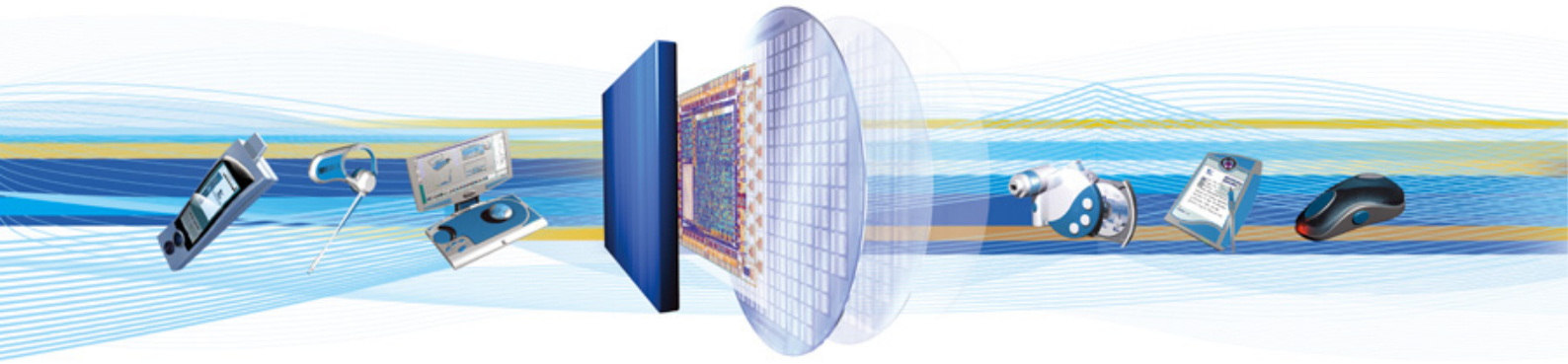
CSR Synergy Bluetooth 18.2.0

HFG – Hands-Free Gateway

API Description

November 2011



Cambridge Silicon Radio Limited

Churchill House
Cambridge Business Park
Cowley Road
Cambridge CB4 0WZ
United Kingdom

Registered in England and Wales 3665875

Tel: +44 (0)1223 692000
Fax: +44 (0)1223 692001
www.csr.com

Contents

List of Figures

List of Tables

1 Introduction

1.1 Introduction and Scope

This document describes the message interface provided by the Hands-Free Gateway profile manager that is implementing the gateway part of the Hands-Free profile and the gateway part of the Headset profile, henceforward referred to as the HFG. The HFG profile manager requirements and functionality are specified in [HF] and [HS].

1.2 Assumptions

The following assumptions and preconditions are made in the following:

- There is a secure and reliable transport between the profile part, i.e. HFG, and the application layer

Knowledge of the HF profile [HF], the HS profile [HS] and the Hands-Free Profile Application Guideline [CCAP] is assumed since part of the functionality is application layer specific and must be implemented in the application layer.

2 Description

2.1 Introduction

The HFG profile manager supplies the interface for applications that should provide the functionality and conform to the gateway side of the hands-free profile and the gateway side of the headset profile. The HFG is implemented as specified in the hands-free profile [HF] with the addition of the requirements as specified in the Bluetooth® Hands-Free profile Application Guidelines [CCAP].

The HFG profile layer provides functionality for:

- Establishing and maintaining a service level connection between the HFG profile layer and one or more peer devices. The peer devices must conform to either the HF side as defined in [HF] or the HS side as defined in [HS]
- Sending and receiving AT-commands and result codes between a HFG profile manager and a HF or HS device
- For the application layer automatic and transparent handling of low power modes
- Establishing eSCO and SCO audio connections with the peer device(s)
- Enabling and disabling special features through CSR proprietary commands. See chapter 3.7.1 for details.

2.2 Reference Model

The application must interface to the HFG, the Security Controller (SC), and the Service Discovery (SD) module. The Service Discovery module can be used for searching for devices and gathering different kinds of information about the device before establishing any trust relations or connections. The Security Controller has an interface that is used for bonding and pairing. The Connection Manager is used by both the SC, SD and HFG, but applications should avoid the use the CM directly.

