



# Health Thermometer Service (HTS)

## 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<sup>®</sup>, Stonestreet One<sup>™</sup>, 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. HTS PROGRAMMING INTERFACE .....</b>	<b>5</b>
2.1 Health Thermometer Service Commands .....	5
HTS_Initialize_Service.....	6
HTS_Cleanup_Service.....	7
HTS_Set_Temperature_Type .....	7
HTS_Query_Temperature_Type .....	8
HTS_Set_Measurement_Interval.....	9
HTS_Query_Measurement_Interval .....	9
HTS_Set_Valid_Range.....	10
HTS_Query_Valid_Range .....	11
HTS_Read_Client_Configuration_Response .....	12
HTS_Notify_Intermediate_Temperature .....	13
HTS_Indicate_Measurement_Interval .....	14
HTS_Indicate_Temperature_Measurement.....	15
HTS_Decode_Temperature_Measurement.....	16
HTS_Decode_Valid_Range.....	17
2.2 Health Thermometer Service Event Callback Prototypes .....	18
2.2.1 SERVER EVENT CALLBACK .....	18
HTS_Event_Callback_t .....	18
2.3 Health Thermometer Service Events .....	19
2.3.1 HEALTH THERMOMETER SERVICE SERVER EVENTS .....	19
etHTS_Server_Read_Client_Configuation_Request .....	20
etHTS_Server_Client_Configuration_Update .....	21
etHTS_Measurement_Interval_Update .....	21
etHTS_Confirmation_Response .....	22
<b>3. FILE DISTRIBUTIONS.....</b>	<b>24</b>

# 1. Introduction

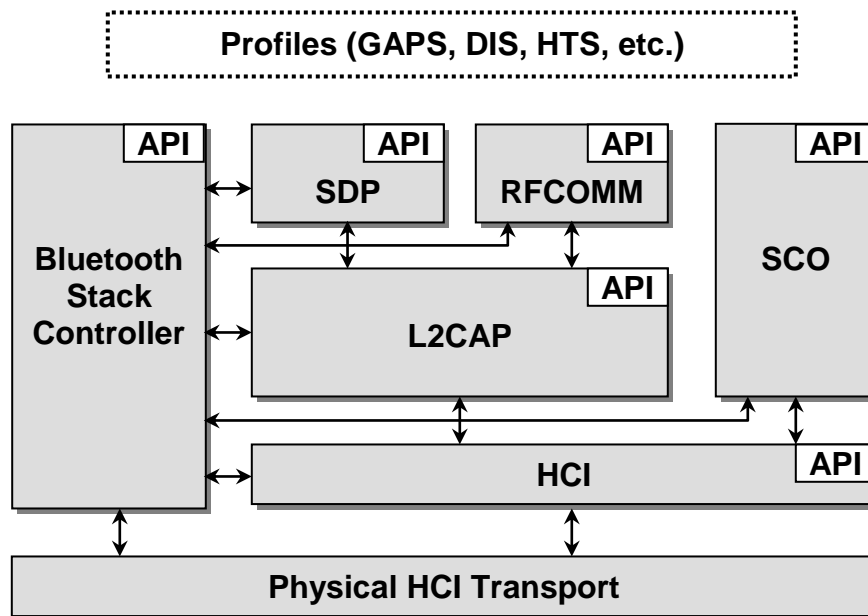
Bluetopia®+LE is Stonestreet One's Bluetooth protocol stack that supports the adopted Bluetooth low energy specification. Stonestreet One's upper level protocol stack that supports Single Mode devices is Bluetopia®+LE Single. More specifically, this stack is a software solution that resides above the 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), HTS (Health Thermometer 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 Health Thermometer Service Profile Stack provided by Bluetopia®+LE Single. And, Chapter 3 contains the header file name list for the Health Thermometer Service library.

