[MS-TCC]:

Tethering Control Channel Protocol

Intellectual Property Rights Notice for Open Specifications Documentation

- **Technical Documentation.** Microsoft publishes Open Specifications documentation for protocols, file formats, languages, standards as well as overviews of the interaction among each of these technologies.
- **Copyrights**. This documentation is covered by Microsoft copyrights. Regardless of any other terms that are contained in the terms of use for the Microsoft website that hosts this documentation, you may make copies of it in order to develop implementations of the technologies described in the Open Specifications and may distribute portions of it in your implementations using these technologies or your documentation as necessary to properly document the implementation. You may also distribute in your implementation, with or without modification, any schema, IDL's, or code samples that are included in the documentation. This permission also applies to any documents that are referenced in the Open Specifications.
- **No Trade Secrets**. Microsoft does not claim any trade secret rights in this documentation.
- **Patents**. Microsoft has patents that may cover your implementations of the technologies described in the Open Specifications. Neither this notice nor Microsoft's delivery of the documentation grants any licenses under those or any other Microsoft patents. However, a given Open Specification may be covered by Microsoft Open Specification Promise or the Community Promise. If you would prefer a written license, or if the technologies described in the Open Specifications are not covered by the Open Specifications Promise or Community Promise, as applicable, patent licenses are available by contacting iplg@microsoft.com.
- **Trademarks**. The names of companies and products contained in this documentation may be covered by trademarks or similar intellectual property rights. This notice does not grant any licenses under those rights. For a list of Microsoft trademarks, visit www.microsoft.com/trademarks.
- **Fictitious Names**. The example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted in this documentation are fictitious. No association with any real company, organization, product, domain name, email address, logo, person, place, or event is intended or should be inferred.

Reservation of Rights. All other rights are reserved, and this notice does not grant any rights other than specifically described above, whether by implication, estoppel, or otherwise.

Tools. The Open Specifications do not require the use of Microsoft programming tools or programming environments in order for you to develop an implementation. If you have access to Microsoft programming tools and environments you are free to take advantage of them. Certain Open Specifications are intended for use in conjunction with publicly available standard specifications and network programming art, and assumes that the reader either is familiar with the aforementioned material or has immediate access to it.

Revision Summary

Date	Revision History	Revision Class	Comments
8/8/2013	1.0	New	Released new document.
11/14/2013	1.1	Minor	Clarified the meaning of the technical content.
2/13/2014	2.0	Major	Significantly changed the technical content.
5/15/2014	2.0	None	No change to the meaning, language, or formatting of the technical content.
6/30/2015	3.0	Major	Significantly changed the technical content.
10/16/2015	4.0	Major	Significantly changed the technical content.

Table of Contents

1 Introduction

This document specifies the Tethering Control Channel Protocol [MS-TCC] which facilitates the sharing of a server's network connection with one or more clients. By using the Tethering Control Channel Protocol, clients can request to share a server's Internet connection. The server responds to clients with the appropriate Wi-Fi information that specifies the tethering configuration settings.

Sections 1.8, 2, and 3 of this specification are normative and can contain the terms MAY, SHOULD, MUST, MUST NOT, and SHOULD NOT as defined in [RFC2119]. Sections 1.5 and 1.9 are also normative but do not contain those terms. All other sections and examples in this specification are informative.

1.1 Glossary

The following terms are specific to this document:

ASCII: The American Standard Code for Information Interchange (ASCII) is an 8-bit character-encoding scheme based on the English alphabet. ASCII codes represent text in computers, communications equipment, and other devices that work with text. ASCII refers to a single 8-bit ASCII character or an array of 8-bit ASCII characters with the high bit of each character set to zero.

basic service set identifier (BSSID): A 48-bit structure that is used to identify an entity such as the access point in a wireless network. This is typically a MAC address.

globally unique identifier (GUID): A term used interchangeably with universally unique identifier (UUID) in Microsoft protocol technical documents (TDs). Interchanging the usage of these terms does not imply or require a specific algorithm or mechanism to generate the value. Specifically, the use of this term does not imply or require that the algorithms described in [RFC4122] or [C706] must be used for generating the **GUID**. See also universally unique identifier (UUID).

paired relationship: In a Bluetooth communication scenario, two devices that have established a relationship through the creation of a shared secret known as a link key. The link key enables confirmation of device identity and is used to maintain security across devices.

passphrase: One or more words entered as a security setting to enable device or identity authentication.

radio frequency communications (RFCOMM): A protocol that provides serial port emulation of EIA-232 (formerly RS-232) control signals over the Bluetooth baseband layer. RFCOMM is used to create a virtual serial data stream to enable binary data transport.

Service Discovery Protocol (SDP): This protocol allows a device to discover services (and their associated configuration settings) offered by other devices. A service is identified by a universally unique identifier (UUID) where recognized services, such as Bluetooth profiles, are assigned a short form UUID (16 bits rather than 128).

service set identifier (SSID): A sequence of characters that names a wireless local area network (WLAN).

Session Description Protocol (SDP): A protocol that is used for session announcement, session invitation, and other forms of multimedia session initiation. For more information see [MS-SDP] and [RFC3264].

tether: Enables a device to gain access to the Internet by establishing a connection with another device that is connected to the Internet. In the case of the Tethering Control Channel Protocol, a connection is established between a client and a server and the client subsequently shares the server's Internet connection.

type-length-value (TLV): A method of organizing data that involves a Type code (16-bit), a specified length of a Value field (16-bit), and the data in the Value field (variable).

Unicode: A character encoding standard developed by the Unicode Consortium that represents almost all of the written languages of the world. The **Unicode** standard [UNICODE5.0.0/2007] provides three forms (UTF-8, UTF-16, and UTF-32) and seven schemes (UTF-8, UTF-16, UTF-16 BE, UTF-16 LE, UTF-32, UTF-32 LE, and UTF-32 BE).

UTF-8: A byte-oriented standard for encoding Unicode characters, defined in the Unicode standard. Unless specified otherwise, this term refers to the UTF-8 encoding form specified in [UNICODE5.0.0/2007] section 3.9.

MAY, SHOULD, MUST, SHOULD NOT, MUST NOT: These terms (in all caps) are used as defined in [RFC2119]. All statements of optional behavior use either MAY, SHOULD, or SHOULD NOT.

1.2 References

Links to a document in the Microsoft Open Specifications library point to the correct section in the most recently published version of the referenced document. However, because individual documents in the library are not updated at the same time, the section numbers in the documents may not match. You can confirm the correct section numbering by checking the Errata.

1.2.1 Normative References

We conduct frequent surveys of the normative references to assure their continued availability. If you have any issue with finding a normative reference, please contact dochelp@microsoft.com. We will assist you in finding the relevant information.