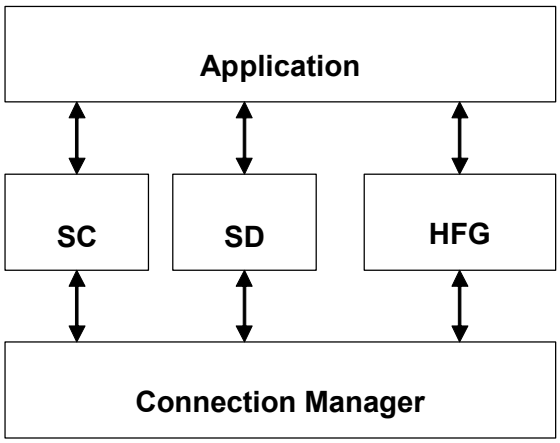


Figure 1: Reference model

Thus e.g. bonding, which is normally carried out before any other transaction, must be accomplished before setting up any transactions using the HFG interface as described below.

2.3 Sequence Overview

Figure 2 outlines the basic functionality of the HFG profile. The profile as such have two major states; idle and activated. When the profile is activated it is possible to connect to and from headsets to the HFG profile.

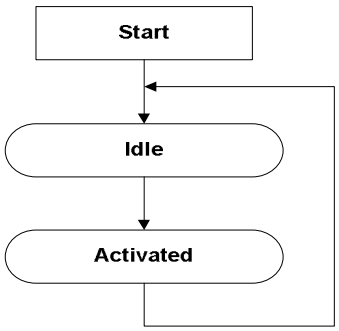


Figure 2: Profile sequence overview

When the CSR Synergy Bluetooth scheduler starts, the profile will automatically perform the most basic initialisation in order to enter the idle state. After that, the application can activate the profile by means of the CSR_BT_HFG_ACTIVATE_REQ signal, after which the profile enters the activated state. Should the application want to disable the HFG profile, it sends the CSR_BT_HFG_DEACTIVATE_REQ, which triggers a transition back to the idle state.

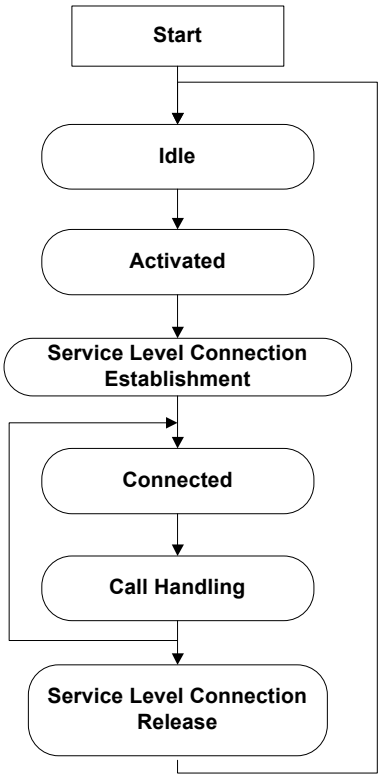


Figure 3: Connection sequence overview

A normal scenario is that a service level connection between the HFG side and the HF side is established after pairing of the devices. The service level connection can be established by both sides. Pairing is described in [SC] and is outside the scope of this document.

Upon completion of the service level connection, call(s) can be initiated. Call handling includes AT-command exchange and setting up a full duplex audio channel between the two devices. Once a call is terminated the service level connection may be maintained for another call setup or may be released.

3 Interface Description

3.1 Activation

Before the profile can be used the application needs to initialize the HFG manager by sending the CSR_BT_HFG_ACTIVATE_REQ message. The purpose of issuing this request is two-fold:

a) To register which features the profile supports. In the csr_bt_hfg_prim.h file there are a number of defines for the different features the hands-free gateway profile supports.

b) To enter a mode where incoming connections are accepted. Please note that whether or not the Bluetooth® device will be discoverable, i.e. can be found by other Bluetooth® devices, must be controlled by the application. For more information, please refer to [CM]. After initialisation of CSR Synergy Bluetooth, the Bluetooth® device is set up to be discoverable.