## 1.1 Scope

This reference manual provides information on the HTS 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 Health Thermometer Service Specification*, version v10r00, April 3, 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
HS	High Speed
HTS	Health Thermometer Service
L2CAP	Logical Link Control and Adaptation Protocol
LE	Low Energy
LSB	Least Significant Bit
MSB	Most Significant Bit

## 2. HTS Programming Interface

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

### 2.1 Health Thermometer Service Commands

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

Server Commands	
Function	Description
HTS_Initialize_Service	Opens a HTS Server.
HTS_Cleanup_Service	Closes an opened HTS Server.
HTS_Set_Temperature_Type	Sets the Temperature Type on the specified HTS Instance.
HTS_Query_Temperature_Type	Queries the current Temperature Type on the specified HTS Instance.
HTS_Set_Measurement_Interval	Sets the Measurement Interval on the specified HTS Instance.
HTS_Query_Measurement_Interval	Queries the current Measurement Interval on the specified HTS Instance.
HTS_Set_Valid_Range	Sets the Valid Range descriptor value on the specified HTS Instance.
HTS_Query_Valid_Range	Queries the Valid Range descriptor value on the specified HTS Instance.
HTS_Read_Client_Configuration_Response	Responds to a HTS Read Client Configuration Request.
HTS_Notify_Intermediate_Temperature	Sends an Intermediate Temperature notification to a specified remote device.
HTS_Indicate_Measurement_Interval	Sends a Measurement Interval indication to a specified remote device.
HTS_Indicate_Temperature_Measurement	Sends a Temperature Measurement indication to a specified remote device.
HTS_Decode_Temperature_Measurement	Parses a value received from a remote HTS Server interpreting it as a

	Temperatue Measurement characteristic.
HTS_Decode_Valid_Range	Parses a value received from a remote HTS Server interpreting it as a Valid Range descriptor.

## HTS\_Initialize\_Service

This function opens a HTS Server on a specified Bluetooth Stack.

### Notes:

1. Only one HTS 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 HTS_Initialize_Service(unsigned int BluetoothStackID,
    HTS_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.
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 HTS service returned from GATT_Register_Service API.

### Return:

Positive non-zero if successful. The return value will be the Service ID of HTS 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:

```
HTS_ERROR_INSUFFICIENT_RESOURCES
HTS_ERROR_SERVICE_ALREADY_REGISTERED
HTS_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.

**HTS\_Cleanup\_Service**

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

**Prototype:**

```
int BTPSAPI HTS_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 HTS_Initialize_Service().

**Return:**

Zero if successful.

Negative if an error occurred. Possible values are:

HTS\_ERROR\_INVALID\_PARAMETER  
HTS\_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.

**HTS\_Set\_Temperature\_Type**

This function is responsible for setting the Temperature Type on the specified HTS Instance.

**Prototype:**

```
int BTPSAPI HTS_Set_Temperature_Type(unsigned int BluetoothStackID, unsigned int  
                                      InstanceID, Byte_t Temperature_Type);
```

**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 DIS_Initialize_Service().
Temperature_Type	The value to be set as the Temperature Type for the specified HTS Instance.

**Return:**

Zero if successful.

Negative if an error occurred. Possible values are:

HTS\_ERROR\_INVALID\_INSTANCE\_ID  
HTS\_ERROR\_INVALID\_PARAMETER  
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.

**HTS\_Query\_Temperature\_Type**

This function is responsible for querying the current Temperature Type on the specified HTS Instance.

**Prototype:**

```
int BTPSAPI HTS_Query_Temperature_Type(unsigned int BluetoothStackID, unsigned  
int InstanceID, Byte_t *Temperature_Type);
```

**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 DIS_Initialize_Service().
Temperature_Type	A pointer to the current Temperature Type for the the specified HTS Instance.

**Return:**

Zero if successful.

Negative if an error occurred. Possible values are:

HTS\_ERROR\_INVALID\_INSTANCE\_ID



HTS\_ERROR\_INVALID\_PARAMETER  
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.

**HTS\_Set\_Measurement\_Interval**

This function is responsible for setting the Measurement Interval on the specified HTS Instance.

**Prototype:**

int BTPSAPI HTS\_Set\_Measurement\_Interval(unsigned int BluetoothStackID, unsigned int InstanceID, Word\_t Measurement\_Interval) ;

**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 DIS_Initialize_Service().
Measurement_Interval	The value to be set as the Measurement Interval for the specified HTS Instance.