[BT-RFCOMM] Bluetooth Special Interest Group, "Bluetooth Specification version 1.1, Part F:1, RFCOMM with TS 07.10, Serial Port Emulation", June 2003, http://www.bluetooth.org

Note There is a charge to download the specification.

[BT-SDP] Bluetooth Special Interest Group, "Bluetooth Specification Version 4.0, Volume 3 - Core System Package [Host Volume], Part B - Service Discovery Protocol (SDP) Specification", June 2010, http://www.bluetooth.org

Note There is a charge to download the specification.

[IEEE802.11-2012] IEEE, "Standard for Information Technology - Telecommunications and Information Exchange Between Systems - Local and Metropolitan Area Networks - Specific Requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications", ANSI/IEEE Std 802.11-2012, http://standards.ieee.org/getieee802/download/802.11-2012.pdf

Note There is a charge to download this document.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997, http://www.rfc-editor.org/rfc/rfc2119.txt

[WF-Security] Wi-Fi Alliance, "Security", http://www.wi-fi.org/discover-wi-fi/security

1.2.2 Informative References

None.

1.3 Overview

The Tethering Control Channel Protocol facilitates the sharing of a server's Internet connection with one or more clients that are using Wi-Fi. To initiate the connection, a client sends a request to a server to indicate that it is seeking to share the server's Internet connection. When the connection is successful, the server responds to the client with the appropriate Wi-Fi information. In the event that connection sharing is unsuccessful, the server returns an error message.

To use this protocol, the client is required to establish a secure, authenticated connection with the server that is capable of **tethering**. The client sends a request to the server to initiate tethering and to obtain the tethering configuration settings. In response, the server enables tethering, if it is not already enabled, and replies to the client with the tethering configuration settings, including the Wi-Fi **service set identifier (SSID)**, **basic service set identifier (BSSID)**, passphrase, and Display Name. The client uses these settings to connect to the tethering network. In the event that the server is unable to successfully enable tethering, the server sends the appropriate error message to the client.

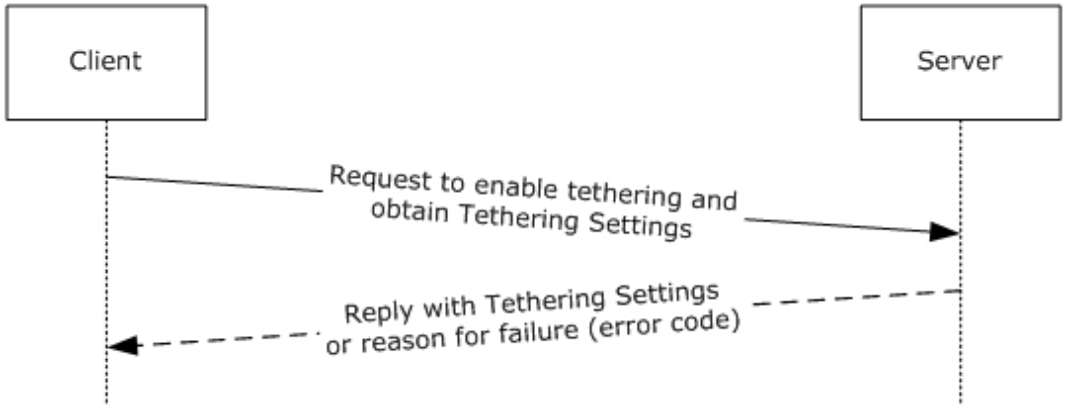


Figure 1: Tethering Control Channel Protocol request-reply sequence

1.4 Relationship to Other Protocols

None.

1.5 Prerequisites/Preconditions

The Tethering Control Channel Protocol depends on a secure and authenticated communication channel between the client and server.

1.6 Applicability Statement

This protocol is only applicable when the client initiates the tethering request. The client is required to support connecting to Wi-Fi networks and the server is required to support Internet connection sharing.

1.7 Versioning and Capability Negotiation

This document covers versioning issues in the following areas:

Protocol Versions: The Tethering Control Channel protocol supports future enhancements as defined in sections 3.1.5.4 and 3.2.5.3.

1.8 Vendor-Extensible Fields

None.

1.9 Standards Assignments

None.

2 Messages

2.1 Transport

To use the Tethering Control Channel Protocol, a byte stream connection MUST be established by using **radio frequency communications (RFCOMM)** [BT-RFCOMM] between the client and server. To identify a tethering-capable server using RFCOMM, the client MUST use the Bluetooth **Service Discovery Protocol (SDP)** [BT-SDP].

Tethering-capable servers MUST be identified through **SDP** by using the **globally unique identifier (GUID)** {232E51D8-91FF-4c24-AC0F-9EE055DA30A5}. To ensure that the RFCOMM communication is authenticated, the client MUST have a Bluetooth **pairing relationship** with the server.

2.2 Message Syntax

The protocol uses a common **type-length-value (TLV)** encoding schema for all messages. All strings are in **Unicode UTF-8** format unless otherwise specified.

2.2.1 Enumerations

2.2.1.1 MessageId Enumeration

The **MessageId** enumeration indicates the type of message being sent within the header of each message. For details about the message header, see section 2.2.2.2. The following values are supported.

Field/Value	Description
BringUpStartRequest 1	Indicates the **BringUpStartRequest** message (section 2.2.3.3).
BringUpSuccessResponse 2	Indicates the **BringUpSuccessResponse** message (section 2.2.3.4).
BringUpFailureResponse 3	Indicates the **BringUpFailureResponse** message (section 2.2.3.2).
ProtocolErrorResponse 4	Indicates the **ProtocolErrorResponse** message (section 2.2.3.4).

2.2.1.2 StatusCodeEnum Enumeration

The **StatusCodeEnum** enumeration specifies possible outcomes for the attempt to start tethering on the server. The following values are supported.

Field/Value	Description
Success 0	The operation succeeded; tethering is enabled on the server.
UnspecifiedError 1	The operation failed and the server is unable to provide a specific status code.

Field/Value	Description
OperationCancel 2	The operation failed because it was canceled by the user.
EntitlementCheckFail 3	The operation failed because the mobile operator has not authorized the subscriber to use tethering on the server.
NoCellularSignal 4	The operation failed because there is no cellular signal on the server.
CellularDataTurnedOff 5	The operation failed because cellular data is turned off on the server.
CannotConnectToCellularNetwork 6	The operation failed because the server is unable to connect to the cellular network.
ConnectToCellularNetworkTimedOut 7	The operation failed because the connection attempt to the cellular network timed out.
RoamingNotAllowed 8	The operation failed because the server is not allowed to connect to the cellular network when the latter is in the roaming state.