Note: Upon activation and until a connection has been established by the remote device, the application is not allowed to send any signals except for the purpose of leaving the connectable mode (see 3.2), for initiating a connection (see 3.3) or for sending status indicator updates (see 3.4.8).

3.2 Deactivation

The Bluetooth® device stays connectable until otherwise instructed by the application. By issuing the CSR_BT_HFG_DEACTIVATE_REQ, the HFG manager is so instructed. In addition the service records are removed and any established connections are disconnected.

If the application decides that the HFG service should no longer be connectable, the application may send a CSR_BT_HFG_DEACTIVATE_REQ and get a CSR_BT_HFG_DEACTIVATE_CFM back when deactivated. Please note that this is a local procedure that does not invoke any remote side functions, unless a connection has already been established.

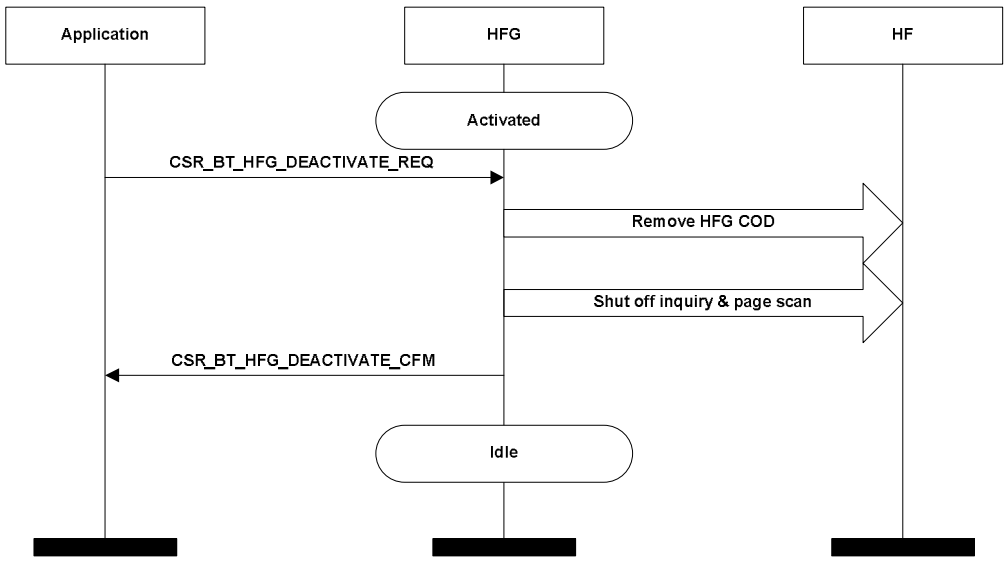


Figure 4: Deactivate accept connection establishment sequence

If the service has deactivated, the application must send a CSR_BT_HFG_ACTIVATE_REQ in order to once again accept incoming connections.

3.3 Service Level Connection Establishment

Before any call handling can take place, a service level connection must be established. Establishing a service level connection, which is basically a communication path for AT-command exchange and audio channel establishment, consists of the following:

- Serial channel connection establishment (including service discovery of remote side features)
- Information exchange on application level (AT-command exchange)

How to establish the service level connection via the HFG interface is described in section 3.3.1 and 0.

If the HFG profile manager is connecting or accepting a connection to/from a hands-free device, there is a sequence of AT commands that must be exchanged before a service level connection (SLC) is fully established. The profile will handle the exchange of the AT commands. If the connection device is a HS device no AT sequence is needed to establish a connection.

