



# iTach API Specification

## Version 1.4

### 1. The iTach Modular Design Concept

The iTach family of products' modular design provides a variety of capabilities. Each module provides a particular function: power and network connections, infrared (IR), digital input (RS232), and contact closures (relay). A module may also support one or more connectors of the same type. For example, an IR module has three independent IR connectors; whereas, a serial module has only one serial RS232 connector. This is due to the fact that the number of connectors a module can support is dictated by its 1.5 inch physical width.

It is important to understand that a module's address is determined solely by its physical position within the iTach enclosure. Each module occupies 1.5 inches of front panel space, even if it is part of a larger product. At power on, module addresses are assigned starting with "0" for the left-most module (containing the network and power connectors) and increasing sequentially to the right until all module addresses are assigned (see figure). This presents a consistent programming interface as additional modules are added.

A connector's address is its position within a module, starting at 1 on the left, and increasing sequentially as you move to the right. A complete connector address includes the module address and the connector location in the module, separated by a colon. In figure 1 below, one IR module is contained in the iTach IP2IR, and is located in module 1. The first IR connector on the IR module has an address of 1:1, whereas the third IR connector has an address of 1:3.

**Note:** In iTach units containing infrared (IR) and relay contact closure (CC) modules, those modules are always physically located in module 1 (1:\*). However, modules 1 through 3 will be accepted in commands to the IR, and 1 through 5 will be accepted by the CC. This is deliberate and allows drivers developed for the GC-100 Network Adapter series of products to function without change.

**Example:** Both of the following commands will set the first relay to closed:

setstate,1:1,1↵	correct module number (1:*)
setstate,5:1,1↵	incorrect module but is an accepted, functional command

#### IR Connectors:

Support for an IR blaster is enabled on the third (\*:3) IR connector on an IR module. This can be configured via command, or must be selected on the web site. Attempts to configure connectors other than \*:3 as an IR blaster will result in a returned error message.

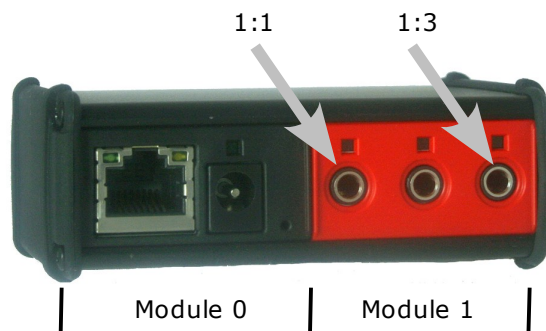


Figure 1



# iTach API Specification

## Version 1.4

### 2. Configure the IP Address

The WiFi connected iTach web server's default address is <http://169.254.1.70>, when in adhoc mode. On the **Network** settings page, either enable DHCP (for automatic IP allocation), or designate a static IP address. The TCP/IP connected iTach is set by default to use DHCP to automatically obtain an IP address from the router. To determine the IP address of an iTach using DHCP, run the iHelp Utility found at [www.globalcache.com/downloads](http://www.globalcache.com/downloads). Within 30 seconds of power up, the iTach will announce its model type, IP address, and MAC ID via a network wide beacon broadcast, which iHelp displays. The iTach will also periodically announce its IP address at intervals between 10 and 60 seconds while powered up to maintain up-to-date iHelp display information. The iHelp display may be refreshed from the **View** tab option.

### 3. Discovery Beacon

The iTach features a beacon message that can assist in locating iTach units on the network. The beacon is a UDP packet sent to the multicast IP address 239.255.250.250 on UDP port number 9131. Any system listening to this address and port will receive the periodic beacon message. The message is sent shortly after power on and then at random intervals of 10 to 60 seconds thereafter.

