operating parameters of a previously-created schedule.

Structure:

```
struct ChangeScheduleMsg
{
   unsigned char ScheduleID;          // Schedule ID value
   unsigned char SyncSignal;          // Sync Signal Identifier
   unsigned int Duration;             // Duration of Schedule
};
```

Message: ChangeScheduleStateMsg

Description: The ChangeScheduleState message is sent to a UECU module to change the operating state of a previously-created schedule.

Size: 0x02

Structure:

```
struct ChangeScheduleStateMsg
{
   unsigned char ScheduleID;          // Schedule ID value
   unsigned char State;               // New state
};
```

Message: CreateEventMsg:

Description: CreateEvent is sent to a UECU module to add a new event to a schedule. Many events are possible. See the Events.h, for a list of events and their parameters.

The source address of the CreateEvent message is used as the initial owner of the event. In other words, the module that sends the CreateEvent message will be sent all messages related to the event, including data replies and error messages. The owner can be changed to any address in the system with the ChangeEventOwner message.

Size: 0x06 + event parameters

Structure:

```
struct CreateEventMsg
{
   unsigned char ScheduleID;          // Schedule ID value
   unsigned int  Delay;               // Delay from beginning of the schedule
   unsigned char Priority;            // Priority
   unsigned char EventType;           // Event type
   unsigned char PortChannel;         // port/channel
   unsigned char Params[1];           // Event Type specific parameters
};
```

Message: DeleteEventMsg

Description: The DeleteEvent message removes an event that was previously created.

Size: 0x01

Structure:

```
struct DeleteEventMsg
{
   unsigned char EventID;             // Event ID value
};
```

Message: ChangeEventSchedMsg:

Description: The ChangeEventSched message can move an event from one schedule to another or change the event's delay from the start of its schedule.

Size: 0x05

Structure:

```
struct ChangeEventSchedMsg
{
```

```
    unsigned char EventID;          // Event ID value
    unsigned char ScheduleID;       // Schedule ID value
    unsigned int  Delay;            // Delay from beginning of the schedule
    unsigned char Priority;         // Priority
};
```

Message: ChangeEventParamsMsg:

Description: The ChangeEventParams message is sent to a module to change the operating parameters for an event.

Size: 0x01 + event parameters

Structure:

```
struct ChangeEventParamsMsg
{
    unsigned char EventID;          // Event ID value
    unsigned char Params[1];        // Event Type specific parameters
};
```

Message: SyncMsg

Description: The Sync message causes schedules to start running. It is generally broadcast to all modules in a UECU system. All schedules with the specified SyncSignal will start running when the Sync message is received. The Sync message is ignored by any modules that have no schedules using the specified SyncSignal.

Certain SyncSignals can come from the UECU hardware as well as from this message. See the UECU Scheduler documentation for more information.

Size: 0x01

Structure:

```
struct SyncMsg
{
    unsigned char SyncSignal;       // Sync Signal value
};
```

Message: EventCommand

Description: The EventCommand message is used to cause something to happen immediately, without going through the rest of the schedule and event system. It is used when an application program or

another module in the UECU system wants to directly control the activity of a board, rather than setting up and controlling scheduled events.

The EventCommand operates as if it causes the creation of a short- lived event record within the module, which is deleted as soon as the event takes place. This event record is not part of a schedule, but is immediately placed in the scheduler priority queue for execution, just like what happens to a scheduled event when its time arrives. When the EventCommand makes it to the top of the priority queue, its event takes place. The event record is then deleted. It cannot automatically repeat as can an event that is part of a schedule.

The Priority field of the EventCommand determines whether it or scheduled events happen first, should an EventCommand arrive when an event is scheduled. If the EventCommand should take precedence over scheduled events, the value of its Priority field should be numerically less than the priorities of the scheduled events.

If only EventCommands are being used, and the program issuing the EventCommands is issuing them no faster than they can be executed by the module, the Priority of each command is inconsequential. If the program issues EventCommands faster than the module can process them, EventCommands of highest priority (lower magnitude) will be processed first, and EventCommands at the same priority will be processed in their order of arrival.