3.3.1 Accept Connection Establishment

In order to accept a connection initiated from a remote device, the HFG profile manager and the local device must be set in a mode where incoming connections are accepted. Accepting a remotely initiated connection is transparent to the application layer, which is only notified after the connection is fully established. The indication that a connection has been established is delivered to the application layer in the form of the CSR_BT_HFG_SERVICE_CONNECT_IND message, as shown in Figure 5.

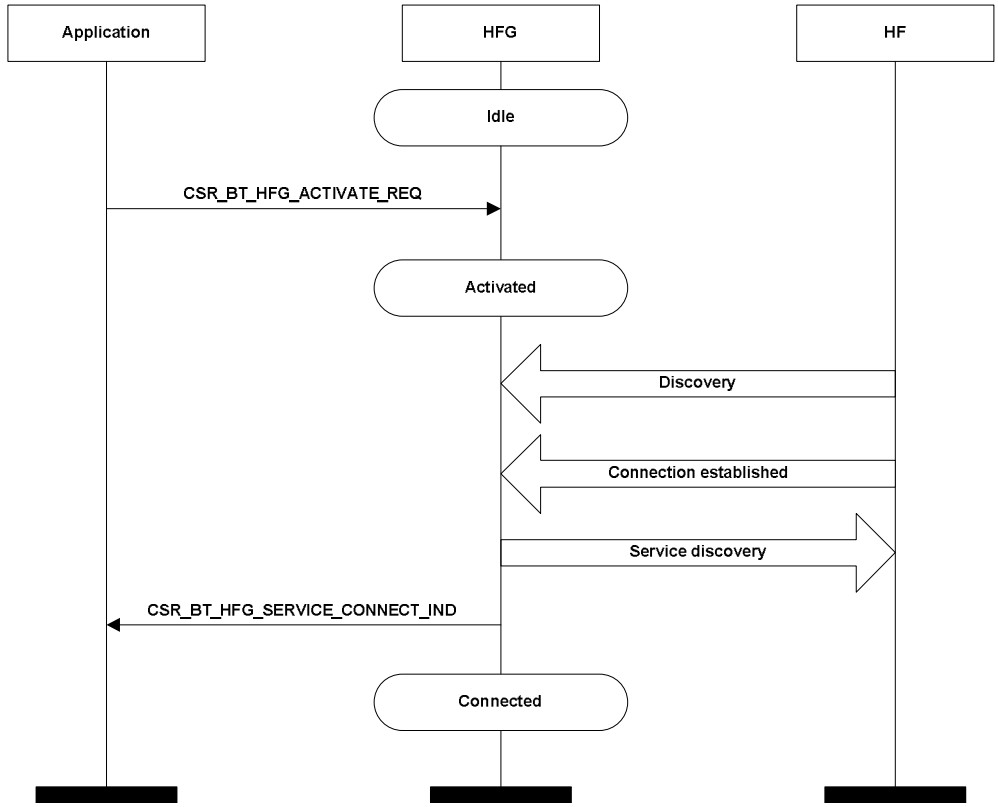


Figure 5: Accept connection establishment sequence

The information stored in the service discovery record is primarily constant compile-time defined data, but it is possible to define the name of the HFG service in the CSR_BT_HFG_ACTIVATE_REQ signal.

The application layer must be prepared to receive a CSR_BT_HFG_SERVICE_CONNECT_IND even when a CSR_BT_HFG_DEACTIVATE_REQ has been sent due to race conditions. In this case, the HFG will disconnect the connection and the application layer will receive a CSR_BT_HFG_DEACTIVATE_CFM when the connection is closed.

3.3.2 Initiate Connection Establishment

In addition to accepting a connection as presented above, it is also possible to initiate a connection to a remote hands-free device. Issuing a CSR_BT_HFG_SERVICE_CONNECT_REQ from the application will accomplish this. Completion of the connection is indicated with a CSR_BT_HFG_SERVICE_CONNECT_IND.