The beacon message has the following format:

```
AMXB<-UUID=GlobalCache_000C1E024239><-SDKClass=Utility>
<-Make=GlobalCache><-Model=iTachWF2IR><-Revision=700-1001-10>
<-Pkg_Level=GCPK001><-Config-URL=http://192.168.1.100.><-PCB_PN=025-0026-06>
<-Status=Ready>
```

The UUID value contains the unique MAC address of the iTach which is also the name registered with the DHCP server (printed as GlobalCache\_[MAC]). The Model value, at this time, will be one of the following:

iTachIP2IR	Ethernet (IP) to infrared (IR)
iTachWF2IR	WiFi to infrared
iTachIP2SL	Ethernet (IP) to serial (RS232)
iTachWF2SL	WiFi to serial
iTachIP2CC	Ethernet (IP) to contact closure (relay)
iTachWF2CC	WiFi to contact closure

Ethernet (TCP/IP) communicates by way of an RJ45 100/10 Mbit/s connection, and WiFi is a standard 802.11b wireless interface.



# iTach API Specification

## Version 1.4

### 4. Command and Data Structure

Communication with the iTach is accomplished by opening a TCP socket on Port 4998. All commands and data, with the exception of serial (RS232) data, are communicated through Port 4998. Port 4998 is used for such things as iTach status, IR data, relay operation, and reading digital input states. All information, with the exception of serial data, is communicated by comma delimited ASCII text strings terminated by a carriage return (`\r`).

Serial data is communicated via Ports 4999 and above. The serial connection in module 1 will communicate over Port 4999. Serial connections in the next lowest module will communicate over Port 5000, and so on (i.e. 1:4999, 2:5000, 3:5001, etc.).

### 5. Command Set

Commands are initiated by short ASCII string representing the command type. Typically, physical address and data information will follow. The structures of iTach commands are described in the following sections. Text enclosed in brackets (`<text>`) must be substituted by its ASCII definition. Multiple ASCII choices are divided by separator (`|`) characters.

**Note:** Commands are case sensitive.

**Example:** The network settings are queried with the `get_NET` command:

```
get_NET,<connectoraddress>\r
```

where;

`<connectoraddress>` is 0:1 (network module address)

The command ASCII string sent to the iTach is:

```
get_NET,0:1\r
```

**Note:** `get_NET` command is documented fully in section 5.1 below.

#### 5.1 General Commands

##### **getdevices**

This iTach command is used to determine installed modules and capabilities. Each module responds with its address and type. This process is completed after receiving an `endlistdevices` response.

Sent to iTach:

```
getdevices\r (query for modules and capabilities)
```

##### **device**

Sent from each iTach module in response to `getdevices`:

```
device,<moduleaddress>,<moduletype> (one sent for each module)
```

where for iTach products;

`<moduleaddress>` is |0|1|



# iTach API Specification

## Version 1.4

<moduletype> is |WIFI|ETHERNET|3 RELAY|3 IR|1 SERIAL|

All modules are included in the response followed by endlistdevices ↵ The following are the possible iTachIR responses to a getdevices command.

device,0,0 WIFI ↵ device,1,3 IR ↵ endlistdevices ↵	for WiFi to three infrared
device,0,0 WIFI ↵ device,1,1 SERIAL ↵ endlistdevices ↵	for WiFi to one serial
device,0,0 WIFI ↵ device,1,3 RELAY ↵ endlistdevices ↵	for WiFi to three relays
device,0,0 ETHERNET ↵ device,1,3 IR ↵ endlistdevices ↵	for IP to three infrared
device,0,0 ETHERNET ↵ device,1,1 SERIAL ↵ endlistdevices ↵	for IP to one serial
device,0,0 ETHERNET ↵ device,1,3 RELAY ↵ endlistdevices ↵	for IP to three relays

### getversion

The module version number may be obtained from any or all modules in an iTach. For backward compatibility with the GC-100, the previous implementation of getversion has been enabled, and both work functionally.