2.2.1.3 TypeId Enumeration

The **TypeId** enumeration identifies the type of structure contained within the message payload. The structure is encoded by using the CommonHeader (section 2.2.2.2). The following values are supported.

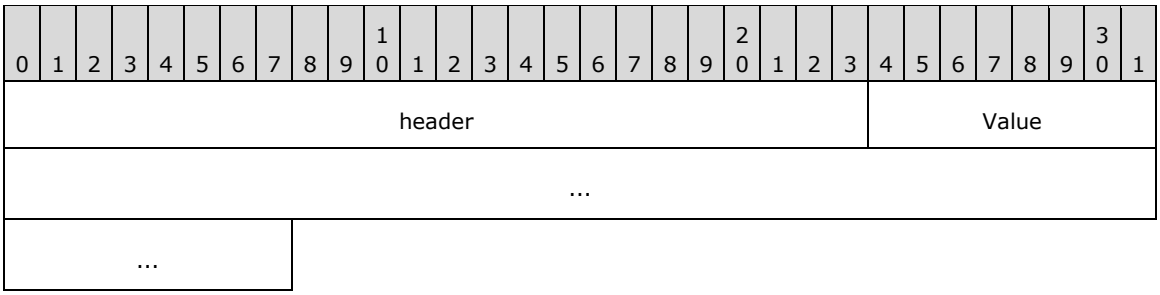
Field/Value	Description
StatusCode 1	The structure contains a StatusCode (section 2.2.2.8).
Ssid 2	The structure contains an SSID (section 2.2.2.7).
Bssid 3	The structure contains a BSSID (section 2.2.2.1).
Passphrase 4	The structure contains a passphrase (section 2.2.2.6).
DisplayName 5	The structure contains a DisplayName (section 2.2.2.3).
ErrorString 6	The structure contains an ErrorString (section 2.2.2.4).
MessageType 7	The structure contains a MessageType (section 2.2.2.5).

2.2.2 Structures

The following sections define the structures that are used to encode the message payload. Each structure is formatted with a TLV encoding schema by using a common header.

2.2.2.1 Bssid Structure

The **Bssid** structure specifies the BSSID of the Wi-Fi network used by the server in the Internet connection, as specified in [IEEE802.11-2012].

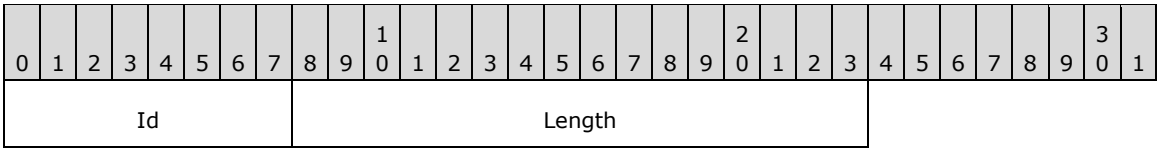


header (3 bytes): The **header** field contains the **CommonHeader** structure (section 2.2.2.2), where the value of the **Id** field is set to 3 (Bssid), as specified in TypeId (section 2.2.1.3), and the value of the **Length** field is set to 6 bytes.

Value (6 bytes): The **Value** field contains the value of the BSSID.

2.2.2.2 CommonHeader Structure

The **CommonHeader** structure is used by all structures to identify the type and length of the structure encoded in the message payload.

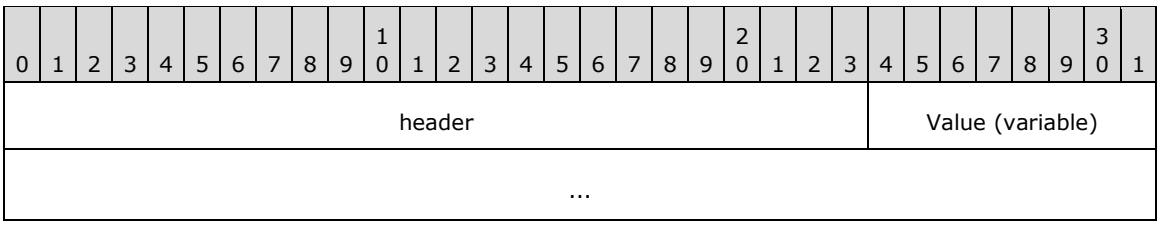


Id (1 byte): The **Id** field specifies the type of the structure encoded in the message, as defined in section 2.2.1.3.

Length (2 bytes): The **Length** field specifies the number of bytes that follow the **CommonHeader** which correspond to the length of the encoded structure. Note that the structure is encoded in network byte order.

2.2.2.3 DisplayName Structure

The **DisplayName** structure specifies the display name for the server.



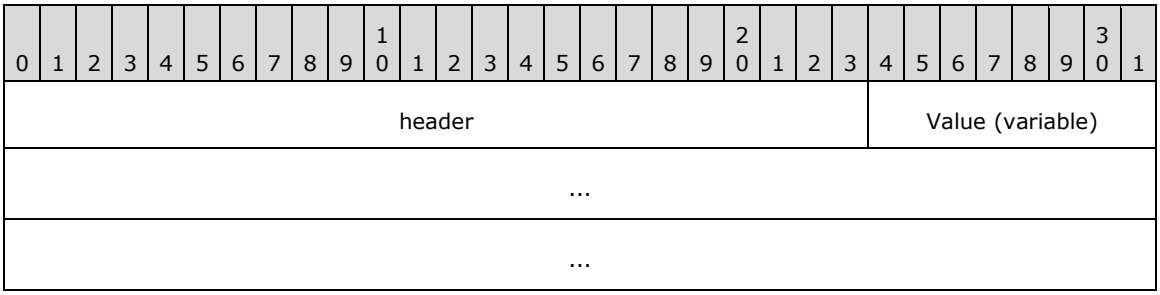
...

header (3 bytes): The **header** field contains the **CommonHeader** structure (section 2.2.2.2), where the value of the **Id** field is set to 5 (DisplayName), as specified in **TypeId** (section 2.2.1.3), and the value of the **Length** field is variable.

Value (variable): The **Value** field contains the Display Name string. Because the length of the **Length** field within the **CommonHeader** structure is 2 bytes, the length of the display name string is limited to a maximum of 65,535 bytes.

2.2.2.4 ErrorString Structure

The **ErrorString** structure specifies the error message corresponding to the result of the tethering attempt to the server.