**Return:**

Zero if successful.

Negative if an error occurred. Possible values are:

HTS\_ERROR\_INVALID\_INSTANCE\_ID  
HTS\_ERROR\_INVALID\_PARAMETER  
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.

**HTS\_Query\_Measurement\_Interval**

This function is responsible for querying the current Measurement Interval on the specified HTS Instance.

**Prototype:**

```
int BTPSAPI HTS_Query_Measurement_Interval(unsigned int BluetoothStackID,  
      unsigned int InstanceID, Word_t *Measurement_Interval) ;
```

**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 DIS_Initialize_Service().
Measurement_Interval	A pointer to the current Measurement Interval for the specified HTS Instance.

**Return:**

Zero if successful.

Negative if an error occurred. Possible values are:

```
HTS_ERROR_INVALID_INSTANCE_ID  
HTS_ERROR_INVALID_PARAMETER  
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.

**HTS\_Set\_Valid\_Range**

This function is responsible for setting the Valid Range descriptor value on the specified HTS Instance.

**Prototype:**

```
int BTPSAPI HTS_Set_Valid_Range(unsigned int BluetoothStackID, unsigned int  
      InstanceID, HTS_Valid_Range_Data_t *ValidRange);
```

**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 DIS_Initialize_Service().
ValidRange	The value to be set as the Valid Range for the specified HTS Instance. The structure for the Valid Range Data is as follows:

```
typedef struct
{
    Word_t    Lower_Bounds;
    Word_t    Upper_Bounds;
} HTS_Valid_Range_Data_t;
```

**Return:**

Zero if successful.

Negative if an error occurred. Possible values are:

```
HTS_ERROR_INVALID_INSTANCE_ID
HTS_ERROR_INVALID_PARAMETER
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.

**HTS\_Query\_Valid\_Range**

This function is responsible for querying the Valid Range descriptor value on the specified HTS Instance.

**Prototype:**

```
int BTPSAPI HTS_Query_Valid_Range(unsigned int BluetoothStackID, unsigned int
    InstanceID, HTS_Valid_Range_Data_t *ValidRange);
```

**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 DIS_Initialize_Service().
ValidRange	A pointer to store the Valid Range structure for the specified HTS Instance. The structure for the Valid Range Data is as follows:

```
typedef struct
{
    Word_t    Lower_Bounds;
    Word_t    Upper_Bounds;
} HTS_Valid_Range_Data_t;
```

**Return:**

Zero if successful.

Negative if an error occurred. Possible values are:

HTS\_ERROR\_INVALID\_INSTANCE\_ID  
HTS\_ERROR\_INVALID\_PARAMETER  
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.

**HTS\_Read\_Client\_Configuration\_Response**

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

**Prototype:**

```
int BTPSAPI HTS_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 HTS_Initialize_Service().
TransactionID	The Transaction ID of the original read request. This value was received in the etHTS_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:

HTS\_ERROR\_INVALID\_INSTANCE\_ID  
HTS\_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.

**HTS\_Notify\_Intermediate\_Temperature**

The following function is responsible for sending an Intermediate Temperature notification to a specified remote device.

**Prototype:**

```
int BTPSAPI HTS_Notify_Intermediate_Temperature(unsigned int BluetoothStackID,
        unsigned int InstanceID, unsigned int ConnectionID,
        HTS_Temperature_Measurement_Data_t *Temperature_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 HTS_Initialize_Service().
ConnectionID	Connection ID of the currently connected remote client device to send the handle/value notification.
Temperature_Measurement	The Intermediate Temperature data to notify. The Temperature Measurement Data structure is as follows:

```
typedef struct
{
    Byte_t          Flags;
    HTS_Temperature_Data_t    Temperature;
    HTS_Time_Stamp_Data_t    Time_Stamp;
    Byte_t          Temperature_Type;
} HTS_Temperature_Measurement_Data_t;
```