Sent to iTach:

getversion ↵

Sent from iTach in response to getversion:

<textversionstring> ↵

where;

<textversionstring> is the version number ASCII string

Sent to iTach:

getversion,<moduleaddress> ↵

where;

<moduleaddress> is |0|1|

Sent from iTach in response to getversion:

version,<moduleaddress>,<textversionstring> ↵

### get\_NET

This command will retrieve the current network settings and return a comma delimited string with the network settings.

Sent to iTach:

get\_NET,0:1 ↵

Sent from iTach in response to get\_NET command:

NET,0:1,<configlock>,<ipsettings>,<ipaddress>,<subnet>,<gateway>

where;

<configlock> |LOCKED|UNLOCKED|

<ipsettings> |DHCP|STATIC|



# iTach API Specification

## Version 1.4

<ipaddress> is the assigned network IP

<subnet> is the network subnet mask

<gateway> is the default network gateway

### unknowncommand

An unknowncommand response will be sent by the iTach when a command is not understood. This can happen if, for example, a connector is set up as a digital input and the command requested is sendir.

Sent from iTach in response to unknown commands:

unknowncommand, [error code]↵

**Note:** Definitions are contained in the **Error Codes** table in section 6.

## 5.2 Serial Connection

iTach serial is bi-directional, RS232 communication. All communication is 8 data bits and one stop bit. Parity, hardware flow control, and baud rate are set through the iTach internal web page or via configuration command, with baud rate enabled up to 115.2 Kbaud.

All serial data is passed through without interpretation or conversion via an assigned, unique IP port. The iTach serial connector is assigned to IP Port 4999.

If a serial connector is not configured correctly, buffer overflows (indicating data loss), or parity errors will occur. Any errors will be captured and presented on the **Serial** web page to aid in proper setup.

### 5.2.1 Serial Commands

#### set\_SERIAL

This command allows for configuration of the iTach serial.

Sent to iTach:

set\_SERIAL,1:1,<baudrate>,<flowcontrol>,<parity>↵

where:

1:1 is the iTach serial connector's address

<baudrate> is |115200|57600|38400|19200|9600|4800|2400|1200|

<flowcontrol> is |FLOW\_HARDWARE|FLOW\_NONE|

<parity> is |PARITY\_NO|PARITY\_ODD|PARITY\_EVEN|

#### Example:

set\_SERIAL,1:1,38400,FLOW\_HARDWARE,PARITY\_NO↵

This command string will set the iTach serial connector to operate at 38400 baud with hardware flow control and no parity.



# iTach API Specification

## Version 1.4

### get\_SERIAL

This command will retrieve the current iTach serial settings.

get\_SERIAL,1:1 ↵

### SERIAL

Sent from iTach in response to set\_SERIAL and get\_SERIAL.

SERIAL,1:1,<baudrate>,<flowcontrol>,<parity> ↵

### 5.2.2 RS232 Multiple Connection Mode

This mode enables four simultaneous TCP sockets to iTach port 4999. In this mode, each socket can transmit data out the serial connector on a packet-by-packet basis. Each packet received for port 4999 will be transmitted completely before another packet (from the same or different socket) is transmitted out the serial connector. This ensures complete commands can be received and understood by the serially-connected device. The only requirement is for each entire command(s) to be sent in one TCP packet. All serial data received by the serial connection is transmitted to all OPEN TCP sockets on port 4999, allowing each connected application to maintain and update serial device status. This mode can only be enabled or disabled from the **Serial** settings web page.

### 5.3 Relay Closure Communication

iTach relays are activated by sending a <1> state (close contacts), and deactivated by a <0> state (open contacts). Relay states are not preserved through a power cycle and all relays will go back to their inactive (open) state until a <1> state is restored via command.

