



# Heart Rate Service (HRS)

## Application Programming Interface Reference Manual

Profile Version: 1.0

Release: 4.0.1  
January 10, 2013



Bluetooth and the Bluetooth logos are trademarks owned by Bluetooth SIG, Inc., USA and licensed to Stonestreet One, LLC. Bluetopia®, Stonestreet One™, and the Stonestreet One logo are registered trademarks of Stonestreet One, LLC, Louisville, Kentucky, USA. All other trademarks are property of their respective owners.  
Copyright © 2000-2013 by Stonestreet One, LLC. All rights reserved.

## Table of Contents

<b>1. INTRODUCTION.....</b>	<b>3</b>
1.1 Scope .....	3
1.2 Applicable Documents .....	4
1.3 Acronyms and Abbreviations .....	4
<b>2. HRS PROGRAMMING INTERFACE.....</b>	<b>5</b>
2.1 Heart Rate Service Commands.....	5
HRS_Initialize_Service.....	5
HRS_Cleanup_Service .....	7
HRS_Set_Body_Sensor_Location.....	7
HRS_Query_Body_Sensor_Location .....	8
HRS_Read_Client_Configuration_Response .....	9
HRS_Notify_Heart_Rate_Measurement .....	10
HRS_Decode_Heart_Rate_Measurement.....	11
HRS_Decode_Body_Sensor_Location.....	12
HRS_Format_Heart_Rate_Control_Command .....	12
2.2 Heart Rate Service Event Callback Prototypes .....	13
2.2.1 SERVER EVENT CALLBACK .....	13
HRS_Event_Callback_t .....	13
2.3 Heart Rate Service Events.....	14
2.3.1 HEART RATE SERVICE SERVER EVENTS.....	14
etHRS_Server_Read_Client_Configuation_Request .....	15
etHRS_Server_Client_Configuration_Update.....	15
etHRS_Server_Heart_Rate_Control_Point_Command .....	16
<b>3. FILE DISTRIBUTIONS.....</b>	<b>18</b>

# 1. Introduction

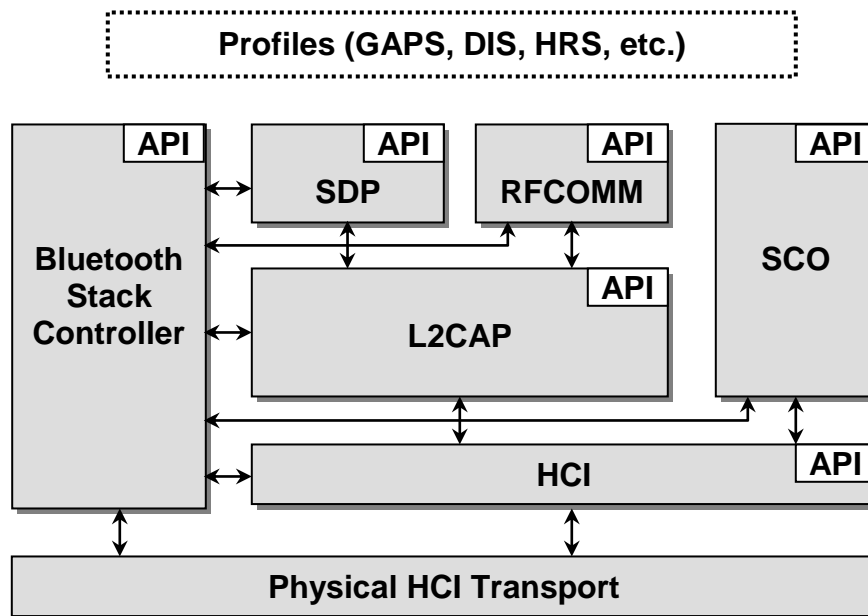
Bluetopia®+LE is Stonestreet One's Bluetooth protocol stack that supports the adopted Bluetooth low energy specification. Stonestreet One's upper level protocol stack that supports Single Mode devices is Bluetopia®+LE Single. More specifically, this stack is a software solution that resides above the Physical HCI (Host Controller Interface) Transport Layer and extends through the L2CAP (Logical Link Control and Adaptation Protocol), ATT (Attribute Protocol) Link Layers, the GAP (Generic Attribute Profile) Layer and the Genetic Attribute Protocol (GATT) Layer. In addition to basic functionality of these layers, the Bluetooth Protocol Stack by Stonestreet One provides implementations of the Device Information Service (DIS), HRS (Heart Rate Service), and several of the Bluetooth Profiles. Program access to these layers, services, and profiles is handled via Application Programming Interface (API) calls.