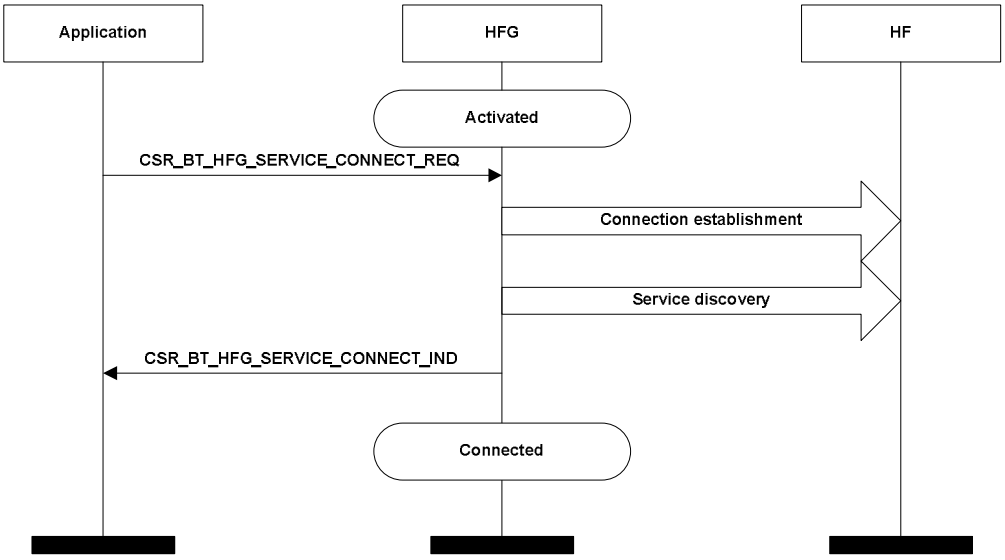


Figure 6: Locally initiated connection establishment sequence

Before sending a CSR_BT_HFG_SERVICE_CONNECT_REQ the Bluetooth® device address of the remote device must be known. If the address is unknown, it can be found by using the inquiry functionality of the Service Discovery Module, see [SD].

When sending the CSR_BT_HFG_SERVICE_CONNECT_REQ signal the profile will search the services of the device with the specified Bluetooth® device. If the remote device supports the HF profile the HFG profile manager will connect to this service, otherwise it will try to connect to the Headset profile if this is supported. It is also possible to instruct the HFG to connect directly to a specific service without taking the service search into account.

In the CSR_BT_HFG_SERVICE_CONNECT_IND received by the application there is a connection id parameter. This connection id shall be used when the application wants to communicate to this specific device.

Note: Only one connection attempt shall be made to a single device at the time – i.e. before trying to connect to the same device again, the pending connection and/or connection attempt must have been completed.

Note: After connection request has been sent, the application is not allowed to send any messages before it receives a CSR_BT_HFG_SERVICE_CONNECT_IND.

Note: Only one connection is allowed to one single device at a time. If the application tries to establish a connection to a device which it is already connected to, the CSR_BT_HFG_SERVICE_CONNECT_REQ primitive will be ignored, i.e. it will no be answered with a CSR_BT_HFG_SERVICE_CONNECT_IND.

3.3.3 Service Level Connection Completion

When the basic serial communication connection is established, the HF will issue a number of AT-commands to inform about and request application layer specific information. AT-commands issued by the HF will be interpreted and handled by the HFG manager. Likewise, the HFG manager provides interface for responding to these AT commands.

Should the HFG receive any AT-command which is does not recognise (vendor specific AT-commands), these can be sent, without any interpretation, to the application, or the profile can reply with the ERROR response code. Whether or not the AT-command is sent to the application is controlled by a parameter in the CSR_BT_HFG_ACTIVATE_REQ signal.

The HFG knows about all mandatory and optional AT-commands defined in the HF and HS specifications and will automatically generate the ERROR or OK responses. If un-interpreted commands are sent to the application, the application must send the ERROR or OK response itself and any other AT response command if applicable.

3.3.4 Service Level Connection Cancelling

If the application for some reason is trying to connect to the wrong device, it is possible to cancel a connection attempt using the CSR_BT_HFG_CANCEL_CONNECT_REQ. Because of race conditions, there are two possible scenarios when the application sends the cancel command and the HFG answers.

