Title:	**Coapt BLE Communication Protocol**
Summary:	This document describes and specifies describes the design of data communication between the Generation 2 Complete Control Embedded Controller and a Client Application

Revision History

Version	Date	Comment	Author
0.1	2024-04-16	Initial Draft for basic comms support for Gen2 (firmware v1.34)	KC
1.0	2024-04-16	Reviewed and Released	BAL
1.01	2024-05-23	Correcting Heartbeat Exchange command	KC

Table of Contents

Introduction

This document specifies the Generation 2 Complete Control communication protocol, which defines all application-level and presentation-level data exchange between the Generation 2 Complete Control Embedded Controller (referred to in this document as *Gen2*, *device*, or *server*) and a remote Complete Control client application (*app*, or *client*). The term *client* here refers to a desktop or mobile application that serves as a graphical user interface to the Gen2 device. The term *user* refers to persons who are clinicians or prosthetic device users, and the term *operator* refers to persons who are part of the Gen 2 technical support team that may use the client application to perform tasks like device serialization and repair sessions.

The protocol specifies a set of data packets and real-time constraints on server-client inter-communication. The server/client terminology stems from the fact that the Gen2 device is a *server* of non-identifiable patient data to the desktop or mobile application, while the latter is a *client* that requests that data from Gen2 and controls Gen2's behavior through various commands. The client also serves as an interface to the Gen2 device for both the user and the operator. In step with the Open Systems Interconnection (OSI) model, the protocol defines, roughly, the *application layer* (layer 7) interface between the client and the server, as well as the *presentation layer* (layer 6).

Because no extensive multi-node network functionality is planned, the OSI Session and Transport layers (5 and 4) are not relevant.

Entities wrapped in < > brackets should be treated as placeholders for variables and values. For example, a series of values that appears as <A><B><C> in the document equates to 0x123456 when A=0x12, B=0x34, and C=0x56. In the same fashion, entities between `[]`, unless specified as arrays, are special ASCII characters.

For reference, the Gen2 Controller employs:

- RN4020 (Microchip) System on Chip (SoC) as its BLE chipset
  - Documentation: https://www.microchip.com/wwwproducts/en/RN4020
- STM32F407 (STMicroelectronics) microcontroller as its main processor
  - ARM Cortex M4 processor – little endian; this endianness is mirrored in the developed communication protocol
  - Uses a hardware CRC-32 MPEG-2 implementation. For development, check out:
    - https://gist.github.com/Miliox/b86b60b9755faf3bd7cf
    - https://crccalc.com/, with settings "Calc CRC-32", CRC-32/MPEG-2

High-Level System Overview

The embedded software of the Gen2 Controller is comprised of three active and independent programs: the bootloader, the main application, and the hardware test application. The execution path between the three is shown in Figure 1.