#### 5.3.1 Relay Commands

##### setstate

Relay state is set as follows:

setstate,1:1,<outputstate> ↵	for the 1 <sup>st</sup> relay output
setstate,1:2,<outputstate> ↵	for the 2 <sup>nd</sup> relay output
setstate,1:3,<outputstate> ↵	for the 3 <sup>rd</sup> relay output

where;

<outputstate> is |0|1| (<0> is open, <1> is closed)

Sent from iTach in response to setstate:

state,1:1<0 1> ↵	for the 1 <sup>st</sup> relay state
state,1:2<0 1> ↵	for the 2 <sup>nd</sup> relay state
state,1:3<0 1> ↵	for the 3 <sup>rd</sup> relay state

**Note:** Relay type connectors can be polled for their state with the getstate command (5.4.1).



# iTach API Specification

## Version 1.4

### 5.4 IR Commands

#### set\_IR

This command allows configuration of each IR connector to the desired mode of operation. The possible modes are IR output, sensor input, sensor notify (see 5.4.2), IR blaster, and **LED lighting** (see 5.4.3). The IR blaster is only supported on the third IR connector.

Sent to iTach:

```
set_IR,1:1,<mode>↵
```

where:

<mode> is |IR|SENSOR|SENSOR\_NOTIFY|IR\_BLASTER|**LED\_LIGHTING**|

#### Example:

```
set_IR,1:3,LED_LIGHTING↵
```

This will set the third IR connector to LED lighting mode.

#### get\_IR

This command will retrieve the current mode setting for a designated connector.

Sent to iTach:

```
get_IR,1:1↵    for the 1st IR connector
```

```
get_IR,1:2↵    for the 2nd IR connector
```

```
get_IR,1:3↵    for the 3rd IR connector
```

Sent from iTach in response to get\_IR query:

```
IR,1:1,<IR|SENSOR|SENSOR_NOTIFY|IR_BLASTER|LED_LIGHTING>↵
```

#### stopir

A stopir command is used to halt IR transmission. Any remaining <repeat> counts will be discarded. A stopir command sent to a connector configured as an input will return an error message. An IR transmission halted with the stopir command will return a stopir response. Furthermore, if an IR command is halted before its completion by another connection, the originating IR connection and the connection sending stopir will both receive a stopir response. If stopped, the originating connection will not receive a completeir response.

Sent to iTach:

```
stopir,1:1↵    to stop the 1st IR connector transmission
```

```
stopir,1:2↵    to stop the 2nd IR connector transmission
```

```
stopir,1:3↵    to stop the 3rd IR connector transmission
```

Sent from iTach in response to stopir command:

```
stopir,<connectoraddress>↵
```

where;

<connectoraddress> is as defined in stopir command



# iTach API Specification

## Version 1.4

A stopir command always returns a stopir response regardless if the connector is idle or an IR transmission is actually halted. A stopir response means only that the stopir command was successfully sent to the iTach, and any transmission has been halted from the designated <connectoraddress>.

### busyIR

A busyIR response occurs when an IR command is received by a connector that is already sending an IR transmission. If multiple IP connections are present (i.e. from multiple iPhone users) there is a possibility of an IR transmission not being transmitted. This occurs when an IR command is sent to the same IR connector from another IP connection, and will cause a busyIR response.

A busyIR response does not occur if the two IR commands are executed on different IR connectors.

Sent from iTach in response to an attempt to interrupt IR transmission by another IP connection or socket:

busyIR,<connectoraddress>,<ID> ↵

where:

<connectoraddress> is the busy connector

<ID> is |0|1|2|...|65535| (ID is specified in sendir command)

**Note:** The busyIR response is returned to the originator of the unexecuted IR command. A command to stopir will only return the stopir response. At no time will a command of stopir return a response of busyIR.

### 5.4.1 Sensor/Relay State Commands

The iTach sends out notifications for digital sensor input state changes as well as allowing the inputs to be polled for their current state at any time. This command can also be used to poll the state of contact closure relays, with a value of <0> representing an open contact, and <1> representing a closed contact. Digital input connectors are the same connectors used for IR output on IR modules. The connector configuration is determined by the iTach configuration on an individual connector basis. For the following commands to operate correctly, the connector being addressed must be configured for sensor input or as a sensor notify connector. If a command requests information from a connector configured as output rather than input, an unknowncommand response will be sent from the iTach.