The remainder of this chapter has sections on the scope of this document, other documents applicable to this document, and a listing of acronyms and abbreviations. Chapter 2 is the API reference that contains a description of all programming interfaces for the Heart Rate Service Profile Stack provided by Bluetopia®+LE Single. And, Chapter 3 contains the header file name list for the Heart Rate Service library.

## 1.1 Scope

This reference manual provides information on the HRS API. This API is available on the full range of platforms supported by Stonestreet One:

- Windows
- Windows Mobile
- Windows CE
- Linux
- QNX
- Other Embedded OS



**Figure 1-1 The Stonestreet One Bluetooth Protocol Stack**

## 1.2 Applicable Documents

The following documents may be used for additional background and technical depth regarding the Bluetooth technology.

1. *Specification of the Bluetooth System, Volume 1, Architecture and Terminology Overview*, version 4.0, June 30, 2010.
2. *Specification of the Bluetooth System, Volume 6, Core System Package [Low Energy Controller Volume]*, version 4.0, June 30, 2010.
3. *Bluetopia® Protocol Stack, Application Programming Interface Reference Manual*, version 4.0.1, January 10, 2013.
4. *Bluetooth Heart Rate Service Specification*, version v10r00, May 22, 2012.

Possible error returns are listed for each API function call. These are the *most likely* errors, but in fact programmers should allow for the possibility of any error listed in the BTErrors.h header file to occur as the value of a function return.

## 1.3 Acronyms and Abbreviations

Acronyms and abbreviations used in this document and other Bluetooth specifications are listed in the table below.

Term	Meaning
API	Application Programming Interface
ATT	Attribute Protocol
BD_ADDR	Bluetooth Device Address
BT	Bluetooth
GAPS	Generic Access Profile Service
GATT	Generic Attribute Protocol
HCI	Host Controller Interface
HRS	Heart Rate Service
HS	High Speed
L2CAP	Logical Link Control and Adaptation Protocol
LE	Low Energy
LSB	Least Significant Bit
MSB	Most Significant Bit

## 2. HRS Programming Interface

The Heart Rate Service, HRS, programming interface defines the protocols and procedures to be used to implement HRS capabilities for both Server and Client services. The HRS commands are listed in section 2.1, the event callback prototypes are described in section 2.2, the HRS events are itemized in section 2.3. The actual prototypes and constants outlines in this section can be found in the **HRSAPI.h** header file in the Bluetopia distribution.

### 2.1 Heart Rate Service Commands

The available HRS command functions are listed in the table below and are described in the text that follows.

Server Commands	
Function	Description
HRS_Initialize_Service	Opens a HRS Server.
HRS_Cleanup_Service	Closes an opened HRS Server.
HRS_Set_Body_Sensor_Location	Sets the Body Sensor Location on the specified HRS Instance.
HRS_Query_Body_Sensor_Location	Queries the current Sensor Location on the specified HRS Instance.
HRS_Read_Client_Configuration_Response	Responds to a HRS Read Client Configuration Request.
HRS_Notify_Heart_Rate_Measurement	Sends a Heart Reate Measurement notification to a specified remote device.
HRS_Decode_Heart_Rate_Measurement	Parses a value received from a remote HRS Server interpreting it as a Heatrt Rate Measurement characteristic.
HRS_Decode_Body_Sensor_Location	Parses a value received from a remote HRS Server interpreting it as a Body Sensor Location value.
HRS_Format_Heart_Rate_Control_Command	Formats a Heart rate Control Command into user specified buffer.

#### HRS\_Initialize\_Service

The following function is responsible for opening a HRS Server on a specified Bluetooth Stack.

**Notes:**

1. Only one HRS Server, per Bluetooth Stack ID, may be open at a time.

2. All Client Requests will be dispatched to the EventCallback function that is specified by the second parameter to this function.

**Prototype:**

```
int BTPSAPI HRS_Initialize_Service(unsigned int BluetoothStackID, unsigned long  
Supported_Commands, HRS_Event_Callback_t EventCallback, unsigned long  
CallbackParameter, unsigned int *ServiceID);
```

**Parameters:**

BluetoothStackID <sup>1</sup>	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
Supported_Commands	The mask of supported Heart Rate Control Commands. The parameter must be made up of a bit mask of the form: HRS_HEART_RATE_CONTROL_POINT_XXX_SUPPORTED
EventCallback	Callback function that is registered to receive events that are associated with the specified service.
CallbackParameter	A user-defined parameter that will be passed back to the user in the callback function.
ServiceID	Unique GATT Service ID of the registered HRS service returned from GATT_Register_Service API.