With the Temperature Data Structure and Time Stamp data structure being defined as follows:

```
typedef __PACKED_STRUCT_BEGIN__ struct
{
    NonAlignedByte_t    Value0;
    NonAlignedByte_t    Value1;
    NonAlignedByte_t    Value2;
    NonAlignedByte_t    Exponent;
} __PACKED_STRUCT_END__ HTS_Temperature_Data_t;

typedef struct
{
    Word_t Year;
```

```
    Byte_t Month;  
    Byte_t Day;  
    Byte_t Hours;  
    Byte_t Minutes;  
    Byte_t Seconds;  
} HTS_Time_Stamp_Data_t;
```

**Return:**

Zero if successful.

Negative if an error occurred. Possible values are:

```
HTS_ERROR_INVALID_INSTANCE_ID  
HTS_ERROR_INVALID_PARAMETER  
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.

**HTS\_Indicate\_Measurement\_Interval**

The following function is responsible for sending a Measurement Interval indication to a specified to a specified remote device.

**Prototype:**

```
int BTPSAPI HTS_Indicate_Measurement_Interval(unsigned int BluetoothStackID,  
    unsigned int InstanceID, unsigned int ConnectionID);
```

**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 GLS_Initialize_Service().
ConnectionID	Connection ID of the currently connected remote client device to send the handle/value indication.

**Return:**

Zero if successful.

Negative if an error occurred. Possible values are:

```
HTS_ERROR_INDICATION_OUTSTANDING
HTS_ERROR_INVALID_INSTANCE_ID
HTS_ERROR_INVALID_PARAMETER
BTGATT_ERROR_NOT_INITIALIZED
BTGATT_ERROR_INVALID_BLUETOOTH_STACK_ID
BTGATT_ERROR_INVALID_PARAMETER
```

### Possible Events:

etGATT\_Connection\_Server\_Indication

### 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.

## HTS\_Indicate\_Temperature\_Measurement

The following function is responsible for sending a Temperature Measurement indication to a specified to a specified remote device.

### Note:

Only 1 Temperature Measurement indication may be outstanding per HTS instance.

### Prototype:

```
int BTPSAPI HTS_Indicate_Temperature_Measurement(unsigned int BluetoothStackID,
    unsigned int InstanceID, unsigned int ConnectionID,
    HTS_Temperature_Measurement_Data_t *Temperature_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 GLS _Initialize_Service().
ConnectionID	Connection ID of the currently connected remote client device to send the handle/value indication.
Temperature_Measurement	The temperature measurement data to be indicated. The Temperature Measurement Data struct is as follows:

```
typedef struct
{
    Byte_t                Flags;
    HTS_Temperature_Data_t Temperature;
    HTS_Time_Stamp_Data_t Time_Stamp;
    Byte_t                Temperature_Type;
} HTS_Temperature_Measurement_Data_t;
```

With the Temperature Data Structure and Time Stamp data structure being defined as follows:

```
typedef __PACKED_STRUCT_BEGIN__ struct
{
    NonAlignedByte_t    Value0;
    NonAlignedByte_t    Value1;
    NonAlignedByte_t    Value2;
    NonAlignedByte_t    Exponent;
} __PACKED_STRUCT_END__ HTS_Temperature_Data_t;

typedef struct
{
    Word_t Year;
    Byte_t Month;
    Byte_t Day;
    Byte_t Hours;
    Byte_t Minutes;
    Byte_t Seconds;
} HTS_Time_Stamp_Data_t;
```

#### Return:

Zero if successful.

Negative if an error occurred. Possible values are:

```
HTS_ERROR_INDICATION_OUTSTANDING
HTS_ERROR_INSUFFICIENT_RESOURCES
HTS_ERROR_INVALID_INSTANCE_ID
HTS_ERROR_INVALID_PARAMETER
BTGATT_ERROR_NOT_INITIALIZED
BTGATT_ERROR_INVALID_BLUETOOTH_STACK_ID
BTGATT_ERROR_INVALID_PARAMETER
```

#### Possible Events:

etGATT\_Connection\_Server\_Indication

#### 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.

### HTS\_Decode\_Temperature\_Measurement

The following function is responsible for parsing a value received from a remote HTS Server interpreting it as a Temperature Measurement characteristic.