#### getstate

Sent to iTach:

getstate,1:1 ↵ for the 1<sup>st</sup> Sensor input or Relay

getstate,1:2 ↵ for the 2<sup>nd</sup> Sensor input or Relay

getstate,1:3 ↵ for the 3<sup>rd</sup> Sensor input or Relay

#### state

Sent from iTach in response to getstate:

state,1:1,<inputstate> ↵ for the 1<sup>st</sup> connector

state,1:2,<inputstate> ↵ for the 2<sup>nd</sup> connector

state,1:3,<inputstate> ↵ for the 3<sup>rd</sup> connector

where;





# iTach API Specification

## Version 1.4

<inputstate> is |0|1|

An inputstate of <1> represents a high digital voltage level, or absence of an input (no connection) and a <0> represents a low digital voltage level, less than .8V. The iTach sensor input level is held high internally and as such will indicate a <1> with no connection.

### 5.4.2 Sensor Notify

iTach IR connectors can now be configured as Sensor Notify. Use of the set\_IR command to enable Sensor Notify status will enable default settings. This will set the UDP broadcast port to 9132 with a timer of 10 seconds. Configuration of Sensor Notify parameters (port and timer) must be changed by way of the **IR** settings web page. There, both the UDP port and timer settings can be changed to suit network and software specifications. Sensor notifications are broadcast by way of UDP packet, sent in timed increments specified by the web configuration (in seconds), and also sent when a sensor state changes, notifying anything listening to the specified UDP port by way of an immediate network broadcast. If the timer value is set to 0, sensor notifications will broadcast only when the sensor state changes.

Sensor Notify UDP broadcast packet contents are as follows:

sensornotify,<connectoraddress>:<inputstate> ↵

where:

<connectoraddress> is as defined in section 1.

<inputstate> is as defined in 5.4.1

### 5.4.3 LED Lighting

This new state allows for pulse width modulated control (at 120Hz) of LED lighting. Wiring for the 3.5mm connector must be connected so that the Dim line comes in on the tip, and the Dim Return line is on the base. The necessary command syntax and parameters for changing the pulse width carried on the connector are located below:

Sent to iTach:

set\_LED\_LIGHTING,<connectoraddress>,<% intensity>,<linear ramp> ↵

where:

<% intensity> is the target percentage light intensity upon completion |1-100|

<linear ramp> |1-10| is the rate of change per unit time; 1 being slowest 10 being fastest.

A ramp of 1 will go from 0% to 100% in about 10 seconds, a rate of 10 will take about 1 second.

Sent from iTach in response to set\_LED\_LIGHTING:

LED\_LIGHTING,<connectoraddress>,<% intensity1>,<% intensity 2> ↵

where:

<% intensity1> |1-100| is the current lighting intensity

<% intensity2> |1-100| is the target lighting intensity upon completion

The state of the LED Lighting connector can be polled for its intensity with the get\_LED\_LIGHTING command. Note that the LED\_LIGHTING response will always return with two <% intensity> values. If the connector is not currently changing the lighting intensity, then both <% intensity> levels will be the same value.



# iTach API Specification

## Version 1.4

Sent to iTach:

```
get_LED_LIGHTING,<connectoraddress>↵
```

Sent from iTach in response to get\_LED\_LIGHTING:

```
LED_LIGHTING,<connectoraddress>,<% intensity1>,<% intensity 2>↵
```

where:

<% intensity1> is the current lighting intensity

<% intensity2> is the target lighting intensity upon completion

### 5.4.4 IR Structure

An IR, or infrared, transmission is created by sending an IR timing pattern to the iTach. This pattern is a collection of <on> and <off> states modulated with a carrier frequency (  $f$  ) which is present during the <on> state. A carrier frequency is typically between 35 to 45 KHz with some equipment manufacturers using as high as 500 KHz. The length of time for an <on> or <off> state is calculated in units of the carrier frequency period. For example, an <off> value of 24 modulated at 40 KHz produces an <off> state of 600 $\mu$ S, as calculated below.

