



Cycling Speed and Cadence Service (CSCS)

Application Programming Interface Reference Manual

Profile Version: 1.0

**Release: 4.0.2.0
November 25, 2014**

www.ti.com

Table of Contents

1. INTRODUCTION.....	3
1.1 Scope	3
1.2 Applicable Documents	4
1.3 Acronyms and Abbreviations	4
2. CSCS PROGRAMMING INTERFACE.....	5
2.1 Cycling Speed and Cadence Service Commands	5
CSCS_Initialize_Service.....	7
CSCS_Cleanup_Service	8
CSCS_Query_Number_Attributes.....	9
CSCS_Query_Sensor_Location.....	10
CSCS_Read_Response_For_Read_Client_Configuration	11
CSCS_Measurements_Notification	12
CSCS_Set_Supported_Features.....	14
CSCS_Query_Supported_Features.....	15
CSCS_SC_Control_Point_Indication	16
CSCS_Set_Sensor_Location_List	18
CSCS_Decode_Cycle_Speed_and_Cadence_Measurements	19
CSCS_Decode_SC_Control_Point_Indication	20
CSCS_Format_Control_Point_Command	22
2.2 Cycling Speed and Cadence Service Event Callback Prototypes	24
2.2.1 SERVER EVENT CALLBACK	24
CSCS_Event_Callback_t	24
2.3 Cycling Speed and Cadence Service Events	26
2.3.1 CYCLING SPEED AND CADENCE SERVICE SERVER EVENTS	26
etCSCS_Server_Read_Client_Configuration_Request	27
etCSCS_Server_Client_Configuration_Update.....	28
etCSCS_Confirmation_Response	29
etCSCS_Server_SC_Control_Point_Command	30
Additional structures.....	31
3. FILE DISTRIBUTIONS.....	33

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) and several of the Bluetooth Profiles. Program access to these layers, services, and profiles is handled via Application Programming Interface (API) calls.

Texas Instruments Incorporated provides the CSCS (Cycling Speed and Cadence Service) that is based on Bluetopia®+LE stack.

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 Cycling Speed and Cadence Service Profile Stack provided by Bluetopia®+LE Single. And, Chapter 3 contains the header file name list for the Cycling Speed and Cadence Service library.

1.1 Scope

This reference manual provides information on the CSCS API.

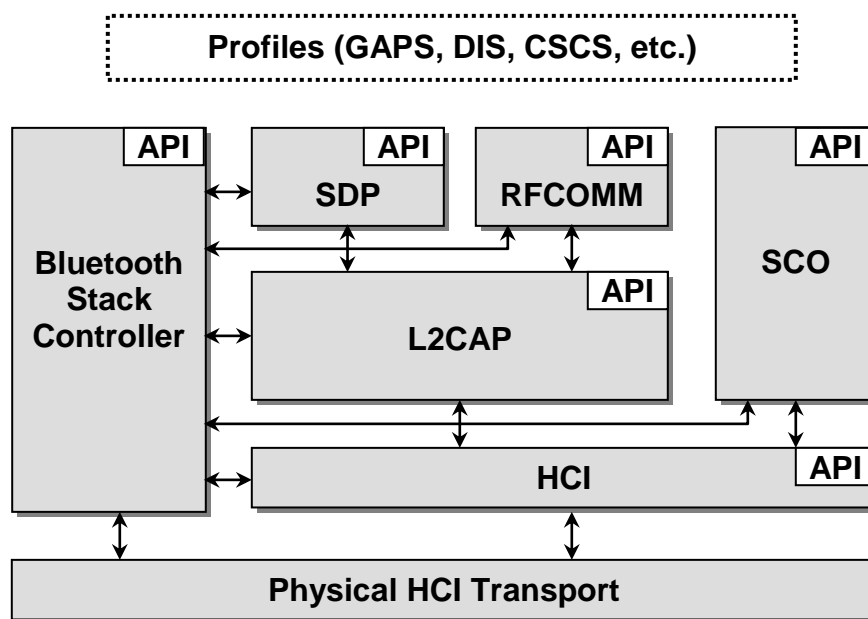


Figure 1-1 The Stonestreet One Bluetooth Protocol Stack

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. *Bluetooth® Protocol Stack, Application Programming Interface Reference Manual*, version 4.0.1, January 10, 2013.
4. *Bluetooth Cycling Speed and Cadence Service Specification*, version v10r00, Aug 21, 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
CSC	Cycling Speed and Cadence
CSCS	Cycling Speed and Cadence Service
HCI	Host Controller Interface
HS	High Speed
L2CAP	Logical Link Control and Adaptation Protocol
LE	Low Energy
LSB	Least Significant Bit
MSB	Most Significant Bit
SC	Speed and Cadence