Size: 0x03 + event parameters

Structure:

```
struct EventCommandMsg
{
   unsigned char EventType;          // Event type
   unsigned char Priority;           // priority of this event
   unsigned char PortChannel;        // port/channel
   unsigned char Params[1];
};
```

Message: HaltMsg

Description: When a UECU module receives a Halt message, it stops all stimulus and data acquisition. The message may optionally reset the module's operating parameters to their power-on defaults.

Halt is often sent as a high-priority broadcast message, in order to stop or initialize all modules in the UECU. This is useful when initializing and when shutting down a UECU.

UECU modules receiving a Halt message will end their activities as quickly as practical. Events that were in progress when the Halt message is received may run to completion before the Halt is processed, but no new events will begin.

The Halt message may also be useful when handling error messages, since it can be used to put a single module or an entire UECU system into a known state.

Size: 0x01

Structure:

```
struct HaltMsg
{
  unsigned char HaltFlags;          // describes action on halt
};
```

// ChannelSetupMsg moved to messagecust.h

Message: TriggerSetupMsg

Description: TriggerSetup is used to configure the hardware trigger lines for input and output. TriggerSetup should be sent as a broadcast so that all modules are informed of the trigger line configuration at the same time and conflicts cannot occur. In the power-on reset condition, all trigger lines are used as inputs by all modules. This message must be used if some of the lines are to be used as outputs.

Size: 0x04

Structure:

```
struct TriggerSetupMsg
{
  unsigned char Output0Addr;        // Address of board using output 0 trigger
  unsigned char Output1Addr;        // Address of board using output 1 trigger
  unsigned char Output2Addr;        // Address of board using output 2 trigger
  unsigned char Output3Addr;        // Address of board using output 3 trigger

};
```

Message: CreateScheduleReply

Description: The CreateScheduleReply message is sent in response to the CreateSchedule message.

Size: 0x01

Structure:

```
struct CreateScheduleReply
{
  unsigned char ScheduleID;          // Schedule ID value
};
```

Message: CreateEventReply

Description: The CreateEventReply message is sent in response to a successful CreateEvent message.

Size: 0x04

Structure:

```
struct CreateEventReply
{
   unsigned char EventID;              // Event ID value
   unsigned char ScheduleID;           // Schedule ID value
   unsigned char EventType;            // Event type
   unsigned char PortChannel;           // port/channel
};
```

Message: ErrorReport

Description: The ErrorReport message is sent to indicate a problem processing a mesage. It is sent to the module that originated the erroneous message.

Size: 0x02

Structure:

```
struct ErrorReport
{
   unsigned char ErrorCode;            // Error code identifier--see Protocal documentation
   unsigned char FailedMessageType;    // message type of message that created this error
};
```

Message: EventError

Description: The EventError message is sent to indicate a problem processing an event. It is sent to the event's owner, which is usually the module that created the event. The owner can be changed with the* ChangeEventOwner message.

Size: 0x04

Structure:

```
struct EventError
{
   unsigned char ErrorCode;            // Error code identifier--see Protocal documentation
```

```
    unsigned char EventID;
    unsigned char EventType;
    unsigned char PortChannel;
};
```

Message: CreateScheduleMsg:

Description: The CreateSchedule message makes a new schedule. When this message is received, a UECU module allocates memory for the schedule and returns a CreateScheduleReply with a one-byte number, the schedule descriptor, that may be used to manipulate the new schedule.

Size: 0x03

Structure:

```
struct CreateScheduleMsg
{
    unsigned char SyncSignal;          // Sync Signal Identifier
    unsigned int Duration;             // Duration of Schedule
};
```

Message: ChannelSetupMsg:

Description: The Channel Setup message setups a channel on a stimulation module. The maximum pulse amplitude and pulse width are set using this message. Then the rest of parameters are different depending on the type of module. For the Surface module each channel must have the anode and cathode specified. This allows for different configurations of bipolar and/or monopolar operation. The Interphase Delay and the Aspect Ratio are used to define the shape of the waveform.