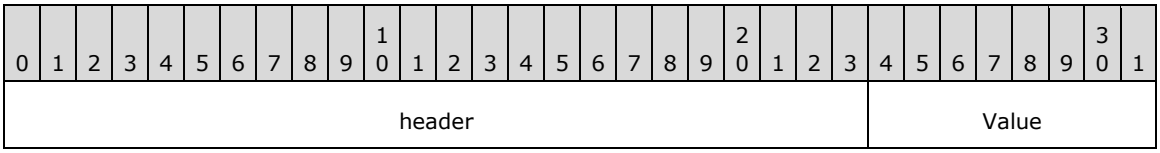


header (3 bytes): The **header** field contains the **CommonHeader** structure (section 2.2.2.2), where the value of the **Id** field is set to 6 (ErrorString), as specified in **TypeId** (section 2.2.1.3), and the value of the **Length** field is variable.

Value (variable): The **Value** field contains the error message string. Because the length of the **Length** field within the **CommonHeader** structure is 2 bytes, the length of the error message string is limited to a maximum of 65,535 bytes.

2.2.2.5 MessageType Structure

The **MessageType** structure identifies the type of structure contained within the message payload.

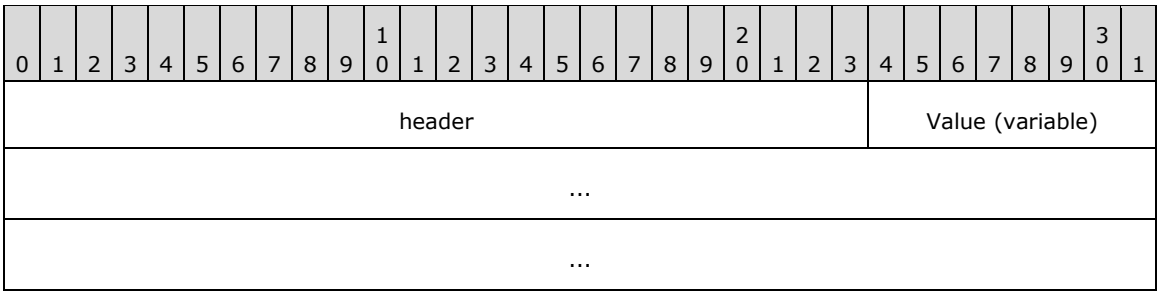


header (3 bytes): The **header** field contains the **CommonHeader** structure (section 2.2.2.2), where the value of the **Id** field is set to 7 (MessageType), as specified in **TypeId** (section 2.2.1.3), and the value of the **Length** field is set to 1.

Value (1 byte): The **Value** field identifies the type of structure contained within the message payload, as defined in section 2.2.1.1.

2.2.2.6 Passphrase Structure

The **Passphrase** structure specifies the Wi-Fi Protected Access 2 (WPA2) **passphrase**, as defined in [WF-Security], that is used in the tethering connection. The passphrase contains 8 to 64 characters.

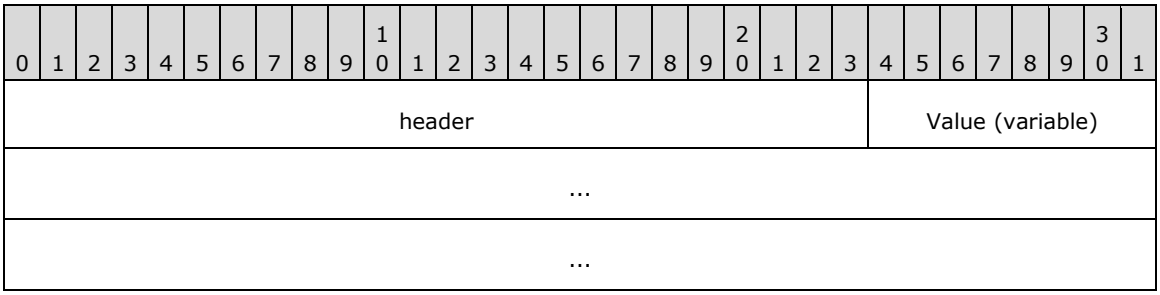


header (3 bytes): The **header** field contains the **CommonHeader** structure (section 2.2.2.2), where the value of the **Id** field is set to 4 (Passphrase), as specified in **TypeId** (section 2.2.1.3), and the value of the **Length** field MUST be 8 to 64 characters.

Value (variable): The **Value** field specifies the WPA2 passphrase encoded as **ASCII**. If the length of the passphrase is 64 characters, all of the characters MUST be hexadecimal characters. If the length is 8 to 63 characters, all of the characters MUST be ASCII characters in the range of 32 to 126.

2.2.2.7 Ssid Structure

The **Ssid** structure specifies the Wi-Fi SSID for the tethering connection. The Wi-Fi SSID is a byte BLOB of 0 to 32 bytes as defined in [IEEE802.11-2012].

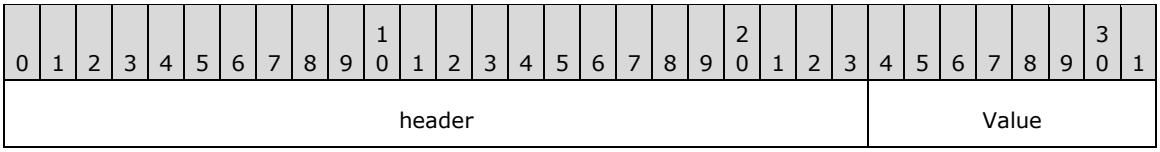


header (3 bytes): The **header** field contains the **CommonHeader** structure (section 2.2.2.2), where the value of the **Id** field is set to 2 (Ssid), as specified in **TypeId** (section 2.2.1.3), and the value of the **Length** field is in the range of 0 to 32 bytes.

Value (variable): The **Value** field specifies the SSID.

2.2.2.8 StatusCode Structure

The **StatusCode** structure specifies the status code representing the outcome of the attempt by the client to enable tethering on the server.



header (3 bytes): The **header** field contains the **CommonHeader** structure (section 2.2.2.2), where the value of the **Id** field is set to 1 (StatusCode), as specified in **TypeId** (section 2.2.1.3), and the value of the **Length** field is set to 1.

Value (1 byte): The **Value** field specifies the status code representing the outcome of the attempt to start tethering on the server, as defined in section 2.2.1.2.