#### Prototype:

```
int BTPSAPI HTS_Decode_Temperature_Measurement(unsigned int ValueLength,
    Byte_t *Value, HTS_Temperature_Measurement_Data_t *TemperatureMeasurement);
```



**Parameters:**

ValueLength	Specifies the length of the Temperature Measurement Context value returned by the remote HTS Server.
Value	Value is a pointer to the Temperature Measurement Context data returned by the remote HTS Server.
TemperatureMeasurement	A pointer to store the parsed Temperature Measurement value. The Temperature Measurement Data struct is as follows:

```
typedef struct
{
    Byte_t          Flags;
    HTS_Temperature_Data_t  Temperature;
    HTS_Time_Stamp_Data_t  Time_Stamp;
    Byte_t          Temperature_Type;
} HTS_Temperature_Measurement_Data_t;
```

With the Temperature Data Structure and Time Stamp data structure being defined as follows:

```
typedef __PACKED_STRUCT_BEGIN__ struct
{
    NonAlignedByte_t  Value0;
    NonAlignedByte_t  Value1;
    NonAlignedByte_t  Value2;
    NonAlignedByte_t  Exponent;
} __PACKED_STRUCT_END__ HTS_Temperature_Data_t;
```

```
typedef struct
{
    Word_t Year;
    Byte_t Month;
    Byte_t Day;
    Byte_t Hours;
    Byte_t Minutes;
    Byte_t Seconds;
} HTS_Time_Stamp_Data_t;
```

**Return:**

DecodeTemperatureMeasurement(ValueLength, Value, TemperatureMeasurement)

**Possible Events:**

Unknown\XXX

**HTS\_Decode\_Valid\_Range**

The following function is responsible for parsing a value received from a remote HTS Server interpreting it as a Valid Range descriptor.

**Prototype:**

```
int BTPSAPI HTS_Decode_Valid_Range(unsigned int ValueLength, Byte_t *Value,
    HTS_Valid_Range_Data_t *ValidRange);
```

**Parameters:**

ValueLength	Specifies the length of the Valid Range Context value returned by the remote HTS Server.
Value	Value is a pointer to the Valid Range Context data returned by the remote HTS Server.
Valid Range	A pointer to store the parsed Valid Range value. The structure for the Valid Range Data is as follows:

```
typedef struct
{
    Word_t    Lower_Bounds;
    Word_t    Upper_Bounds;
} HTS_Valid_Range_Data_t;
```

**Return:**

DecodeTemperatureMeasurement(ValueLength, Value, TemperatureMeasurement)

**Possible Events:**

Unknown\XXX

## 2.2 Health Thermometer Service Event Callback Prototypes

### 2.2.1 Server Event Callback

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

**HTS\_Event\_Callback\_t**

This The event callback function mentioned in the HTS\_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 HTS Service Event Packets. A Deadlock WILL occur because NO HTS Event Callbacks will be issued while this function is currently outstanding.

**Prototype:**

```
typedef void (BTPSAPI *HTS_Event_Callback_t)(unsigned int BluetoothStackID,
    HTS_Event_Data_t *HTS_Event_Data, unsigned long CallbackParameter);
```

**Parameters:**

**BluetoothStackID**<sup>1</sup> Unique identifier assigned to this Bluetooth Protocol Stack via a call to BSC\_Initialize.