**Return:**

Positive non-zero if successful. The return value will be the Service ID of HRS Server that was successfully opened on the specified Bluetooth Stack ID. This is the value that should be used in all subsequent function calls that require Instance ID.

Negative if an error occurred. Possible values are:

```
HRS_ERROR_INSUFFICIENT_RESOURCES  
HRS_ERROR_SERVICE_ALREADY_REGISTERED  
HRS_ERROR_INVALID_PARAMETER  
BTGATT_ERROR_INVALID_SERVICE_TABLE_FORMAT  
BTGATT_ERROR_INSUFFICIENT_RESOURCES  
BTGATT_ERROR_INVALID_PARAMETER  
BTGATT_ERROR_INVALID_BLUETOOTH_STACK_ID  
BTGATT_ERROR_NOT_INITIALIZED
```

**Notes:**

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

## HRS\_Cleanup\_Service

This function is responsible for cleaning up and freeing all resources associated with a Heart Rate Service Instance. After this function is called, no other Heart Rate Service function can be called until after a successful call to the HRS\_Initialize\_Service() function is performed.

### Prototype:

```
int BTPSAPI HRS_Cleanup_Service(unsigned int BluetoothStackID,  
                                unsigned int InstanceID);
```

### Parameters:

BluetoothStackID <sup>1</sup>	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
InstanceID	The Service Instance ID to close. This InstanceID was returned from the HRS_Initialize_Service().

### Return:

Zero if successful.

Negative if an error occurred. Possible values are:

HRS\_ERROR\_INVALID\_PARAMETER  
HRS\_ERROR\_INVALID\_INSTANCE\_ID

### Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

## HRS\_Set\_Body\_Sensor\_Location

This function is responsible for setting the Body Sensor Location on the specified HRS Instance.

### Prototype:

```
int BTPSAPI HRS_Set_Body_Sensor_Location(unsigned int BluetoothStackID, unsigned  
int InstanceID, Byte_t Body_Sensor_Location);
```

### Parameters:

BluetoothStackID <sup>1</sup>	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
InstanceID	The Service Instance ID to close. This InstanceID was returned from the HRS_Initialize_Service().
Body_Sensor_Location	The value for which the Body Sensor Location should be set for the specified HRS Instance. The parameter should be an

enumerated value of the form  
HRS\_BODY\_SENSOR\_LOCATION\_XXX.

**Return:**

Zero if successful.

Negative if an error occurred. Possible values are:

HRS\_ERROR\_INVALID\_INSTANCE\_ID  
HRS\_ERROR\_INVALID\_PARAMETER  
BTPS\_ERROR\_FEATURE\_NOT\_AVAILABLE  
BTGATT\_ERROR\_NOT\_INITIALIZED  
BTGATT\_ERROR\_INVALID\_BLUETOOTH\_STACK\_ID  
BTGATT\_ERROR\_INVALID\_PARAMETER

**Notes:**

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

**HRS\_Query\_Body\_Sensor\_Location**

This function is responsible for querying the current Body Sensor Location on the specified HRS Instance.

**Prototype:**

```
int BTPSAPI HRS_Query_Body_Sensor_Location(unsigned int BluetoothStackID,
    unsigned int InstanceID, Byte_t *Body_Sensor_Location);
```

**Parameters:**

BluetoothStackID <sup>1</sup>	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
InstanceID	The Service Instance ID to close. This InstanceID was returned from the HRS_Initialize_Service().
Body_Sensor_Location	A pointer to return the current Body Sensor Location for the specified HRS Instance.

**Return:**

Zero if successful.

An error code if negative; one of the following values:

HRS\_ERROR\_INVALID\_INSTANCE\_ID  
HRS\_ERROR\_INVALID\_PARAMETER  
BTPS\_ERROR\_FEATURE\_NOT\_AVAILABLE  
BTGATT\_ERROR\_NOT\_INITIALIZED  
BTGATT\_ERROR\_INVALID\_BLUETOOTH\_STACK\_ID  
BTGATT\_ERROR\_INVALID\_PARAMETER



**Notes:**

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

**HRS\_Read\_Client\_Configuration\_Response**

The following function is responsible for responding to a HRS Read Client Configuration Request.