2. CSCS Programming Interface

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

2.1 Cycling Speed and Cadence Service Commands

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

Server Commands	
Function	Description
CSCS_Initialize_Service	Opens a CSCS Server.
CSCS_Cleanup_Service	Closes an opened CSCS Server.
CSCS_Query_Number_Attributes	Queries the number of attributes on the specified CSCS Service
CSCS_Query_Sensor_Location	Queries the current Sensor Location on the specified CSCS Instance.
CSCS_Read_Response_For_Read_Client_Configuration	Responds to a CSCS Read Client Configuration Request.
CSCS_Measurements_Notification	Sends a CSC Measurement notification to a specified remote device.
CSCS_Set_Supported_Features	Sets the Supported Features on the specified CSCS Instance.
CSCS_Query_Supported_Features	Queries the Supported Features on the specified CSCS Instance.
CSCS_SC_Control_Point_Indication	Formats a SC Control Point Indication into user specified buffer and send it.
CSCS_Set_Sensor_Location_List	Sets the Supported Sensor locations on the specified CSCS Instance.
CSCS_Format_Control_Point_Command	Formats a SC Control Point Command into user specified buffer.

CSCS_Decode_Cycle_Speed_and_Cadence_Measurements	Parses a value received from a remote CSCS Server interpreting it as a CSC Measurement characteristic.
CSCS_Decode_SC_Control_Point_Indication	Parses a value received from a remote CSCS Server interpreting it as a SC Control Point characteristic.

CSCS_Initialize_Service

The following function is responsible for opening a CSCS Server on a specified Bluetooth Stack, also for initiating the service features to all features disable and setting the sensor location to other.

Notes:

1. Only one CSCS 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 CSCS_Initialize_Service(unsigned int BluetoothStackID,  
    CSCS_Event_Callback_t EventCallback, unsigned long CallbackParameter, unsigned int  
    *ServiceID);
```

Parameters:

BluetoothStackID	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 CSCS service returned from GATT_Register_Service API.

Return:

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

```
CSCS_ERROR_INSUFFICIENT_RESOURCES  
CSCS_ERROR_SERVICE_ALREADY_REGISTERED  
CSCS_ERROR_INVALID_PARAMETER  
BTGATT_ERROR_INVALID_SERVICE_TABLE_FORMAT  
BTGATT_ERROR_INSUFFICIENT_RESOURCES  
BTGATT_ERROR_INSUFFICIENT_HANDLES  
BTGATT_ERROR_INVALID_PARAMETER  
BTGATT_ERROR_INVALID_BLUETOOTH_STACK_ID  
BTGATT_ERROR_NOT_INITIALIZED  
BTPS_ERROR_FEATURE_NOT_AVAILABLE
```

CSCS_Cleanup_Service

This function is responsible for cleaning up and freeing all resources associated with a Cycling Speed and Cadence Service Instance. After this function is called, no other Cycling Speed and Cadence Service function can be called until after a successful call to the CSCS_Initialize_Service() function is performed.

Prototype:

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

Parameters:

BluetoothStackID	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 CSCS_Initialize_Service().

Return:

Zero if successful.

Negative if an error occurred. Possible values are:

```
CSCS_ERROR_INVALID_PARAMETER  
CSCS_ERROR_INVALID_INSTANCE_ID
```


CSCS_Query_Number_Attributes

This function is responsible for querying the number of attributes on the specified CSCS Service

Prototype:

```
unsigned int BTPSAPI CSCS_Query_Number_Attributes(void)
```

Parameters:

None

Return:

The number of attributes on the CSCS Service

CSCS_Query_Sensor_Location

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

Prototype:

```
int BTPSAPI CSCS_Query_Sensor_Location(unsigned int BluetoothStackID, unsigned int InstanceID, Byte_t *Sensor_Location)
```

Parameters:

BluetoothStackID	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 CSCS_Initialize_Service().
Body_Sensor_Location	A pointer to return the current Body Sensor Location for the specified CSCS Instance.

Return:

Zero if successful.

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

CSCS_ERROR_INVALID_INSTANCE_ID
CSCS_ERROR_INVALID_PARAMETER

CSCS_Read_Response_For_Read_Client_Configuration

The following function is responsible for responding to a CSCS Read Client Configuration Request from CSC Measurements or SC Control Point characteristics.

Prototype:

```
int BTPSAPI CSCS_Read_Response_For_Read_Client_Configuration(unsigned int  
    BluetoothStackID, unsigned int InstanceID, unsigned int TransactionID, Word_t  
    Client_Configuration)
```

Parameters:

BluetoothStackID	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 CSCS_Initialize_Service().
TransactionID	The Transaction ID of the original read request. This value was received in the etCSCS_Server_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:

```
CSCS_ERROR_INVALID_INSTANCE_ID  
CSCS_ERROR_INVALID_PARAMETER  
BTGATT_ERROR_NOT_INITIALIZED  
BTGATT_ERROR_INVALID_BLUETOOTH_STACK_ID  
BTGATT_ERROR_INVALID_TRANSACTION_ID  
BTGATT_ERROR_INVALID_PARAMETER  
BTPS_ERROR_FEATURE_NOT_AVAILABLE
```

Possible Events:

etGATT_Client_Read_Response

CSCS_Measurements_Notification

The following function is responsible for sending a CSC Measurement notificaiton to a specified remote device.

Prototype:

```
int BTPSAPI CSCS_Measurements_Notification(unsigned int BluetoothStackID, unsigned
    int InstanceID, unsigned int ConnectionID, CSCS_Measurements_Data_t
    *CSCS_Measurement)
```

Parameters:

BluetoothStackID	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 CSCS_Initialize_Service().
ConnectionID	Connection ID of the currently connected remote client device to send the handle/value notification.
CSCS_Measurement	The CSC Measurement data to notify. The CSC Measurement Data structure is as follows:

```
typedef struct _tagCSCS_Measurements_Data_t
{
    Byte_t    Flags;
    CSCS_Wheel_Data_t  *Wheel_Data;
    CSCS_Crank_Data_t  *Crank_Data;
} CSCS_Measurements_Data_t;
```

```
typedef struct _tagCSCS_Wheel_Data_t
{
    DWord_t    Cumulative_Wheel_Revolutions;
    Word_t     Last_Wheel_Event_Time;
} CSCS_Wheel_Data_t;
```

```
typedef struct _tagCSCS_Crank_Data_t
{
    Word_t     Cumulative_Crank_Revolutions;
    Word_t     Last_Crank_Event_Time;
} CSCS_Crank_Data_t;
```

Return:

Zero if successful.

Negative if an error occurred. Possible values are:

```
CSCS_ERROR_MALFORMATTED_DATA
CSCS_ERROR_INSUFFICIENT_RESOURCES
CSCS_ERROR_INVALID_INSTANCE_ID
CSCS_ERROR_INVALID_PARAMETER
BTGATT_ERROR_INVALID_HANDLE_VALUE
```

BTGATT_ERROR_INVALID_CONNECTION_ID
BTGATT_ERROR_NOT_INITIALIZED
BTGATT_ERROR_INVALID_BLUETOOTH_STACK_ID
BTGATT_ERROR_INVALID_PARAMETER
BTPS_ERROR_FEATURE_NOT_AVAILABLE

Possible Events:

etGATT_Connection_Server_Notification

CSCS_Set_Supported_Features

This function is responsible for setting the supported features that are available on the specified CSCS Instance.

Prototype:

```
int BTPSAPI CSCS_Set_Supported_Features(unsigned int BluetoothStackID, unsigned int InstanceID, Word_t SupportedFeaturesMask)
```

Parameters:

BluetoothStackID ¹	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 CSCS_Initialize_Service().
SupportedFeaturesMask	The value for which the CSC_Features should be set for the specified CSCS Instance. The parameter should be an enumerated value of the form CSCS_CSC_FEATURE_BIT_MASK_XXX

Return:

Zero if successful.

Negative if an error occurred. Possible values are:

```
CSCS_ERROR_INVALID_INSTANCE_ID  
CSCS_ERROR_INVALID_PARAMETER  
CSCS_ERROR_MALFORMATTED_DATA  
CSCS_ERROR_WHEEL_NOT_SUPPORTED  
CSCS_ERROR_CRANK_NOT_SUPPORTED  
CSCS_ERROR_WHEEL_AND_CRANK_NOT_SUPPORTED  
CSCS_ERROR_MULTIPLE_LOCATION_NOT_SUPPORTED  
CSCS_ERROR_WHEEL_AND_MULTIPLE_LOCATION_NOT_SUPPORTED  
CSCS_ERROR_CRANK_AND_MULTIPLE_LOCATION_NOT_SUPPORTED  
CSCS_ERROR_WHEEL_AND_CRANK_AND_MULTIPLE_LOCATION_NOT_SUPPORTED  
CSCS_ERROR_MALFORMATTED_DATA
```

CSCS_Query_Supported_Features

This function is responsible for querying the current Supported Features on the specified CSCS Instance.

Prototype:

```
int BTPSAPI CSCS_Query_Supported_Features(unsigned int BluetoothStackID, unsigned  
int InstanceID, Word_t *Features)
```

Parameters:

BluetoothStackID	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 CSCS_Initialize_Service().
Features	A pointer to return the current Body Sensor Location for the specified CSCS Instance.

Return:

Zero if successful.

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

CSCS_ERROR_INVALID_INSTANCE_ID
CSCS_ERROR_INVALID_PARAMETER

CSCS_SC_Control_Point_Indication

This function is responsible for querying the current Supported Features on the specified CSCS Instance.

Prototype:

```
int BTPSAPI CSCS_SC_Control_Point_Indication(unsigned int BluetoothStackID,
      unsigned int InstanceID, unsigned int ConnectionID, CSCS_Control_Point_Data_t
      *Op_Code_Response)
```

Parameters:

BluetoothStackID	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 CSCS_Initialize_Service().
ConnectionID	Connection ID of the currently connected remote client device to send the handle/value notification.
Op_Code_Response	The SC Control Point data to indicate. The SC Control Point structure is as follows:

```
tagCSCS_Control_Point_Data_t
{
    CSCS_Connection_Header_Data_t    ConnectionHeader;
    Byte_t                           Op_Code;
    union _tagCommand_data_Buffer
    {
        DWord_t                      Cumulative_Value;
        Byte_t                       Sensor_Location_Value;
        CSCS_Control_Point_Indication_Data_t    Indication;
    } Command_data_Buffer;
} CSCS_Control_Point_Data_t;
```

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

```
typedef struct _tagCSCS_Control_Point_Indication_Data_t
{
    Byte_t                           Request_Op_Code;
    unsigned int                     Response_Value;
    Byte_t                           Number_Of_Parameters;
    Byte_t                           Response_Parameter[1];
} CSCS_Control_Point_Indication_Data_t;
```


Return:

Zero if successful.

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

CSCS_ERROR_INVALID_PARAMETER
CSCS_ERROR_INSUFFICIENT_RESOURCES
CSCS_ERROR_INVALID_INSTANCE_ID
CSCS_ERROR_MALFORMATTED_DATA
CSCS_ERROR_INDICATION_IN_PROGRESS
CSCS_ERROR_LOCATION_LIST_IS_GREATER_THAN_
MAXIMUM_SIZE
BTPS_ERROR_FEATURE_NOT_AVAILABLE
BTGATT_ERROR_INVALID_PARAMETER
BTGATT_ERROR_INSUFFICIENT_RESOURCES
BTGATT_ERROR_INVALID_HANDLE_VALUE

Possible Events:

etGATT_Connection_Server_Indication.

CSCS_Set_Sensor_Location_List

The following function is responsible for setting the sensor location list on a specified remote device.

Prototype:

```
int BTPSAPI CSCS_Set_Sensor_Location_List(unsigned int BluetoothStackID, unsigned  
int InstanceID, Word_t SensorListBitMask)
```

Parameters:

BluetoothStackID	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 CSCS_Initialize_Service().
SensorListBitMask	A bit mask that holds the supported sensor location list. The parameter should be an enumerated value of the form CSC_SENSOR_LOCATION_XXX_BIT_MASK.

Return:

Zero if successful.

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

```
Function_Error  
INVALID_STACK_ID_ERROR  
CSCS_ERROR_INVALID_PARAMETER  
CSCS_ERROR_CRANK_AND_MULTIPLE_LOCATION_NOT_  
SUPPORTED
```

Possible Events:

Unknown\XXX

CSCS_Decode_Cycle_Speed_and_Cadence_Measurements

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

Prototype:

```
int BTPSAPI CSCS_Decode_Cycle_Speed_and_Cadence_Measurements(unsigned int
    ValueLength, Byte_t *Value, CSCS_Measurements_Data_t
    *CSCS_Measurement)Parameters:
```

ValueLength	Specifies the length of the CSC Measurement value returned by the remote CSCS Server.
Value	Value is a pointer to the CSC Measurement data returned by the remote CSCS Server.
CSCS_Measurement	A pointer to store the parsed CSC Measurement value. The CSC Measurement Data structure is as follows:

```
typedef struct _tagCSCS_Measurements_Data_t
{
    Byte_t    Flags;
    CSCS_Wheel_Data_t  *Wheel_Data;
    CSCS_Crank_Data_t  *Crank_Data;
} CSCS_Measurements_Data_t;
```

```
typedef struct _tagCSCS_Wheel_Data_t
{
    DWord_t    Cumulative_Wheel_Revolutions;
    Word_t     Last_Wheel_Event_Time;
} CSCS_Wheel_Data_t;
```

```
typedef struct _tagCSCS_Crank_Data_t
{
    Word_t     Cumulative_Crank_Revolutions;
    Word_t     Last_Crank_Event_Time;
} CSCS_Crank_Data_t;
```

Return:

Zero if successful.

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

```
CSCS_ERROR_INVALID_PARAMETER
CSCS_ERROR_MALFORMATTED_DATA
```

Possible Events:

Unknown\XXX

CSCS_Decode_SC_Control_Point_Indication

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

Prototype:

```
int BTPSAPI CSCS_Decode_SC_Control_Point_Indication(Word_t ValueLength, Byte_t
    *Value, CSCS_Control_Point_Data_t *CSCS_Control_Point);
```

Parameters:

ValueLength	Specifies the length of the SC Control point indication value returned by the remote CSCS Server.
Value	Value is a pointer to the SC Control point indication data returned by the remote CSCS Server.
CSCS_Control_Point	A pointer to store the parsed SC Control Point value that was used as indication by the remote CSCS Server. The SC Control Point Data structure is as follows: <pre>typedef struct _tagCSCS_Control_Point_Data_t { CSCS_Connection_Header_Data_t ConnectionHeader; Byte_t Op_Code; union _tagCommand_data_Buffer { DWord_t Cumulative_Value; Byte_t Sensor_Location_Value; CSCS_Control_Point_Indication_Data_t Indication; } Command_data_Buffer; } CSCS_Control_Point_Data_t;</pre> <pre>typedef struct _tagCSCS_Control_Point_Indication_Data_t { Byte_t Request_Op_Code; unsigned int Response_Value; Byte_t Number_Of_Parameters; Byte_t Response_Parameter[1]; } CSCS_Control_Point_Indication_Data_t;</pre>

Return:

Zero if successful.

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

```
CSCS_ERROR_INVALID_PARAMETER
CSCS_ERROR_MALFORMATTED_DATA
CSCS_ERROR_LOCATION_LIST_IS_GREATER_THEN_
MAXIMUM_SIZE
BTPS_ERROR_FEATURE_NOT_AVAILABLE
```

Possible Events:

Unknown\XXX

CSCS_Format_Control_Point_Command

The following function is responsible for formatting a SC Control Point Command into a user specified buffer.

Prototype:

```
int BTPSAPI CSCS_Format_Control_Point_Command(unsigned int BufferLength,
Byte_t *Buffer, CSCS_Control_Point_Data_t *CSCS_Control_Point)
```

Parameters:

BufferLength	Specifies the Length of the Buffer
Buffer	A pointer, pointing to memory of size BufferLength to store the SC Control Point request Data after formatting.
CSCS_Control_Point	A pointer to store the parsed SC Control Point value that was used as indication by the remote CSCS Server. The SC Control Point Data structure is as follows:

```
typedef struct _tagCSCS_Control_Point_Data_t
{
    CSCS_Connection_Header_Data_t    ConnectionHeader;
    Byte_t                           Op_Code;
    union _tagCommand_data_Buffer
    {
        DWord_t                      Cumulative_Value;
        Byte_t                       Sensor_Location_Value;
        CSCS_Control_Point_Indication_Data_t  Indication;
    } Command_data_Buffer;
} CSCS_Control_Point_Data_t;
```