A period is  $1/f$  or  $1/40000$  or .000025 seconds or 25 $\mu$ S,

and a value of 24 periods is 600 $\mu$ S

Figure 5.4.1 illustrates an IR timing pattern that would be created for the value shown below. IR timing patterns typically have a long, final <off> value (or rest state) to ensure the next IR command is interpreted as a separate IR transmission.

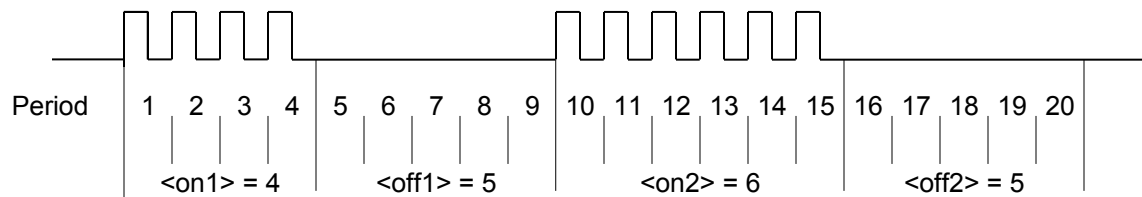


Figure 5.4.1

### 5.4.5 Sending IR

Control of IR devices is accomplished through use of the sendir command. Since IR commands may take up to 100mS to complete, the iTach provides a completeir acknowledgment to indicate when it is ready to accept the next IR command for the connector in use.

#### sendir

Sent to iTach:

```
sendir,<connectoraddress>,<ID>,<frequency>,<repeat>,<offset>,<on1>,<off1>,<on2>,<off2>,...,<onN>,<offN>↵ (where N is less than 260 or a total of 520 numbers)
```

where;



# iTach API Specification

## Version 1.4

<connectoraddress> is as defined in section 1.

<ID> is |0|1|2|...|65535| <sup>(1)</sup> (for the completeir response, see below)

<frequency> is |15000|15001|...|500000| (in hertz)

<repeat> is |1|2|...|50| <sup>(2)</sup> (the IR command is sent <repeat> times)

<offset> is |1|3|5|...|383| <sup>(3)</sup> (used if <repeat> is greater than 1, see below)

<on1> is |1|2|...|65635| <sup>(4)</sup> (number of pulses)

<off1> is |1|2|...|65635| <sup>(4)</sup> (absence of pulse periods of the carrier frequency)

<sup>(1)</sup> The <ID> is an ASCII number generated by the sender of the sendir command, which is included later in the completeir command to confirm completion of each respective sendir transmission.

<sup>(2)</sup> The <repeat> is the number of times an IR transmission is sent, if it is not halted early via a *stopir* or another IR command (see section 5.4). Values above 50 are accepted, but IR commands are sent only the maximum 50 times. In all cases, the preamble is only sent once (see <offset> below).

<sup>(3)</sup> An <offset> applies when the <repeat> is greater than 1. For IR commands that have preambles, an <offset> is employed to avoid repeating the preamble during repeated IR timing patterns. The <offset> value indicates the location within the timing pattern to start repeating the IR command as indicated below. The <offset> will always be an odd value since a timing pattern begins with an <on> state and must end with an <off> state.

<offset> odd value	repeat start location	<offset> even value	should not point here
1	<on1>	2	<off1> no preamble
3	<on2>	4	<off2>
....	....	....	....
n-1	<on((n/2) -1)>	n	off(n/2)> where n is an even number

<sup>(4)</sup> Since IR transmissions ends in an <off> condition, there must be an equal number of <on> and <off> states. Also, every <on> and <off> state must meet an 80µS minimum time requirement for the iTach to work properly.