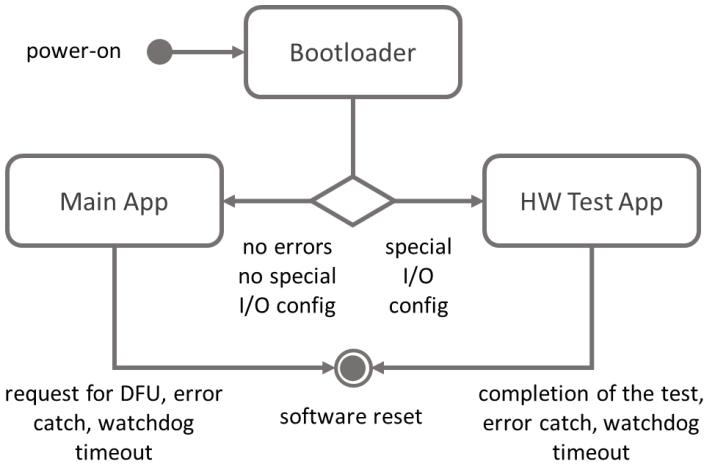

Figure 1: Gen2 Software System Design

Firmware Programs and Their Roles

On power on, the device executes a *bootloader* program, which takes care of the most essential system functionality. The bootloader is flashed only once into each controller, and it is responsible for vital procedures like one-time device serialization, direct over-the-air (OTA) device firmware updates (DFU), jumping to the main application, performing integrity checks on a variety of memory entities, and enabling over-the-air repair sessions in case the device reaches an error state. It is also responsible for carrying out firmware updates sourced from cached images of the main application. Finally, it acts as the last line of defense against system faults – it analyzes prior history of the system and takes actions based on the results.

In lieu of any errors or special requests, the bootloader turns execution over to the *main application.* This program performs the routine system functionality, and this is where the Gen2 Controller spends most of its time. The application performs routine tasks like calibration, data acquisition, pattern recognition, communication of data over Bluetooth Low Energy (BLE), and many others. Once the device reaches the application mode, it remains there until power-down or a special event like a request for immediate firmware update or a fatal error catch. The main application is updated wirelessly, as needed, using the DFU functionality of the bootloader.

The *hardware test application* is a small program accessible only via a specific I/O configuration on power-on. Like the bootloader, it will be flashed into memory of the device only once during its lifetime. Normally, this small program is inaccessible in the field and would only be used for in-house repair sessions. On power-on, the bootloader checks the electrical state of the device's external I/O lines and jumps to the test application instead of the main application.

System Modes

When communication is required with an external client, the Gen2 Controller may find itself in one of four distinct modes. Each mode enables certain features of the system that may or may not be accessible in other modes. All communication is performed over a BLE connection, and it is the responsibility of the client application to check the mode status of a connected controller at connection time. The four device modes are:

1. Serialization mode
2. Firmware Update mode
3. Application mode
4. Repair mode

The four modes are distinguished by slightly varied BLE profiles (see chapter on Bluetooth Low Energy). While most of the BLE Generic ATTribute Profile (GATT) entities are shared by all four modes, the custom RX/TX service universally unique identifier (UUID) is unique to each mode. As a rule, the Gen2 controller will force a BLE disconnect event when it switches into a new mode; hence, mode identification happens upon a connection or a re-connection event.

Serialization mode is assumed on power-on if the device has never been serialized or if the device's serial number is corrupt. None of the other modes are accessible until the device is properly serialized – given a unique ID that is used for the in-house product tracking system.

Firmware Update mode (also referred to as DFU mode) is accessible upon successful initial serialization, upon an explicit request during Application mode, and upon an explicit request during Repair mode. The DFU mode enables a direct download of new main application firmware over a BLE connection from the client. Once the download procedure and integrity checks are complete, the system can be reset wirelessly by the client application. After such reset or a power cycle event, the device will execute the new main application.

Application mode is what the end-user sees as the "normal functionality" of the system. In this mode, the device exchanges various types of data with the wireless client application, including EMG signal features, pattern recognition decisions, logging packets, and other data. Some of the Application mode commands can be used by the client to force an entry into the DFU or Repair modes.

Finally, the *Repair mode* is assumed when the system encounters repeated occurrences of fatal system errors. This mode can also be forced via the Application mode. The Repair mode features a minimal wireless BLE connection to the client with a different subset of features and commands not available in other modes. It can be used to start a new DFU procedure, to programmatically repair certain Non-Volatile Memory (NVM) entities, and perform other important actions.

Moreover, it is important to note that the Serialization, Firmware Update, and Repair modes are all maintained by the controller's *bootloader*, while the Application mode is maintained by the *main application*.

Bluetooth Low Energy

The primary physical mode of communication between the Gen2 device and the client is through BLE (Bluetooth v4.1 specification). The low-level networking associated with BLE is taken care of

by a system-on-a-chip module on the Gen2 device side and by a BLE dongle on the client side, so this document will not go into any detail on BLE networking. This document only focuses on the high-level GATT information required to implement communication between the client and the server. Regarding BLE device roles, the Gen2 controller assumes the role of a peripheral server device in a one-to-one network.