2.2.3 Messages

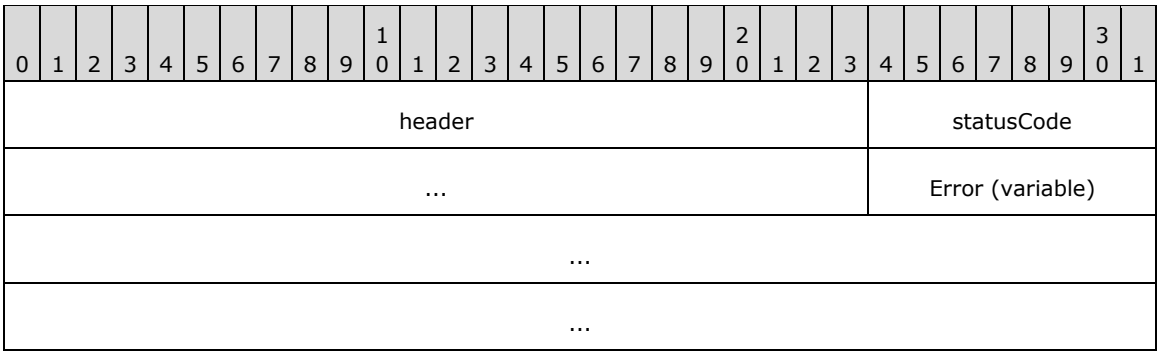
The messages described in the following sections each contain the **CommonHeader** structure (section 2.2.2.2) that specifies the message type and length, and zero or more other structures as defined in section 2.2.2. In an implementation, the messages MUST contain all defined fields unless a field is marked as "(optional)". Optional fields MAY be included as required by the implementation.

When a message contains more than one structure, the location of the structures within the message MUST be in increasing numeric order as indicated by the value of the **TypeId** enumeration (section 2.2.1.3). Unless otherwise specified, a message MUST NOT contain multiple structures with the same **TypeId** (section 2.2.1.3) value.

Messages MAY contain structures that are not defined in this protocol. However, implementations of this protocol MUST ignore all message structures that are not specified in this specification [MS-TCC] to enable compatibility with future protocol versions.

2.2.3.1 BringUpFailureResponse Message

The **BringUpFailureResponse** message is sent by the server to the client in response to a **BringUpStartRequest** message (section 2.2.3.2) when the request for tethering fails. If the reason for the failure cannot be adequately described through the use of the **statusCode** field, the message SHOULD only contain an error string.



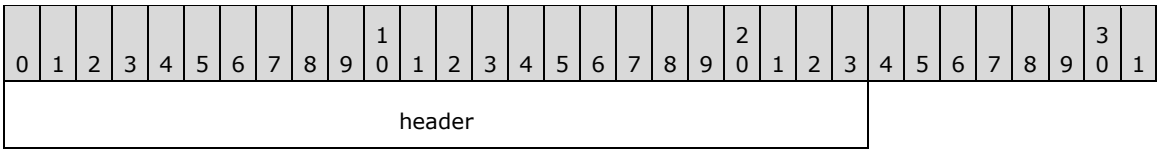
header (3 bytes): The **header** field contains the **CommonHeader** structure (section 2.2.2.2), where the value of the **Id** field is set to 3 (BringUpFailureResponse), as specified in **MessageId** (section section 2.2.1.1), the length of the **Length** field is variable, and the value MUST be set to the combined size of all structures contained within the message.

statusCode (4 bytes): The **statusCode** field specifies the status code, as defined in the **StatusCodeEnum** enumeration (section 2.2.2.8), that represents the outcome of the attempt to start tethering on the server. For the **BringUpFailureResponse** message, the value of the **statusCode** field MUST NOT be set to 0 (Success).

Error (variable): (optional) When the **Error** field is present in the message, it contains an error message, as defined in the **ErrorString** structure (section 2.2.2.4), that corresponds to the result of the tethering attempt to the server. If the error message is an empty string, this field SHOULD NOT be included in the message.

2.2.3.2 BringUpStartRequest Message

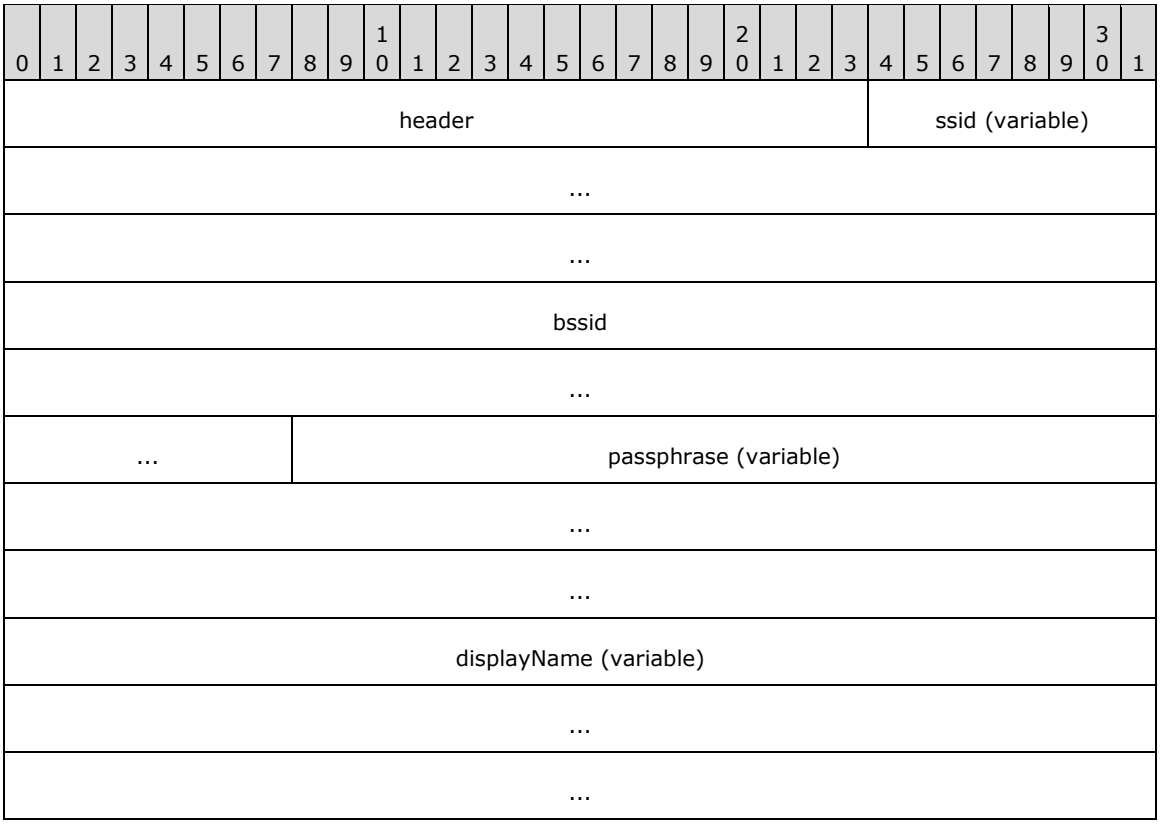
The **BringUpStartRequest** message is sent from the client to the server to request to share the server's Internet connection. In this protocol version, the message has no additional payload.



header (3 bytes): The **header** field contains the **CommonHeader** structure (section 2.2.2.2), where the value of the **Id** field is set to 1 (BringUpStartRequest), as specified in **MessageId** (section section 2.2.1.1), the length of the **Length** field is variable, and the value MUST be set to the combined size of all structures contained within the message. In this protocol version, the message does not contain any additional structures.