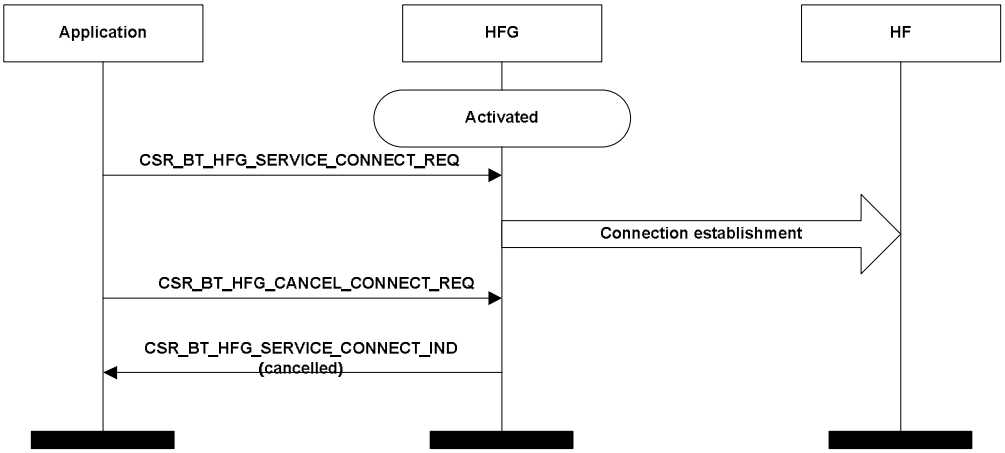


Figure 7: Cancel connection attempt

In Figure 7, the cancel command is received before the HFG has begun establishing the connection. This means that the outgoing connection attempt can be taken down immediately, and a CSR_BT_HFG_SERVICE_CONNECT_IND is sent to the application with the result code telling the application that the connection attempt was cancelled.