Phase	Startup	Connection	Data exchange
Role	Peripheral	Slave	Server

Custom Serial Emulation Service

The Gen2 system's BLE networking only uses one custom service, which includes two characteristics – one for outgoing data from the server (TX), and one for incoming data to the server (RX). The TX characteristic is used by the Gen2 controller to send data notifications to the client; the RX characteristic can be written to or written to without response by the client. The UUID of the custom service varies depending on the mode of the connected Gen2 controller (see chapter on High-Level Design Overview). The client's responsibility is to check the mode of the device using this UUID. A mode change is always prefaced with a disconnection event; hence, the client can resort to only checking the mode of the device upon a re-connection event.

Advertising Behavior

Once the device disconnects from the client to change into a different device mode, it automatically re-starts the advertising process. The client is responsible for re-connecting to the device in a timely manner. If the device switches its mode into the Application mode, the client must connect to the server within 5 minutes upon power up or the last BLE disconnection from the PC or mobile application, otherwise, the BLE module will be put into a sleep mode. To exit this sleep mode, there are two options: 1) a power cycle of the device or 2) the Gen2 Calibrate button can be pressed once and it will re-awake the BLE module. In all other modes (Serialization, DFU, Repair), the device advertises non-stop until a connection is made.

Server BLE Properties

Each device advertises using the following properties:

Service Name	`Generic Access`			
Service UUID	`00001800-0000-1000-8000-00805f9b34fb`			
Characteristics	Name	Type	Handle	Purpose
	`DeviceName`	`R,W`	`2`	Defines the device name. Value: `CCXYZT`, where `XYZT` stands for the last 4 digits of the device's serial number.
	`Appearance`	`R`	`4`	Defines what type of device the Gen2 server is recognized as. Value: *undefined*
	`PeripheralPreferredConnection Parameters`	`R`	`6`	Shows the connection params needed by the server. Value: `0A-00-10-00-64-00-E2-04`
Service Name	`Generic Attribute`			
Service UUID	`00001801-0000-1000-8000-00805f9b34fb`			
Characteristics	Name	Type	Handle	Purpose
	`ServiceChanged`	`I`	`9`	Indicates to client changes in services
Service Name	`Device Information`			
Service UUID	`0000180a-0000-1000-8000-00805f9b34fb`			
Characteristics	Name	Type	Handle	Purpose
	`SerialNumberString`	`R`	`13`	Defines the MAC address of the Bluetooth SoC. Value: 00:1E:ab:cd:ef:gh, where a, b, c, d, e, f, g, and h are device-specific characters
	`HardareRevisionString`	`R`	`15`	Hardware version of the Bluetooth SoC used on the Gen2 device.
	`FirmwareRevisionString`	`R`	`17`	Firmware version of the Bluetooth SoC used on the Gen2 device.
	`SoftwareRevisionString`	`R`	`19`	Software version of the Bluetooth SoC used on the Gen2 device.
	`ManufacturerNameString`	`R`	`21`	Manufacturer Name. Value: `Microchip`
	`ModelNumberString`	`R`	`23`	Number of the SoC model. Value: `RN4020`
Custom Service UUID	`bd505a55-c892-4a2d-9fd0-4ed48997e555`			(Application Mode)
Characteristics	Name	Type	Handle	Purpose
	`799846a2-44c5-44ca-b620-41a48ac4459c`	`R,N`	`26`	TX: Outgoing data from server
	`d6b87f3a-2905-463f-8e5a-40d3dce8c186`	`W,WN`	`29`	RX: Incoming data to server

Connection Behaviors

The Client UI should enable easy and intuitive BLE connection mechanism to each user's device. Only one device may be connected at a time. The following table summarizes the behaviors the client application needs to implement to handle device connections. Examples and control names are approximations.