**Prototype:**

```
int BTPSAPI HRS_Read_Client_Configuration_Response(unsigned int BluetoothStackID,  
    unsigned int InstanceID, unsigned int TransactionID,  
    Word_t ClientConfiguration);
```

**Parameters:**

BluetoothStackID <sup>1</sup>	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
InstanceID	The Service Instance ID to close. This InstanceID was returned from the HRS_Initialize_Service().
TransactionID	The Transaction ID of the original read request. This value was received in the etHRS_Read_Client_Configuration_Request event.
ClientConfiguration	The Client Configuration to send to the remote device.

**Return:**

Zero if successful.

Negative if an error occurred. Possible values are:

```
HRS_ERROR_INVALID_INSTANCE_ID  
HRS_ERROR_INVALID_PARAMETER  
BTGATT_ERROR_NOT_INITIALIZED  
BTGATT_ERROR_INVALID_BLUETOOTH_STACK_ID  
BTGATT_ERROR_INVALID_PARAMETER
```

**Possible Events:**

etGATT\_Client\_Read\_Response

**Notes:**

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

## HRS\_Notify\_Heart\_Rate\_Measurement

The following function is responsible for sending a Heart Rate Measurement notification to a specified remote device.

### Prototype:

```
int BTPSAPI HRS_Notify_Heart_Rate_Measurement(unsigned int BluetoothStackID,
    unsigned int InstanceID, unsigned int ConnectionID,
    HRS_Heart_Rate_Measurement_Data_t *Heart_Rate_Measurement)
```

### Parameters:

BluetoothStackID <sup>1</sup>	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
InstanceID	The Service Instance ID to close. This InstanceID was returned from the HRS_Initialize_Service().
ConnectionID	Connection ID of the currently connected remote client device to send the handle/value notification.
Heart_Rate_Measurement	The Heart Rate Measurement data to notify. The Heart Rate Measurement Data structure is as follows:

```
typedef struct
{
    Byte_t    Flags;
    Word_t    Heart_Rate;
    Word_t    Energy_Expended;
    Word_t    Number_Of_RR_Intervals;
    Word_t    RR_Intervals[1];
} HRS_Heart_Rate_Measurement_Data_t;
```

### Return:

Zero if successful.

Negative if an error occurred. Possible values are:

```
HRS_ERROR_INVALID_INSTANCE_ID
HRS_ERROR_INVALID_PARAMETER
HRS_ERROR_INSUFFICIENT_RESOURCES
BTGATT_ERROR_NOT_INITIALIZED
BTGATT_ERROR_INVALID_BLUETOOTH_STACK_ID
BTGATT_ERROR_INVALID_PARAMETER
```

### Possible Events:

etGATT\_Connection\_Server\_Notification

### Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

## HRS\_Decode\_Heart\_Rate\_Measurement

The following function is responsible for parsing a value received from a remote HRS Server interpreting it as a Heart Rate Measurement characteristic.

### Note:

1. On INPUT the Number\_Of\_RR\_Intervals member of the HeartRateMeasurementn parameter must contain the number of entries in the RR\_Intervals array. On RETURN this parameter will contain the actual number of RR Interval values that were parsed from the Heart Rate Measurement value (which will always be less than or equal to the number of entries allocated in the structure).
2. It is possible to query the total number of RR Interval values in the Heart Rate measurement value by passing 0 for the Number\_Of\_RR\_Intervals member of the HeartRateMeasurement parameter. In this case on return the Number\_Of\_RR\_Intervals member will contain the total number of RR Interval values in the Heart Rate Measurement value BUT no RR Intervals will be parsed into the HeartRateMeasurement structure.

### Prototype:

```
int BTPSAPI HRS_Decode_Heart_Rate_Measurement(unsigned int ValueLength, Byte_t
    *Value, HRS_Heart_Rate_Measurement_Data_t *HeartRateMeasurement);
```

### Parameters:

ValueLength	Specifies the length of the Heart Rate Context value returned by the remote HRS Server.
Value	Value is a pointer to the Heart Rate Context data returned by the remote HRS Server.
HeartRateMeasurement	A pointer to store the parsed Heart Rate Measurement value. The Heart Rate Measurement Data structure is as follows:

```
typedef struct
{
    Byte_t    Flags;
    Word_t    Heart_Rate;
    Word_t    Energy_Expended;
    Word_t    Number_Of_RR_Intervals;
    Word_t    RR_Intervals[1];
} HRS_Heart_Rate_Measurement_Data_t;
```

### Return:

DecodeHeartRateMeasurement(ValueLength, Value, HeartRateMeasurement)

### Possible Events:

Unknown\XXX

## HRS\_Decode\_Body\_Sensor\_Location

The following function is responsible for parsing a value received from a remote HRS Server interpreting it as a Body Sensor Location value.