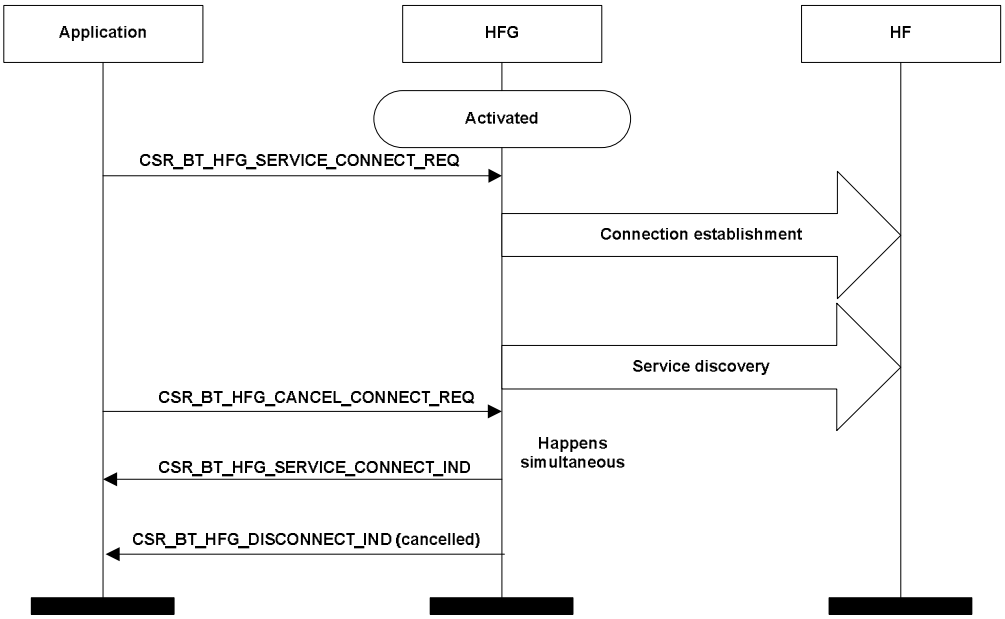


Figure 8: Cancel connection with crossing complete/cancel

In Figure 8, the CSR_BT_HFG_CANCEL_CONNECT_REQ is sent at the same time as the HFG has just sent the notification to the application that the service level connection has been established, i.e. a CSR_BT_HFG_SERVICE_CONNECT_IND with a *success* result code. When the HFG then receives the CSR_BT_HFG_CANCEL_CONNECT_REQ, it treats the command the same way as a normal disconnect command (a CSR_BT_HFG_DISCONNECT_REQ), and closes the connection. It then informs the application that the connection has been disconnected with a *cancelled* result code.

3.3.5 Low Power Modes

Once the service level connection is established, the HFG will, transparently for the application layer, make use of low power modes according to the rules described in [CCAP]; this implies use of park and sniff mode if supported by the HF. It also implies that the HFG manager may disconnect the service level connection and the application layer will have to re-establish a service level connection if the HF does not support low power modes. Note that the HFG supports both park and sniff mode, but the HFG profile will never actively try to enter park mode itself but only sniff mode. In order for the HFG to use park mode, the peer device must request it.

After the SLC has been established, the HFG will immediately try to enter sniff mode. If the peer device rejects sniff mode, 3 more attempts to enter sniff-mode will be made each with 3 seconds apart. If the final attempt also fails, no more retries will take place until the HFG sends data over the SLC to the device. Alternately, the HFG can also be setup to disconnect the SLC if the final sniff attempt fails.

3.4 Call Handling

Once a service level connection is established, the application layer can initiate an incoming call towards the HF. Further, the HF can initiate outgoing calls towards the HFG. If the remote device is a headset device, only the HFG profile manager can initiate a call. When connected to a headset that only supports the headset profile the HFG profile manager will hide (emulate) the differences between a HF and HS device. This means that signals normally only being supported when connected to a HF should also be sent to a headset device. See section 3.4.6 for details on how the HFG handles a headset connection.

3.4.1 Incoming Call Set-Up

When the application layer wants to indicate to the HF that an incoming call is coming and start ringing on the HF it should send a CSR_BT_HFG_STATUS_INDICATOR_SET_REQ (status indicators/CIEV) followed by a CSR_BT_HFG_RING_REQ to the HFG manager. The HFG manager will take care of the repeated ringing for the specified amount of time. If the ring is not answered or rejected by the HF device, the HFG will send a CSR_BT_HFG_RING_CFM to the application when finished ringing. The ring request must contain the A-number if available, and can also contain the alpha numeric name of the calling party. If the A-number is included and A-number transfer is enabled by the HF, the HFG manager will automatically send it following ring message, as specified in [HF]. If the remote device is a headset, the HFG will not send the A-number, because this is not supported by the HS profile. Furthermore the HS profile does not support the CSR_BT_HFG_STATUS_INDICATOR_SET_REQ but the information in these primitives will be used by the HFG to determine how to interpret the headset button press.