Condition	UI Behavior
Fresh start or none of the previously connected devices are present	(1) On clicking "Search for devices" control, the app shows a list of nearby/available Gen2 devices. Clicking on a device initiates connection with it. On connection, the app remembers the device in cache.
One or multiple devices are already known and only one of them is present	(2) Perform connection in the background. When the app is opened, it should, within a reasonably shot time, connect to the advertising device on its own but notify the user that the connection was made. Also, (1) applies as well.

Upon unexpectedly receiving a disconnect event from the device, the client must attempt to re-connect to the same device for 10 seconds. During this time, the user should be notified of this effort unobtrusively – the data stream will stop, but since the connection may be intermittent, the flow should not be disturbed. After that time elapses, the user should be notified that the device was lost and suggest searching for new devices.

When searching for devices and/or connecting, the client's first responsibility is to check the mode of the device by checking the device's RX/TX service UUID. This will determine further UI actions for the user – whether the device is in Application, Serialization (for Coapt operators), Firmware Update, or Repair mode.

In Application mode, the device advertises for 5 minutes after a power on event and after an unexpected loss of connection with the client. Once this time elapses, the device shuts off its BLE chipset. When this happens, the client needs to power cycle the device or quickly press the calibrate button once to awaken the BLE chipset in order to initiate a new connection.

In all other modes, advertising is done without a time limit.

Presentation Layer

Typically, the communication between Complete Control servers and clients is one-to-one – one client (App) communicates with one server (Gen2 Controller). At the presentation layer, packets are routed based on a type-specific prefix byte and are formatted in the following fashion:

`<id><packet>`

The following identifiers are recognized as belonging to different *data channels*. Relevant channels to establish basic connection and streaming of EMG feature data and classifier end-point decisions are highlighted below:

ID	Data Channel	Description	System Mode	Direction
0x01	Heartbeat	Exchanged between client and server for robust connection failure handling	Application	All
0x02	Reserved	-	Serialization, DFU, Repair	
0x03	Application Command Interface	Command interface between client and server when server is in application mode	Application	
0x04	EMG Signal Features	Pushed to the client by the server when new EMG feature data is available	Application	Server to Client only
0x05	End-Point Command	Pushed to the client by the server when new control decisions are made by the server	Application	
0x06	Reserved	-	Application	
0x07	Reserved	-	Application	
0x08	Real Time Signals Check Status	Pushed to the client by the server when real time signals check evaluates EMG input	Application	
0x09	Reserved	-	-	-
0x0a	Reserved	-	Application	Server to Client only
0x0b	Reserved	-	-	-
0x0c	Reserved	-	Application	Server to Client only

Channels 0x01-0x03 contain commands sent by the client and responded to by the server, unless explicitly stated. Packets of type 0x01 are exchanged between the two entities for a robust disconnection-handling mechanism. Types 0x02 and 0x03 are command/response interface channels. Channels 0x04 and onwards are categorized as Real Time Data channels and are streamed strictly from server to client as data becomes available without explicit requests. Data is sent over Real Time Data channels only when the server is in Application mode. When the

server is in one of the three modes of Serialization, Firmware Update, or Repair, only the Bootloader Command Interface (0x02) interface is available, and 0x03 requests are denied. In application mode, all channels except 0x02 are active.

Application Layer

Heartbeat Exchange (0x01)

To establish robust connection handling between the server and the client during *Application mode only*, the protocol requires the two entities to exchange a heartbeat packet at regular time intervals. If either entity stops issuing/responding to heartbeats within the specified time limits, the other may assume that the former has entered an unresponsive state and must try to disconnect and continuously re-initiate a connection with the faulting side until a successful connection (and issuance of heartbeats from both sides) is established.