2.2.3.3 BringUpSuccessResponse Message

The **BringUpSuccessResponse** message is sent by the server to the client in response to a **BringUpStartRequest** message (section 2.2.3.2) when the request for tethering is successful.



header (3 bytes): The **header** field contains the **CommonHeader** structure (section 2.2.2.2), where the value of the **Id** field is set to 2 (BringUpSuccessResponse), as specified in **MessageId** (section section 2.2.1.1), the length of the **Length** field is variable, and the value MUST be set to the combined size of all structures contained within the message.

ssid (variable): The **ssid** field has a variable length and specifies the Wi-Fi SSID for the tethering connection, as defined in the **Ssid** structure (section 2.2.2.7). The Wi-Fi SSID is a byte BLOB of 0 to 32 bytes as defined in [IEEE802.11-2012].

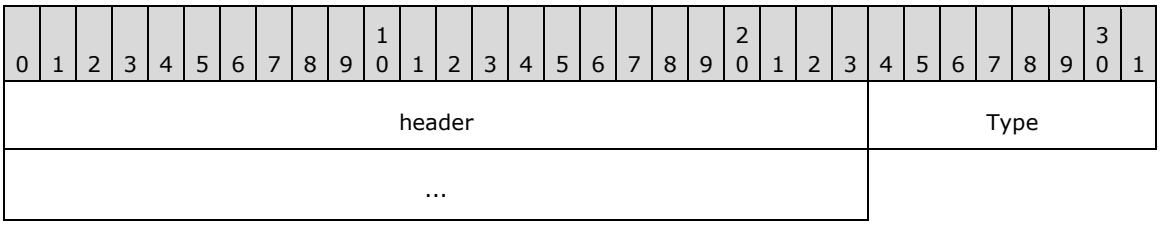
bssid (9 bytes): (optional) The **bssid** field contains the **Bssid** structure (section 2.2.2.1), which consists of the header (3 bytes) and the BSSID value (6 bytes) of the Wi-Fi network used by the server in the Internet connection, as specified in [IEEE802.11-2012].

passphrase (variable): The **passphrase** field has a variable length and specifies the Wi-Fi WPA2 passphrase, as defined in the **Passphrase** structure (section 2.2.2.6), used in the tethering connection, as defined by the Wi-Fi Alliance Security Certification [WF-Security]. The passphrase contains 8 to 64 characters. If the length of the passphrase is 64 characters, all of the characters MUST be hexadecimal characters. If the length is 8 to 63 characters, all of the characters MUST be ASCII characters in the range of 32 to 126.

displayName (variable): The **displayName** field has a variable length and specifies the display name for the server, as defined in the **DisplayName** structure (section 2.2.2.3).

2.2.3.4 ProtocolErrorResponse Message

The **ProtocolErrorResponse** message is sent in response to the receipt of a message from the client that is not specified as expected according to the structures defined in this specification [MS-TCC]. The **ProtocolErrorResponse** message enables compatibility with future protocol versions that MAY contain messages, structures, or values not defined in this protocol version. In this protocol version, an implementation sends this message in response to the receipt of a message where the **MessageId** value is not within the expected range as defined in section 2.2.1.1.



header (3 bytes): The **header** field contains the **CommonHeader** structure (section 2.2.2.2), where the value of the **Id** field is set to 4 (ProtocolErrorResponse), as specified in **MessageId** (section section 2.2.1.1), the length of the **Length** field is variable, and the value MUST be set to the combined size of all structures contained within the message.

Type (4 bytes): The **Type** field specifies the type of the received message, as defined in the **MessageType** structure (section 2.2.2.5). The value of the field is set to the **MessageId** value of the received message that is not within the expected range as defined in section 2.2.1.1.

3 Protocol Details

3.1 Client Details

In the Tethering Control Channel Protocol, the client role performs one primary operation and that is to send a **BringUpStartRequest** message (section 2.2.3.2) to the server and to wait for the server's response. When the client receives a **BringUpSuccessResponse** message (section 2.2.3.3) or **BringUpFailureResponse** message (section 2.2.3.1) from the server, the client disconnects the transport, notifies the higher layer that it has completed the operation, and terminates the connection.

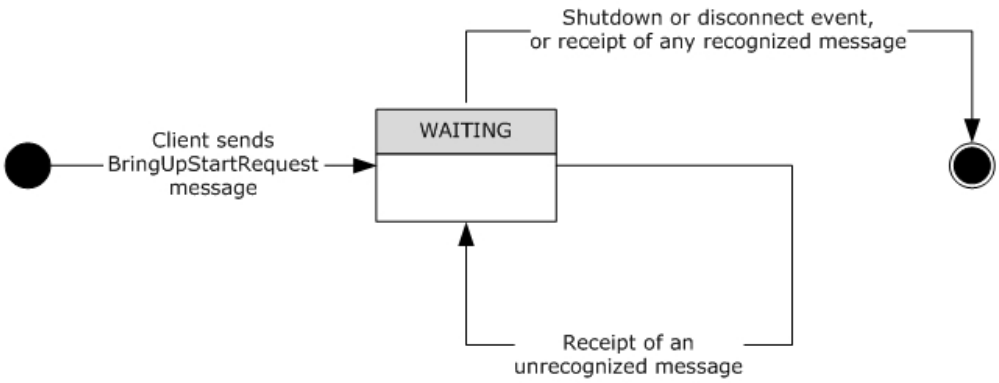


Figure 2: Client message processing

3.1.1 Abstract Data Model

None.

3.1.2 Timers

MessageTimer: Specifies the time-out interval for the request operation. The interval is set to one minute.

3.1.3 Initialization

The Tethering Control Channel Protocol is initialized after the transport protocol has created a communication channel with the server. After the communication channel is initialized, the client sends a **BringUpStartRequest** message (section 2.2.3.3) to the server.

3.1.4 Higher-Layer Triggered Events

3.1.4.1 Cancellation

The higher layer can terminate the client's connection while the client is waiting for a response from the server.

3.1.5 Message Processing Events and Sequencing Rules

Messages in the Tethering Control Channel Protocol are identified by the values specified in the **CommonHeader** structure (section 2.2.2.2) contained in each message. A message is only processed

after the entire contents of the message have been received as indicated by the value of the **Length** field specified within the **CommonHeader**.

When the client receives a message, the **MessageTimer** (see section 3.1.2) is started or restarted.