### Prototype:

```
int BTPSAPI HRS_Decode_Body_Sensor_Location(unsigned int ValueLength, Byte_t
*Value, Byte_t *BodySensorLocation);
```

### Parameters:

ValueLength	Specifies the length of the Heart Rate Context value returned by the remote HRS Server.
Value	Value is a pointer to the Heart Rate Context data returned by the remote HRS Server.
BodySensorLocation	A pointer to store the parsed Body Sensor Location value.

### Return:

DecodeBodySensorLocation(ValueLength, Value, BodySensorLocation)

### Possible Events:

Unknown\XXX

## HRS\_Format\_Heart\_Rate\_Control\_Command

The following function is responsible for formatting a Heart Rate Control Command into a user specified buffer.

### Prototype:

```
int BTPSAPI HRS_Format_Heart_Rate_Control_Command
(HRS_Heart_Rate_Control_Command_t Command, unsigned int BufferLength, Byte_t
*Buffer);
```

### Parameters:

Command	The command to format. The Heart Rate Control Command enum is as follows: <pre>typedef enum {     ccResetEnergyExpended =         HRS_HEART_RATE_CONTROL_POINT_RESET_         ENERGY_EXPENDED } HRS_Heart_Rate_Control_Command_t;</pre>
BufferLength	Specifies the Length of the Buffer
Buffer	A pointer, pointing to memory of size BufferLength to store the Heart Rate Control request Data after formatting.

**Return:**

Zero if successful.

Negative if an error occurred. Possible values are:

HRS\_ERROR\_INVALID\_PARAMETER  
 BTGATT\_ERROR\_NOT\_INITIALIZED  
 BTGATT\_ERROR\_INVALID\_BLUETOOTH\_STACK\_ID  
 BTGATT\_ERROR\_INVALID\_PARAMETER

**Possible Events:**

Unknown/XXX

## 2.2 Heart Rate Service Event Callback Prototypes

### 2.2.1 Server Event Callback

The event callback function mentioned in the HRS\_Initialize\_Service command accepts the callback function described by the following prototype.

**HRS\_Event\_Callback\_t**

This The event callback function mentioned in the HRS\_Initialize\_Service command accepts the callback function described by the following prototype.

**Note:**

This function MUST NOT Block and wait for events that can only be satisfied by Receiving HRS Service Event Packets. A Deadlock WILL occur because NO HRS Event Callbacks will be issued while this function is currently outstanding.

**Prototype:**

```
typedef void (BTPSAPI *HRS_Event_Callback_t)(unsigned int BluetoothStackID,
      HRS_Event_Data_t *HRS_Event_Data, unsigned long CallbackParameter);
```

**Parameters:**

BluetoothStackID <sup>1</sup>	Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC_Initialize.
HRS_Event_Data_t	Data describing the event for which the callback function is called. This is defined by the following structure:

```
typedef struct
{
    HRS_Event_Type_t    Event_Data_Type;
    Word_t              Event_Data_Size;
    union
    {
        HRS_Read_Client_Configuration_Data_t
            *HRS_Read_Client_Configuration_Data;
```

```

HRS_Client_Configuration_Update_Data_t
    *HRS_Client_Configuration_Update_Data;
HRS_Heart_Rate_Control_Command_Data_t
    *HRS_Heart_Rate_Control_Command_Data;
} Event_Data;
} HRS_Event_Data_t;

```

Where, Event\_Data\_Type is one of the enumerations of the event types listed in the table in section 2.3, and each data structure in the union is described with its event in that section as well.

CallbackParameter

User-defined parameter that was defined in the callback registration.

### Return:

XXX/None

### Notes:

1. The BluetoothStackID parameter is not included in versions of Bluetopia that have been optimized to only control a single Bluetooth device, such as some embedded versions of Bluetopia. Please refer to the appropriate header file to determine if this parameter is part of the function call or not.

## 2.3 Heart Rate Service Events

The Heart Rate Service contains events that are received by the Server. The following sections detail those events.