The client sends out the heartbeat packet of the following construction:

<type><id><nzdata:3><end>		
Description	Provides the client with commands to drive a virtual prosthetic device or to monitor output data	
Parameters	**type**	0x01
	id	"Unique" ID (0x00-0xFF) that must be incremented by the client with each new heartbeat packet. The same ID must be present in the server's heartbeat return.
	nzdata	Three bytes of non-zero data. Set to 0xFF.
	end	0x0A
Info	Server must see a new ID from the client every 2 seconds, and client must see that same ID from the server within 2 seconds of the original issued heartbeat.	

In direct response to the client's heartbeat, the server sends out:

<type><id><chkeng:2>		
Description	Provides the client with commands to drive a virtual prosthetic device or to monitor output data	
Parameters	**type**	0x01
	id	"Unique" ID (0x00-0xFF) that must be incremented by the client with each new heartbeat packet. The same ID must be present in the server's heartbeat return.
	chkeng	16-bit bitmask signifying the status of the device. See below for decoding info
Info	Server must see a new ID from the client every 2 seconds, and client must see that same ID from the server within 2 seconds of the original issued heartbeat.	

The purpose of this is to make sure the Gen2 controller knows when to break a connection with a BLE chipset that is no longer directly controlled by a client application. The Heartbeat Exchange only needs to occur while the device is in the Application mode. In this mode, the device puts its BLE SoC into deep sleep mode when a connection is not established within a 5-minute window. However, the Heartbeat Exchange is not necessary when the controller is in other modes because the device always advertises in those modes.

The Check Engine bitmask (**chkeng** above) specifies the current status of the device:

Bit	Description	Notes	Device UI	Client UI
0	Reserved	---	---	---
1	BLE Advertising	Set when the device is advertising. Reset when advertisement time runs out (~30s after power on and disconnect), or on connection event.	Solid white	No direct effect*
2	BLE Connected	Set when the device is connected to client via Bluetooth. Reset when the device is not connected to client.	Solid blue	No effect
3	Recording Calibration	Set when the calibration sequence records EMG data. Reset at all other times.	Solid green	No direct effect**
4	Warning: Leadoff Detected	Set when the system detects that one or more EMG leads lift off the patient's skin. Reset as soon as none of the EMG leads show lift off behavior	Solid yellow	Solid yellow
5	Warning: Poor Calibration Quality	Set at the end of calibration if the system determines that the quality of calibration data is poor. Reset upon a better new calibration during the same session or on next power-on	Solid yellow	Solid yellow
6	Warning: No Button UI Present	Set when the system detects the absence of the calibrate button. Reset only on next power-on.	Solid yellow	Solid yellow
7	Null Classification	Set when the system is forced into a "Null Classifier" mode, in which the system has no classes to classify and does not output any signals to terminal devices. Reset on successful calibration and the presence of a valid classification model	Flashing yellow	Flashing yellow
8	Critical Failure	Set when the system experiences a critical failure	S/F Red or None	Flashing Red
9+	Reserved	---	---	---

* Device will not be connected to the client, and the client should turn the check engine light grey/white
** Throughout calibration, client should turn its check engine light to Solid Green

If none of bits 4+ are set, the check engine light in the UI should be solid green, with no textual warnings. If some of the bits 4+ are set, the check engine light should assume the designated behavior. The priority of the warnings and states are as follows, from highest to lowest:

1. Critical Failure
2. Calibration Recording
3. Null Classifier
4. Calibration Quality
5. Leadoff Detected
6. No Button UI Present

Application Command Interface (0x03)

When the server is in the normal operation mode, the client can request certain actions from the server using the Application Command Interface (ACI). It features multiple simple ASCII Command Line Interface (CLI) client commands and server responses. The foundational structure of the ACI command and response packets stems from the design of the FreeRTOS+CLI framework.

The following commands are accepted by Gen2. All the following commands have immediate responses, and special responses are available for some commands. Commands must be only issued upon a return of the command prompt from the server.