3.1.5.1 BringUpSuccessResponse

When the client receives a **BringUpSuccessResponse** message (section 2.2.3.3), the client sends the contents of the message to the higher layer and terminates the connection.

3.1.5.2 BringUpFailureResponse

When the client receives a **BringUpFailureResponse** message (section 2.2.3.1), the client sends the contents of the message to the higher layer and terminates the connection.

3.1.5.3 Failure Messages

When the client receives a **BringUpStartRequest** message (section 2.2.3.2) or **ProtocolErrorResponse** message (section 2.2.3.4), the client indicates to the higher layer that a protocol failure event has occurred and terminates the connection.

3.1.5.4 Other Messages

When the client receives a message with an unrecognized message type, that is, a **MessageId** value that is not within the expected range defined in section 2.2.1.1, the client sends a **ProtocolErrorResponse** message (section 2.2.3.4) with the **Value** field of the **MessageType** structure (section 2.2.2.5) set to the unrecognized value.

When the client receives a message that cannot be parsed according to the message syntax specified in section 2.2, the client indicates to the higher layer that a protocol error has occurred and terminates the connection.

3.1.6 Timer Events

After the **MessageTimer** (see section 3.1.2) expires, the client sends the time-out to the higher layer then disconnects and terminates the connection.

3.1.7 Other Local Events

3.1.7.1 Disconnect Event of Transport Channel

If the transport channel becomes disconnected, the client indicates to the higher layer that a transport failure has occurred and terminates the connection.

3.2 Server Details

In the Tethering Control Channel Protocol, initialization occurs when a client connects to the server and the server creates an instance of the server role for each connection with a client. The server MAY handle multiple clients simultaneously by having an instance of the server role for each client.

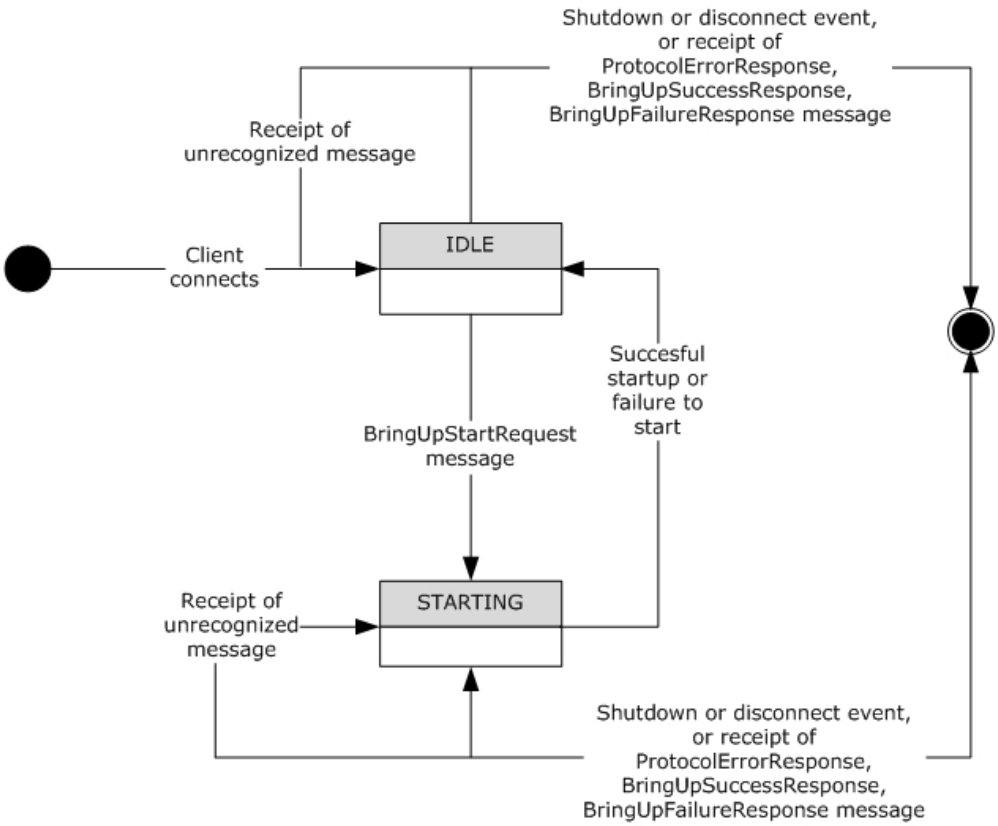


Figure 3: Server message processing

Note When the server is in the STARTING state, the server MUST NOT process messages.

3.2.1 Abstract Data Model

This section describes a conceptual model of possible data organization that an implementation maintains to participate in this protocol. The described organization is provided to facilitate the explanation of how the protocol behaves. This document does not mandate that implementations adhere to this model as long as their external behavior is consistent with that described in this document.

The following elements are specific to this protocol:

State: Indicates the server state, either IDLE or STARTING.

Tethering Settings: A data set that contains the following items:

- **SSID:** A Wi-Fi SSID which is a byte blob of 0 to 32 bytes as defined in [IEEE802.11-2012].
- **BSSID:** A Wi-Fi **BSSID** as specified in [IEEE802.11-2012].
- **Passphrase:** A Wi-Fi WPA2 passphrase. The passphrase contains 8 to 64 characters. If the length of the passphrase is 64 characters, all of the characters MUST be hexadecimal characters. If the length is 8 to 63 characters, all of the characters MUST be ASCII characters in the range of 32 to 126.

- **DisplayName:** A Unicode string.

Failure Description: A data set that contains the following items.

- **Status Code:** A status code as defined in the **StatusCodeEnum** structure (section 2.2.1.2).
- **ErrorString:** A Unicode string. Note that the string can be empty.

3.2.2 Timers

ServerTimer: Specifies the time-out interval for the response to the client. The interval is set to one minute.

3.2.3 Initialization

The server is initialized when the transport protocol indicates that a client has connected. The initial state for the server is set to IDLE.

3.2.4 Higher-Layer Triggered Events

3.2.4.1 Shutdown

The higher-layer can shut down the server at any time. The server disconnects the transport protocol at shutdown.

3.2.4.2 Tethering Started or Failed to Start

When the server indicates to the higher layer that tethering has to be started, the higher layer responds either by confirming that tethering has been started, or that tethering failed to start.

When the higher layer confirms that tethering has been started, the higher layer provides the current **Tethering Settings**. The server sends a **BringUpSuccessResponse** message (section 2.2.3.3) that contains the **Tethering Settings** and returns to the IDLE state.

If the higher layer failed to start tethering, the higher layer provides a reason for the failure. The server sends a **BringUpFailureResponse** message (section 2.2.3.1) that contains the reason for the failure and returns to the IDLE state.