```
typedef struct _tagCSCS_Control_Point_Indication_Data_t
{
    Byte_t                           Request_Op_Code;
    unsigned int                     Response_Value;
    Byte_t                           Number_Of_Parameters;
    Byte_t                           Response_Parameter[1];
} CSCS_Control_Point_Indication_Data_t;
```

Return:

Zero if successful.

Negative if an error occurred. Possible values are:

```
CSCS_ERROR_INVALID_PARAMETER
CSCS_ERROR_MALFORMATTED_DATA
CSCS_ERROR_LOCATION_LIST_IS_GREATER_THEN_
MAXIMUM_SIZE
BTPS_ERROR_FEATURE_NOT_AVAILABLE
```

Possible Events:

Unknown/XXX

2.2 Cycling Speed and Cadence Service Event Callback Prototypes

2.2.1 Server Event Callback

The event callback function mentioned in the `CSCS_Initialize_Service` command accepts the callback function described by the following prototype.

CSCS_Event_Callback_t

The event callback function mentioned in the `CSCS_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 CSCS Service Event Packets. A Deadlock **WILL** occur because **NO** CSCS Event Callbacks will be issued while this function is currently outstanding.

Prototype:

```
typedef void (BTPSAPI *CSCS_Event_Callback_t)(unsigned int BluetoothStackID,
      CSCS_Event_Data_t *CSCS_Event_Data, unsigned long CallbackParameter);
```

Parameters:

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

```
typedef struct _tagCSCS_Event_Data_t
{
    CSCS_Event_Type_t  Event_Data_Type;
    Word_t             Event_Data_Size;
    union
    {
        CSCS_Read_Client_Configuration_Data_t
            *CSCS_Read_Client_Configuration_Data;
        CSCS_Client_Configuration_Update_Data_t
            *CSCS_Client_Configuration_Update_Data;
        CSCS_Control_Point_Data_t
            *CSCS_Control_Point_Data;
        CSCS_Confirmation_Data_t
            *CSCS_Confirmation_Data;
    } Event_Data;
} CSCS_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

2.3 Cycling Speed and Cadence Service Events

The Cycling Speed and Cadence Service contain events that are received by the Server. The following sections detail those events.

2.3.1 Cycling Speed and Cadence Service Server Events

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

Server Commands	
Event	Description
etCSCS_Server_Read_Client_Configuration_Request	Dispatched to a CSCS Server when a CSCS Client is attempting to read a descriptor.
etCSCS_Server_Client_Configuration_Update	Dispatched to a CSCS Server when a CSCS Client has written a Client Configuration descriptor.
etCSCS_Confirmation_Response	Dispatched to a CSCS Server when a CSCS client sends a confirmation in response to SC Control Point Indication.
etCSCS_Server_SC_Control_Point_Command	Dispatched to a CSCS Server when a CSCS client sends a request to write SC Control Point data.

etCSCS_Server_Read_Client_Configuration_Request

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

Return Structure:

```
typedef struct _tagCSCS_Read_Client_Configuration_Data_t
{
    CSCS_Connection_Header_Data_t  ConnectionHeader;
    unsigned int                   TransactionID;
    CSCS_Characteristic_Type_t      ClientConfigurationType;
} CSCS_Read_Client_Configuration_Data_t;
```

Event Parameters:

ConnectionHeader	This data structure is described in the end of this section.
TransactionID	The TransactionID identifies the transaction between a client and server. This identifier should be used to respond to the current request.
ClientConfigurationType	The specified configuration type of the client as defined by the enum:

```
typedef enum
{
    ctCyclingSpeedandCadenceMeasurement,
    ctSCControlPoint
} CSCS_Characteristic_Type_t;
```

etCSCS_Server_Client_Configuration_Update

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

Return Structure:

```
typedef struct _tagCSCS_Client_Configuration_Update_Data_t
{
    CSCS_Connection_Header_Data_t    ConnectionHeader;
    CSCS_Characteristic_Type_t        ClientConfigurationType;
    Word_t                            ClientConfiguration;
} CSCS_Client_Configuration_Update_Data_t;
```

Event Parameters:

ConnectionHeader	This data structure is described in the end of this section.
ClientConfigurationType	The specified configuration type of the client as defined by the enum: <pre>typedef enum { ctCyclingSpeedandCadenceMeasurement, ctSCControlPoint } CSCS_Characteristic_Type_t;</pre>
ClientConfiguration	The New Client Configuration for the specified characteristic.

etCSCS_Confirmation_Response

The following CSCS Profile Event is dispatched to a CSCS Server when a CSCS client sends a confirmation in response to SC Control Point Indication.