Command and Response Packet Format

All CLI packets are ASCII encoded. All packets are prefaced with special signifiers specifying the type of command or response issued. The preface is followed by an appropriate number of parameters, depending on the specifier. All entered entities have to be space delimited. Each packet must be terminated with a line feed character (`[\n]`). Example of valid command:

“Comma-delimited series of ASCII characters” representation:

```
M,I,D,[space],0,1,7,[space],K,[\n]
```

CLI representation:

```
MID[space]017[space]K[line feed]
```

Meaning:

> “Get the enabled state for motion class with ID 17.”

Prompts

All ACI responses end with special prompts, signifying whether the server next expects a command or a data packet. The following table specifies these packet suffixes:

`>>[line feed]`	
Description	Command prompt (***CP***). Returned from server when server is ready to accept a new command.
`>[line feed]`	
Description	Data prompt (***DP***). Returned from server when server is ready to accept a new data packet.

Command/Response Procedure

Client should wait for one of the above prompts from the server before issuing any new data or commands. The client must issue new packets in agreement with the type of prompt the server responded with.

Most commands return a command prompt, after which the client can issue a new command. If a non-command packet is issued when a command is expected, server ignores the new packet and returns the error response **ERR**.

ACI Command Reference

The following packets are each either a command or a response to a command, as noted by their table. Command packets are sent strictly from the client to the server and request certain actions to be undertaken by the server. The respective responses are specified in the next section. Response packets are sent strictly from the server to the client in response to command packets specified above.

Profile Ready Command (PR)

PR	
Role	Command
Description	Request if device has initialized profile. Must be polled on connection to get initialization state before querying other commands.
Parameters	None

PR <status> <prompt>			
Role	Response to …		
Description	Respond with device profile initialization status		
Parameters	**status**		"OK" if device is ready and has initialized profile otherwise responses is "NR" for not ready
	prompt	***CP***	Always

Get Version Information (VN)

VN	
Role	Command
Description	Request bootloader and firmware version number
Parameters	None

VN <btvn> <fwvn> <prompt>			
Role	Response to …		
Description	Respond with bootloader and firmware version number		
Parameters	**btvn**		Version number for the bootloader, represented as the major revision number and the minor revision number separated by a ".". Each version number will be a decimal integer no larger than 2 digits (e.g. "3.5" or "0.11")
	fwvn		Version number for the application firmware, represented as the major revision number and the minor revision number separated by a ".". Each version number will be a decimal integer no larger than 2 digits (e.g. "3.5" or "0.11")
	prompt	***CP***	Always

Get Handedness (HND)

HND <state>			
Role	Command		
Description	Gets the device's prothesis side as left- or right-handed		
Parameters	**state**	**K**	Query the side of the prosthesis.

HND <state> <prompt>			
Role	Response to **HND <state>**		
Description	Respond with handedness state after the command has been executed		
Parameters	**state**	**L**	Device is behaving as left-handed.
		R	Device is behaving as right-handed.
	prompt	***CP***	Always

Get EMG Channel Status (EMG)

EMG <statemask>		
Role	Command	
Description	Gets the bitmask of EMG channels, which are configured on or off	
Parameters	**statemask**	An eight-character field with each character representing the state of one EMG channel in order with the first character representing the grey channel. "KKKKKKKK" should be used to query the current state of each EMG channel.

EMG <statemask> <prompt>			
Role	Response to **EMG <statemask>**		
Description	Respond with the state of the EMG channels after the command has been executed		
Parameters	**statemask**	A two-character field representing a hexadecimal representation of a bitmask, with each bit representing the state of one EMG channel. "80" (0x3830) would represent only the first of 8 channels being on, "04" (0x3034) would represent only the 6th channel being on, and "FF" (0x4646) would represent all channels being on.	
	prompt	***CP***	Always

Get Motion Status (MID)

MID <id> <state>			
Role	Command		
Description	Query a motion class's current state (enabled or disabled)		
Parameters	**id**	Specific decimal motion ID (0 – 128), represented using three ASCII characters, right-aligned and zero-padded from the left. If a motion ID is decimal 97, this parameter would be entered as a character array **097**, consisting of *characters* **0** (0x30), **9** (0x39), and **7** (0x37).	
	state	**K**	Query the current motion class state.