3.2.5 Message Processing Events and Sequencing Rules

Messages in the Tethering Control Channel Protocol are identified by the values specified in the **CommonHeader** structure (section 2.2.2.2) contained in each message. A message is only processed after the entire contents of the message have been received as indicated by the value of the **Length** field specified within the **CommonHeader**.

When the server receives a message, the **ServerTimer** (see section 3.2.2) is started or restarted.

3.2.5.1 BringUpStartRequest

When the server is in the IDLE state and receives a **BringUpStartRequest** message (section 2.2.3.2), the server enters the STARTING state and indicates to the higher layer that tethering has to be started.

When the server is in the STARTING state, it MUST NOT process messages.

3.2.5.2 Failure Messages

When the server receives a **BringupFailureResponse**, **BringUpSuccessResponse** or **ProtocolErrorResponse** message (see section 2.2.3), the server indicates to the higher layer that a protocol failure event has occurred and terminates the connection.

3.2.5.3 Other Messages

When the server receives a message with an unrecognized message type, that is, a **MessageId** value that is not within the expected range defined in section 2.2.1.1, the server sends a **ProtocolErrorResponse** message (section 2.2.3.4) with the **Value** field of the **MessageType** structure (section 2.2.2.5) set to the unrecognized value.

When the server receives a message that cannot be parsed according to the message syntax specified in section 2.2, the server indicates to the higher layer that a protocol error has occurred and terminates the connection.

3.2.6 Timer Events

After the **ServerTimer** (see section 3.2.2) expires, the server disconnects from the client and terminates the connection.

3.2.7 Other Local Events

3.2.7.1 Disconnect Event of Transport Channel

If the transport channel becomes disconnected, the server indicates to the higher layer that a transport failure has occurred and terminates the connection.

4 Protocol Examples

4.1 Successful Startup

In the following example, the server successfully completes the client request and sends a BringUpSuccessResponse message (section 2.2.3.3). For more information, see the figure in section 1.3.

4.1.1 BringUpStartRequest Example (Successful)

```
Message Header: 0x01 0x00 0x00 (Type == BringUpStartRequest, Length == 0)
```

4.1.2 BringUpSuccessResponse Example (Successful)

```
Message Header: 0x02 00 31 (Type == BringUpSuccessResponse, Length == 49)
Common Header: 0x02 00 0B (Type == SSID, Length == 11)
SSID payload: 0x53 0x61 0x6D 0x70 0x6C 0x65 0x20 0x53 0x53 0x49 0x44 ("Sample SSID")
Common Header: 0x03 00 06 (Type == BSSID, Length == 6)
BSSID payload: 0x01 0x02 0x03 0x04 0x05 0x06 (BSSID: 01:02:03:04:05:06)
Common Header: 0x04 00 09 (Type == Passphrase, Length == 9)
Passphrase payload: 0x73 0x65 0x63 0x72 0x65 0x74 ("secret123")
Common Header: 0x05 00 0B (Type == DisplayName, Length == 11)
DisplayName payload: 0x42 0x6F 0x62 0x27 0x73 0x20 0x70 0x68 0x6F 0x6E 0x65 ("Bob's phone")
```

4.2 Unsuccessful Startup

In the following example, the server does not successfully complete the client request and sends a BringUpFailureResponse message (section 2.2.3.1). For more information, see the figure in section 1.3.

4.2.1 BringUpStartRequest Example (Unsuccessful)

```
Message Header: 0x01 0x00 0x00 (Type == BringUpStartRequest, Length == 0)
```

4.2.2 BringUpFailureResponse Example (Unsuccessful)

```
Message Header: 0x03 00 (Type == BringUpFailureResponse, Length == 4)
Common Header: 0x01 00 01 (Type == StatusCode, Length == 1)
Status code payload: 0x04 (StatusCode.NoCellularSignal)
```

5 Security

5.1 Security Considerations for Implementers

The Tethering Control Channel Protocol requires an authenticated and encrypted communication channel.

5.2 Index of Security Parameters

None.

6 Appendix A: Product Behavior

The information in this specification is applicable to the following Microsoft products or supplemental software. References to product versions include released service packs.

- Windows 8.1 operating system
- Windows Server 2012 R2 operating system
- Windows 10 operating system

Exceptions, if any, are noted below. If a service pack or Quick Fix Engineering (QFE) number appears with the product version, behavior changed in that service pack or QFE. The new behavior also applies to subsequent service packs of the product unless otherwise specified. If a product edition appears with the product version, behavior is different in that product edition.

Unless otherwise specified, any statement of optional behavior in this specification that is prescribed using the terms SHOULD or SHOULD NOT implies product behavior in accordance with the SHOULD or SHOULD NOT prescription. Unless otherwise specified, the term MAY implies that the product does not follow the prescription.

7 Change Tracking

This section identifies changes that were made to this document since the last release. Changes are classified as New, Major, Minor, Editorial, or No change.

The revision class **New** means that a new document is being released.

The revision class **Major** means that the technical content in the document was significantly revised. Major changes affect protocol interoperability or implementation. Examples of major changes are:

- A document revision that incorporates changes to interoperability requirements or functionality.
- The removal of a document from the documentation set.

The revision class **Minor** means that the meaning of the technical content was clarified. Minor changes do not affect protocol interoperability or implementation. Examples of minor changes are updates to clarify ambiguity at the sentence, paragraph, or table level.

The revision class **Editorial** means that the formatting in the technical content was changed. Editorial changes apply to grammatical, formatting, and style issues.

The revision class **No change** means that no new technical changes were introduced. Minor editorial and formatting changes may have been made, but the technical content of the document is identical to the last released version.

Major and minor changes can be described further using the following change types:

- New content added.
- Content updated.
- Content removed.
- New product behavior note added.
- Product behavior note updated.
- Product behavior note removed.
- New protocol syntax added.
- Protocol syntax updated.
- Protocol syntax removed.
- New content added due to protocol revision.
- Content updated due to protocol revision.
- Content removed due to protocol revision.
- New protocol syntax added due to protocol revision.
- Protocol syntax updated due to protocol revision.
- Protocol syntax removed due to protocol revision.
- Obsolete document removed.

Editorial changes are always classified with the change type **Editorially updated**.

Some important terms used in the change type descriptions are defined as follows:

- **Protocol syntax** refers to data elements (such as packets, structures, enumerations, and methods) as well as interfaces.
- **Protocol revision** refers to changes made to a protocol that affect the bits that are sent over the wire.

The changes made to this document are listed in the following table. For more information, please contact dochelp@microsoft.com.

Section	Tracking number (if applicable) and description	Major change (Y or N)	Change type
6 Appendix A: Product Behavior	Added Windows Server 2012 R2 to the applicability list.	Y	Content update.

8 Index