**Example:** With a carrier frequency of 48 KHz, the minimum value for <on> and <off> states is calculated below.

$$\langle \text{off} \rangle_{\min} = \langle \text{on} \rangle_{\min} \geq 80\mu\text{S} * f = 80\mu\text{S} * 48\text{KHz} = 3.84$$

For proper iTach operation, all <on> and <off> values in the timing pattern must be 4 or higher.

All of the conditions above must be met for valid sendir commands. When a variable is missing or outside the accepted range, an unknowncommand will be sent by the iTach. As an exercise, the sendir commands below will trigger an iTach unknowncommand response.

sendir,5:3,3456,23400,1,1,24,48,24,960 ↵

iTach module 5 does not exist

Response: ERR\_0:0,002

sendir,1:2.23333,40000,2,3,24,48,24,48,960 ↵ not an equal number of <on> and <off>



# iTach API Specification

## Version 1.4

*Response:* ERR\_1:2,010

sendir,1:3,0,40000,2,2,24,48,24,960↵

<offset> is an even number

*Response:* ERR\_1:3,007

IR compressed format assigns the first 15 *unique* <on><off> pairs capital letters (i.e. A,B,C, etc.) to represent them. In the event that a pair is used in many places inside an IR command, commands can be written with the capital letter in place of the designated pair without being offset by commas.

**Example:** The simple IR command

sendir,1:2,2445,40000,1,1,4,5,4,5,8,9,4,5,8,9,8,9↵

can be shortened with this feature: ("4,5" is assigned A, and "8,9" is assigned B)

sendir,1:2,2445,40000,1,1,4,5A8,9ABB↵

Both commands are syntactically correct, are accepted by the iTach, and will transmit an identical IR command.

### completeir

All successful sendir commands are acknowledged with a completeir response from the iTach after completion of the IR transmission. The completeir response associates with the sendir command through an <ID>. When utilized, the <ID>s can provide a unique identifier to determine which IR transmission has completed.

Sent from iTach in response to successful sendir:

completeir,1:1,<ID>↵ for the 1<sup>st</sup> IR connector

completeir,1:2,<ID>↵ for the 2<sup>nd</sup> IR connector

completeir,1:3,<ID>↵ for the 3<sup>rd</sup> IR connector

where;

<ID> is |0|1|2|...|65535| (ID is specified by originating sendir command)

**Example:** A few simple IR commands are shown below:

The following will send the IR timing sequence illustrated in figure 4.2a to the 2nd IR connector on the iTach shown in Figure 1.

Sent to iTach:

sendir,1:2,2445,40000,1,1,4,5,6,5↵

Sent by iTach in response to sendir:

completeir,1:2,2445↵

In the next example, the following two IR commands will send the same IR timing pattern.

Below are two ways to send the same simple IR timing pattern of 24,12,24,960 four times with a preamble of 34,48:

sendir,1:2,4444,34500,1,1,34,48,24,12,24,960,24,12,24,960,24,12,24,960,24,12,24,960↵

sendir,1:2,34,34500,4,3,34,48,24,12,24,960↵



# iTach API Specification

## Version 1.4

Acknowledgments for above IR commands are:

```
completeir,1:2,4444 ↵
```

```
completeir,1:2,34 ↵
```

Although the same command is sent four times by both sendir commands, the <ID>s are different, and therefore cannot be considered the same command. The second IR command structure is the recommended method, avoiding long commands and allowing repetition of the command to be halted if requested.

### 5.4.6 Smooth Continuous IR Commands

A general discussion is necessary to better understand how smooth continuous IR commands are executed by the iTach. This desirable feature is utilized for smooth volume control or repeating an operation without the appearance of choppy actions. The approach of sending an IR command with very large repeat counts and stopping it upon request will work, but can lead to undesirable incidents. Consider the scenario of large repeat count IR command for raising the volume in a smooth fashion. The command works properly until the connection is broken. A repeating IR command is sent, volume continuously increases, then the controlling iPhone is unexpectedly dropped. The volume continues to rise (possibly damaging equipment) until a stopir command is ultimately received.