Return Structure:

```
typedef struct _tagCSCS_Confirmation_Data_t
{
    CSCS_Connection_Header_Data_t  ConnectionHeader;
    CSCS_Characteristic_Type_t      Characteristic_Type;
    Byte_t                          Status;
} CSCS_Confirmation_Data_t;
```

Event Parameters:

ConnectionHeader	This data structure is described in the end of this section.
ClientConfigurationType	The specified configuration type of the client as defined by the enum: <pre>typedef enum { ctCyclingSpeedandCadenceMeasurement, ctSCControlPoint } CSCS_Characteristic_Type_t;</pre>
Status	This variable Holds the status of the response.

etCSCS_Server_SC_Control_Point_Command

The following CSCS Profile Event is dispatched to a CSCS Server when a CSCS Client has sent a SC Control Point Command.

Return Structure:

```
typedef struct _tagCSCS_Control_Point_Data_t
{
    CSCS_Connection_Header_Data_t      ConnectionHeader;
    unsigned int                       TransactionID;
    Word_t                             AttributeOffset;

    Byte_t                             Op_Code;
    union _tagCommand_data_Buffer
    {
        DWord_t                       Cumulative_Value;
        Byte_t                         Sensor_Location_Value;
        CSCS_Control_Point_Indication_Data_t Indication;
    } Command_data_Buffer;
} CSCS_Control_Point_Data_t;
```

Event Parameters:

ConnectionHeader	This data structure is described in the end of this section.
TransactionID	The TransactionID identifies the transaction between a client and server. This identifier should be used to respond to the current request.
AttributeOffset	The AttributeOffset identifies the Offset of the Attribute that the client is trying to change. This identifier should be used to respond to the current request.
Op_Code	The SC Control Point Command that the client sent. The SC Control Point Op code that are in use for this event are as follows:

SET_CUMULATIVE_VALUE

UPDATE_SENSOR_LOCATION

REQUEST_SUPPORTED_SENSOR_LOCATION

Command_data_Buffer union:

Cumulative_Value	This Value holds the Cumulative value entered by the client when using SET_CUMULATIVE_VALUE op code
Sensor_Location_Value	This Value holds the Location value entered by the client when using UPDATE_SENSOR_LOCATION op code
Indication	This data structure holds the indication values that are sent by the server as a response to SC Control Point command.

Additional structures

CSCS_Control_Point_Indication_Data_t

This data structure holds the indication values that are sent by the server as a response to SC Control Point command.

Structure:

```
typedef struct _tagCSCS_Control_Point_Indication_Data_t
{
    Byte_t          Request_Op_Code;
    unsigned int    Response_Value;
    Byte_t          Number_Of_Parameters;
    Byte_t          Response_Parameter[1];
} CSCS_Control_Point_Indication_Data_t;
```

Parameters:

Request_Op_Code	This variable holds the SC Control Point op code that was sent from the client.
Response_Value	This variable holds the response status for the SC Control Point op code that was sent from the client. Available options: SUCCESS OP_CODE_NOT_SUPPORTED INVALID_PARAMETER OPERATION_FAILED
Number_Of_Parameters	If the SC Control Point op code is REQUEST_SUPPORTED_SENSOR_LOCATION, this value holds the size of the sensor location list.
Response_Parameter	If the SC Control Point op code is REQUEST_SUPPORTED_SENSOR_LOCATION, this array holds the sensor location list.

CSCS_Connection_Header_Data_t

This data structure holds the Connection parameters.

Structure:

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

Parameters:

InstanceID	Identifies the Local Server Instance to which the Remote Client has connected.
ConnectionID	Connection ID of the currently connected remote CSCS 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.

3. File Distributions

The header files that are distributed with the Bluetooth Cycling Speed and Cadence Service Library are listed in the table below

File	Contents/Description
CSCSAPI.h	Bluetooth Cycling Speed and Cadence Service (GATT based) API Type Definitions, Constants, and Prototypes.
CSCSTypes.h	Bluetooth Cycling Speed and Cadence Service Types.
SS1BTCSCS.h	Bluetooth Cycling Speed and Cadence Service Include file