### 2.3.1 Heart Rate Service Server Events

The possible Heart Rate Service Server Events from the Bluetooth stack are listed in the table below and are described in the text which follows:

Server Commands	
Function	Description
etHRS_Server_Read_Client_Configuration_Request	Dispatched to a HRS Server when a HRS Client is attempting to read a descriptor.
etHRS_Server_Client_Configuration_Update	Dispatched to a HRS Server when a HRS Client has written a Client Configuration descriptor.
etHRS_Server_Heart_Rate_Control_Point_Command	Dispatched to a HRS Server when a HRS client sends a request to read Heart Rate data.

**etHRS\_Server\_Read\_Client\_Configuation\_Request**

The following HRS Profile Event is dispatched to a HRS Server when a HRS Client is attempting to read a descriptor.

**Return Structure:**

```
typedef struct
{
    unsigned int          InstanceID;
    unsigned int          ConnectionID;
    unsigned int          TransactionID;
    GATT_Connection_Type_t ConnectionType;
    BD_ADDR_t            RemoteDevice;
    HRS_Characteristic_Type_t ClientConfigurationType;
} HRS_Read_Client_Configuration_Data_t;
```

**Event Parameters:**

InstanceID	Identifies the Local Server Instance to which the Remote Client has connected.
ConnectionID	Connection ID of the currently connected remote HRS server device.
TransactionID	The TransactionID identifies the transaction between a client and server. This identifier should be used to respond to the current request.
ConnectionType	Identifies the type of remote Bluetooth device that is connected. Currently this value will be gctLE only.
RemoteDevice	Specifies the address of the Client Bluetooth device that has connected to the specified Server.
ClientConfigurationType	The specified configuration type of the client as defined by the enum:

```
typedef enum
{
    ctHeartReateMeasurement
} HRS_Characteristic_Type_t;
```

**etHRS\_Server\_Client\_Configuration\_Update**

The following HRS Profile Event is dispatched to a HRS Server when a HRS Client has written a Client Configuation descriptor.

**Return Structure:**

```
typedef struct
{
    unsigned int          InstanceID;
    unsigned int          ConnectionID;
    GATT_Connection_Type_t ConnectionType;
    BD_ADDR_t            RemoteDevice;
```

```

        HRS_Characteristic_Type_t    ClientConfigurationType;
        Word_t                      ClientConfiguration;
    } HRS_Client_Configuration_Update_Data_t;

```

**Event Parameters:**

InstanceID	Identifies the Local Server Instance to which the Remote Client has connected.
ConnectionID	Connection ID of the currently connected remote HRS server device.
ConnectionType	Identifies the type of remote Bluetooth device that is connected. Currently this value will be gctLE only.
RemoteDevice	Specifies the address of the Client Bluetooth device that has connected to the specified Server.
ClientConfigurationType	This specifies the descriptor that the Client is writing as defined by the enum:  <pre> typedef enum {     ctHeartReateMeasurement } HRS_Characteristic_Type_t; </pre>
ClientConfiguration	The New Client Configuration for the specified characteristic.

**etHRS\_Server\_Heart\_Rate\_Control\_Point\_Command**

The following HRS Profile Event is dispatched to a HRS Server when a HRS Client has sent a Heart Rate Control Point Command.

**Return Structure:**

```

typedef struct
{
    unsigned int                InstanceID;
    unsigned int                ConnectionID;
    GATT_Connection_Type_t      ConnectionType;
    BD_ADDR_t                   RemoteDevice;
    HRS_Heart_Rate_Control_Command_t Command;
} HRS_Read_Current_Time_Request_Data_t;

```

**Event Parameters:**

InstanceID	Identifies the Local Server Instance to which the Remote Client has connected.
ConnectionID	Connection ID of the currently connected remote HRS server device.
ConnectionType	Identifies the type of remote Bluetooth device that is connected. Currently this value will be gctLE only.



**RemoteDevice** Specifies the address of the Client Bluetooth device that has connected to the specified Server.

**Command** The Heart Rate Control Point Command that the client sent. The Heart Rate Control Command enum is as follows:

```
typedef enum
{
    ccResetEnergyExpended =
        HRS_HEART_RATE_CONTROL_POINT_RESET_
        ENERGY_EXPENDED
} HRS_Heart_Rate_Control_Command_t;
```

### 3. File Distributions

The header files that are distributed with the Bluetooth Heart Rate Service Library are listed in the table below

File	Contents/Description
HRSAPI.h	Bluetooth Heart Rate Service (GATT based) API Type Definitions, Constants, and Prototypes.
HRSTypes.h	Bluetooth Heart Rate Service Types.
SS1BTHRS.h	Bluetooth Heart Rate Service Include file