The iTach solution is to limit the repeat count. Hence, to create a smooth IR operation, the iTach resets the IR repeat count each time the identical IR command (from the same IP connection) is resent. This method will **not** interrupt and restart the IR command, but reset the IR repeat count back to the original value.

**Example:** If the IR repeat count is set to 5, and the IR command has transmitted 3 times, receipt of the same command causes the repeat count to be reset back to 5. This process can continue indefinitely while a volume button is held down to create a smooth operation. However, at no time can the command repeat more than 5 times after the button is released or an IP connection is inadvertently lost, preventing a potentially serious issue.

By selecting an appropriate <repeat> value, the need for a stopir command is eliminated. In this example the volume continues to increase smoothly by retransmitting repeated IR commands due to the volume button being pressed. As long as the next repeated IR command is received before the previous command finishes, smooth operation is realized. By choosing a low repeat value, the volume increase will stop when the volume button is released. Also, proper IR operations happen even with unintended network delays due to traffic or WiFi connectivity. In this unlikely event, only small hesitations will be experienced during IR operation.

In the event that the identical command is not received before the original command is finished, the command will be registered as a brand new command, and is sent as such. The command in question will operate functionally the same, but delays between commands may be evident when used in this way. Increasing the <repeat> value will likely eliminate these discrepancies.

### 5.4.7 IR Learning

Each iTach unit contains an on-board IR learner, which is located in the small hole located below and to the right of the power connector. IR Learner Mode is enabled by implementing the get\_IRL command.

#### get\_IRL

Sent to iTach:

```
get_IRL ↵
```



# iTach API Specification

## Version 1.4

Sent from iTach in response to get\_IRL:

IR Learner Enabled↵

Once enabled, the iTach sends an uncompressed Global Caché format sendir command, terminated by a carriage return, through TCP packets via port 4998. Although the iTach family of products supports up to eight simultaneous connections, the captured command will only be sent to the connection that initiated learner mode. Learner mode is disabled when iTach units receive any command, or when stop\_IRL is sent.

### **stop\_IRL**

Sent to iTach:

stop\_IRL↵

Sent from iTach in response to stop\_IRL:

IR Learner Disabled↵



# iTach API Specification

## Version 1.4

### 6. Error Codes

The chart below provides a list of error messages returned by the iTach from port 4998 and the explanation of each message. Messages are returned in the aforementioned syntax.

Error Message	Explanation
ERR_01	Invalid command. Command not found.
ERR_02	Invalid module address (does not exist).
ERR_03	Invalid connector address (does not exist).
ERR_04	Invalid ID value.
ERR_05	Invalid frequency value.
ERR_06	Invalid repeat value.
ERR_07	Invalid offset value.
ERR_08	Invalid pulse count.
ERR_09	Invalid pulse data.
ERR_10	Uneven amount of <on off> statements.
ERR_11	No carriage return found.
ERR_12	Repeat count exceeded.
ERR_13	IR command sent to input connector.
ERR_14	Blaster command sent to non-blaster connector.
ERR_15	No carriage return before buffer full.
ERR_16	No carriage return.
ERR_17	Bad command syntax.
ERR_18	Sensor command sent to non-input connector.
ERR_19	Repeated IR transmission failure.
ERR_20	Above designated IR <on off> pair limit.
ERR_21	Symbol odd boundary.
ERR_22	Undefined symbol.
ERR_23	Unknown option.
ERR_24	Invalid baud rate setting.
ERR_25	Invalid flow control setting.
ERR_26	Invalid parity setting.
ERR_27	Settings are locked.



# iTach API Specification

## Version 1.4

### 7. Bugs

There are no known bugs that affect the current revision of iTach models.