MID <id> <state> <prompt>			
Role	Response to **MID <id> <state>**		
Description	Respond with the state of the motion class after the command has been executed		
Parameters	**id**	Specific decimal motion ID (0 – 128), represented using three ASCII characters, right-aligned and zero-padded from the left. If a motion ID is decimal 97, this parameter would be entered as a character array **097**, consisting of *characters* **0** (0x30), **9** (0x39), and **7** (0x37).	
	state	**E**	Motion class is enabled
		D	Motion class is disabled
		I	This class is invalid given the current device profile. This class cannot be enabled.
	prompt	***CP***	Always

Error Response (ERR)

ERR <code> <prompt>			
Role	Response to any command		
Description	Error in command execution		
Parameters	**code**	**ILL**	Illegal or unknown command
		PAR	Parameters are in an invalid format
		NEF	Command did not take effect
	prompt	***CP***	Always

Real Time Device Data (0x04 - 0x0c)

Whereas the ACI protocol and BCI protocol entail bi-directional data exchanges, real-time device data like EMG features and device output commands are pushed unidirectionally to the client by the server as new data becomes available. The real time device data packet types include:

- EMG Signal Features
- End-Point Commands
- Real Time Signals Check Updates

EMG Signal Features (0x04)

The server gathers EMG data from the patient's able musculature via multiple front-end EMG channels. The gathered data is used to extract certain mathematical features from the signals; these features, in turn, allow the server to make motion class decisions based on a trained classifier data set. Instead of sending out raw EMG data to the client, the server sends out only Mean Relative Values (MRVs) of all enabled EMG channels. MRVs are used for plotting the general user's effort seen at each EMG front-end channel. The packet is constructed as follows:

<type><feat0><feat1><feat2><feat3><feat4><feat5><feat6><feat7>		
Description	Provides the client with commands to drive a virtual prosthetic device or to monitor output data	
Parameters	**type**	0x04
	featN	16-bit Mean Relative Value of EMG channel **N**. Zero if channel is disabled via ACI
Info	Issued by the server to the client only if `emgStream` key in Signals block is set to **1**.	

End-Point Commands (0x05)

The motion class decisions made by the classification algorithm are translated into physical or virtual movement of one or multiple end-point prosthetic device(s) using End-Point Commands. The commands communicate motion and speed requests to the end-point devices. Whereas the client can communicate the same information through ACI in the clinician controls mode, the server uses a separate packet type to communicate output information to virtual arm within the client application. The End-Point Commands are constructed as follows:

<type><motion><speed>		
Description	Provides the client with commands to drive a virtual prosthetic device or to monitor output data	
Parameters	**type**	0x05
	motion	Hexadecimal motion ID (0x00-0x7F)
	speed	Motor speed (linearly represented by a hex value of 0x00-0xFE, where 0x00 is no movement, and 0xFE is full speed)
Info	Issued by the server to the client only if `outStream` key in Signals block is set to **1** OR `outputDir` key in Output block is set to **3** (Virtual Arm)	

Real Time Signals Check (Leadoff) Packets (0x08)

The server sends a current through the EMG electrodes, checking for a circuit completed across the skin of the user. If the circuit is completed, the electrodes have good contact to the skin and may gather good EMG data. When the circuit is not complete, an indicator is displayed to inform the user that the electrodes are not reliably in contact with their skin. The packet is constructed as follows:

<type><chan0><chan1><chan2><chan3><chan4><chan5><chan6><chan7>		
Description	Provides the client with commands to drive a virtual prosthetic device or to monitor output data	
Parameters	**type**	0x08
	chanN	16-bit value set to 0x00 if the electrodes are in place or 0x01 if they are not
Info	Issued by the server to the client only if `emgStream` key in Signals block is set to **1**.	