Size: 0x07

Structure (Surface Module):

```
struct ChannelSetupMsg
{
    unsigned char PortChannel;          // Port/Channel number
    unsigned char AmplitudeLimit;       // Amplitude Limit value
    unsigned char PulseWidthLimit;      // Pulse Width Limit value
    unsigned int  InterphaseDelay;      // interphase delay in usec
    unsigned char AspectRatio;          // ratio of phase 1 to phase 2,
                        // 0 means monophasic phase 1 only,  FF means monophasic phase 2 only
                        // Lower nibble is phase 1 and upper nibble is phase 2
    unsigned char AnodeCathode;          // Upper nibble is lead of anode and lower
                                         // nibble is lead of cathode
};
```

Example

- Setup 4 bipolar UECU channels with:
    - Amplitude Limit: 0x64
    - PulseWidthLimit: 0xFF
    - Interphase Delay: 0x000A
    - AspectRatio: Equal cathodic and anodic amplitude (0x11)

Amulet:UART.streamOut(0x04+0x80+0x47+0x07+0x00+0x64+0xff+0x00+0x0a+0x11+0x01+0xAC),
Amulet:UART.streamOut(0x04+0x80+0x47+0x07+0x00+0x64+0xff+0x00+0x0a+0x11+0x23+0x8A),
Amulet:UART.streamOut(0x04+0x80+0x47+0x07+0x00+0x64+0xff+0x00+0x0a+0x11+0x45+0x68),
Amulet:UART.streamOut(0x04+0x80+0x47+0x07+0x00+0x64+0xff+0x00+0x0a+0x11+0x67+0x46)

- Setup a Schedule with a sync signal of 0xAA that repeats every 0x0064 msec

Amulet:UART.streamOut(0x04+0x80+0x10+0x03+0xaa+0x00+0x64+0x59),

- A Create Schedule Reply message will be returned in Amulet memory starting at byte address 0xAA. This message contains the Schedule ID which will be used for the Create Event messages. Each byte can be read using the Amulet InternalRAM functions:

Amulet:internalRAM.byte(170).value()

- Setup an Event for each channel in the schedule with:
    - Schedule ID: 0x01 (from schedule create reply)
    - Events spaced 5 msec apart. Delays are set to: 0x00, 0x05, 0x0A, 0x0F
    - Priority: 0x00 (not implemented)
    - EventType: 0x03 , for Stimulus Event
    - Port Channel: 0x00, 0x01, 0x02, 0x03 for each channel
    - Pulse Width: 0x20 usec for each channel
    - Amplitude: 0x20 mA for each channel
    - Zone: 0x00 (not implemented)

Amulet:UART.streamOut(0x04+0x80+0x15+0x09+0x01+0x00+0x00+0x00+0x03+0x00+0x20+0x20+0x00+0x19),
Amulet:UART.streamOut(0x04+0x80+0x15+0x09+0x01+0x00+0x05+0x00+0x03+0x01+0x20+0x20+0x00+0x13),
Amulet:UART.streamOut(0x04+0x80+0x15+0x09+0x01+0x00+0x0a+0x00+0x03+0x02+0x20+0x20+0x00+0x0d),
Amulet:UART.streamOut(0x04+0x80+0x15+0x09+0x01+0x00+0x0f+0x00+0x03+0x03+0x20+0x20+0x00+0x07)

- A Create Event Reply message will be returned in Amulet memory starting at byte address 0xAA. This message contains the Event ID for each event which will be used in the Change Event Params command. Each byte can be read using the Amulet InternalRAM functions:

Amulet:internalRAM.byte(170).value()

- Start the Schedule using a Sync command with the sync signal specified in the Create Schedule Command (0xAA in this example). The Stimulus module will start to output pulses.

Amulet:UART.streamOut(0x04+0x80+0x1b+0x01+0xaa+0xb4)

- Halt the Schedule using a Halt command. The Stimulus module will halt output pulses.

Amulet:UART.streamOut(0x04+0x80+0x04+0x01+0x00+0x76)

- To change the pulse width or the amplitude of an existing event, use the Change Event Parameter Command.
    - Event ID is 1 for channel 1
    - PulseWidth is changed to 0x10 usec
    - Amplitude is changed to 0x10 mA

Amulet:UART.streamOut(0x04+0x80+0x19+0x04+0x01+0x10+0x10+0x00+0x3D)

- To change the duration (ie: frequency) of an existing schedule, use the Change Schedule Message
    - Schedule ID is 0x01 for schedule 1
    - Sync signal stays 0xAA
    - Duration is changed to 0x32 msec

Amulet:UART.streamOut(0x04+0x80+0x13+0x04+0x01+0xAA+0x0+0x32+0x86)