**HTS\_Event\_Data\_t** Data describing the event for which the callback function is called. This is defined by the following structure:

```
typedef struct
{
    HTS_Event_Type_t    Event_Data_Type;
    Word_t              Event_Data_Size;
    union
    {
        HTS_Read_Client_Configuration_Data_t
            *HTS_Read_Client_Configuration_Data;
        HTS_Client_Configuration_Update_Data_t
            *HTS_Client_Configuration_Update_Data;
        HTS_Measurement_Interval_Update_Data_t
            *HTS_Measurement_Interval_Update_Data;
        HTS_Confirmation_Data_t
            *HTS_Confirmation_Data;
    } Event_Data;
} HTS_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 Health Thermometer Service Events

The Health Thermometer Service contains events that are received by the Server. The following sections detail those events.

### 2.3.1 Health Thermometer Service Server Events

The possible Health Thermometer 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
etHTS_Server_Read_Client_Configuration_Request	Dispatched to a HTS Server when a HTS Client is attempting to read a descriptor.
etHTS_Server_Client_Configuration_Update	Dispatched to a HTS Server when a HTS Client has written a Client Configuration descriptor.
etHTS_Measurement_Interval_Update	Dispatched to a HTS Server when a HTS Client is attempting to write the Measurement Interval Characteristic.
etHTS_Confirmation_Response	Dispatched to a HTS Server when a HTS client has sent a confirmation response to a previously sent confirmation request.

### etHTS\_Server\_Read\_Client\_Configuartion\_Request

The following HTS Profile Event is dispatched to a HTS Server when a HTS 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;
    HTS_Characteristic_Type_t ClientConfigurationType;
} HTS_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 HTS 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	Specifies the valid Read Request types that a server may receive in an etHTS_Server_Read_Client_Configuration_Request or etHTS_Server_Client_Configuration_Update event. This is also used by the HTS_Send_Notification to denote the characteristic value to notify.
-------------------------	---

### etHTS\_Server\_Client\_Configuration\_Update

The following HTS Profile Event is dispatched to a HTS Server when a HTS Client has written a Client Configuration descriptor.

#### Return Structure:

```
typedef struct
{
    unsigned int          InstanceID;
    unsigned int          ConnectionID;
    GATT_Connection_Type_t ConnectionType;
    BD_ADDR_t            RemoteDevice;
    HTS_Characteristic_Type_t ClientConfigurationType;
    Word_t               ClientConfiguration;
} HTS_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 HTS 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	Specifies the valid Read Request types that a server may receive in an etHTS_Server_Read_Client_Configuration_Request or etHTS_Server_Client_Configuration_Update event. This is also used by the HTS_Send_Notification to denote the characteristic value to notify.
ClientConfiguration	The New Client Configuration for the specified characteristic.

### etHTS\_Measurement\_Interval\_Update

The following HTS Profile Event is dispatched to a HTS Server when a HTS Client is attempting to write the Measurement Interval characteristic.

**Return Structure:**

```
typedef struct
{
    unsigned int          InstanceID;
    unsigned int          ConnectionID;
    GATT_Connection_Type_t ConnectionType;
    BD_ADDR_t             RemoteDevice;
    Word_t                NewMeasurementInterval;
} HTS_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 HTS 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.
NewMeasurementInterval	The new Measurement Interval that is to be written.

**etHTS\_Confirmation\_Response**

The following HTS Profile Event is dispatched to a HTS Server when a HTS client has sent a confirmation response to a previously sent confirmation request.

**Return Structure:**

```
typedef struct
{
    unsigned int          InstanceID;
    unsigned int          ConnectionID;
    GATT_Connection_Type_t ConnectionType;
    BD_ADDR_t             RemoteDevice;
    HTS_Characteristic_Type_t Characteristic_Type;
    Byte_t                Status;
} HTS_Read_Reference_Time_Information_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 HTS 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.
Characteristic_Type	Specifies which Indication the Client has sent a confirmation for. This parameter will NEVER be set to ctIntermediateTemperature for this event.
Status	This specifies the status of the indication. This is set to one of the following values: GATT_CONFIRMATION_STATUS_SUCCESS GATT_CONFIRMATION_STATUS_TIMEOUT

### 3. File Distributions

The header files that are distributed with the Bluetooth Health Thermometer Service Library are listed in the table below

File	Contents/Description
HTSAPI.h	Bluetooth Health Thermometer Service (GATT based) API Type Definitions, Constants, and Prototypes.
HTSTypes.h	Bluetooth Health Thermometer Service Types.
SS1BTHTS.h	Bluetooth Health Thermometer Service Include file