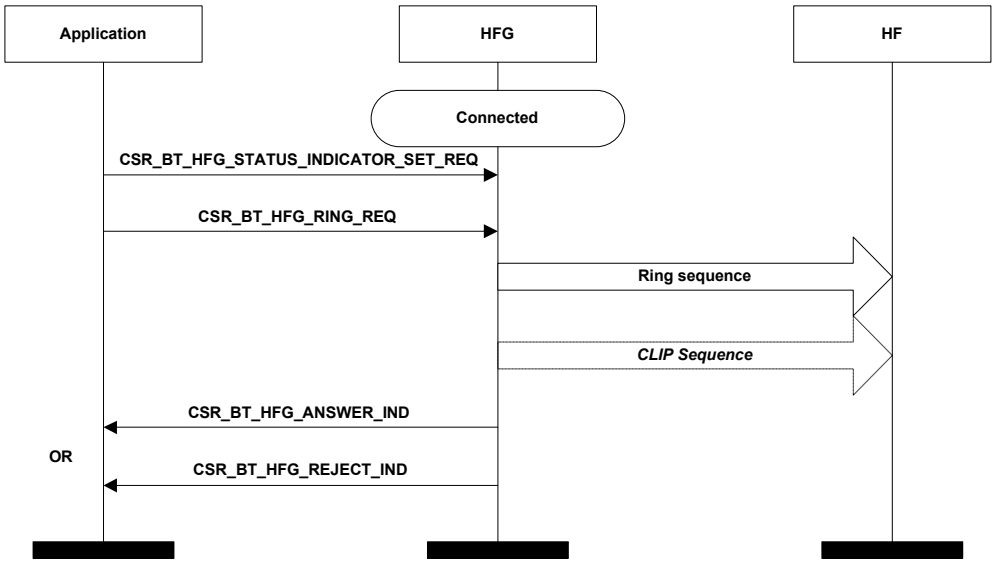


Figure 9: Initiate incoming call

The HF answer can be either accepted or rejected. If the HF accepts the call, a CSR_BT_HFG_ANSWER_IND is returned to the application layer and the call is established (with or without audio). If the HF for any reason wants to reject the call, a CSR_BT_HFG_REJECT_IND is returned to the application layer. In any case the application must send a CSR_BT_HFG_STATUS_INDICATOR_SET_REQ to the HFG after one of these primitives has been received. Also note that if the application receives a CSR_BT_HFG_ANSWER_IND or a CSR_BT_HFG_REJECT_IND any ongoing ring sequence will be stopped, but the application will not receive the CSR_BT_HFG_RING_CFM.

If the application wants to make use of in-band ringing, audio can be connected before the ring signal is sent to the HFG manager. Audio is connected using the CSR_BT_HFG_AUDIO_CONNECT_REQ signal. If no in-band ring is used, audio can be connected upon answer from the HF, i.e. CSR_BT_HFG_ANSWER_IND.

The application layer on the HFG side may also answer the call itself. If the HFG application layer wants to answer the call it is done by means of the CSR_BT_HFG_STATUS_INDICATOR_SET_REQ.

3.4.2 SCO and eSCO Parameters

Whenever the HFG wants to connect the audio connection, i.e. SCO or eSCO, it sends a CSR_BT_HFG_AUDIO_CONNECT_REQ. The HFG will automatically try to establish the best possible eSCO link and fallback to SCO if the peer device does not support eSCO. It is also possible to setup a custom set of (e)SCO settings, but it is **strongly** recommended that the default settings are used.

The (e)SCO setup procedure uses prioritised set of parameters that are tried in turn in order to get the best possible quality on the audio. The prioritised list of parameters is

1. User defined settings from the CSR_BT_HFG_CONFIG_AUDIO_REQ. If this signal has not been sent by the application, the default settings from the "*csr_bt_usr_config.h*" header file will be used.
2. "S3" from the Hands-Free 1.5 Profile specification (eSCO+EDR).
3. "S2" from the Hands-Free 1.5 Profile specification (eSCO+EDR).
4. "S1" from the Hands-Free 1.5 Profile specification (eSCO+EDR).
5. Regular SCO.

When the audio connection has been established (or disconnected), a CSR_BT_HFG_AUDIO_IND is sent to the application. This signal will also be sent to the application when a HF device connects the SCO connection to the HFG.

Besides, the CSR2CSR protocol described in chapter 3.7 allows the possibility to negotiate the audio codec and the sampling rate to be used in the audio connection, if the remote device supports it. This feature is called

Auristream. If the HFG has been activated and configured to negotiate these parameters, it will try to use these settings automatically.

3.4.3 Incoming Audio with Dynamic PCM mapping

The HFG profile supports multiple simultaneous (e)SCO connections, so it may be necessary for the application to setup the SCO-over-PCM routing, i.e. what PCM slot a particular SCO connection should use. This means that the signalling for an incoming audio connection will always start with the PCM mapping signals:

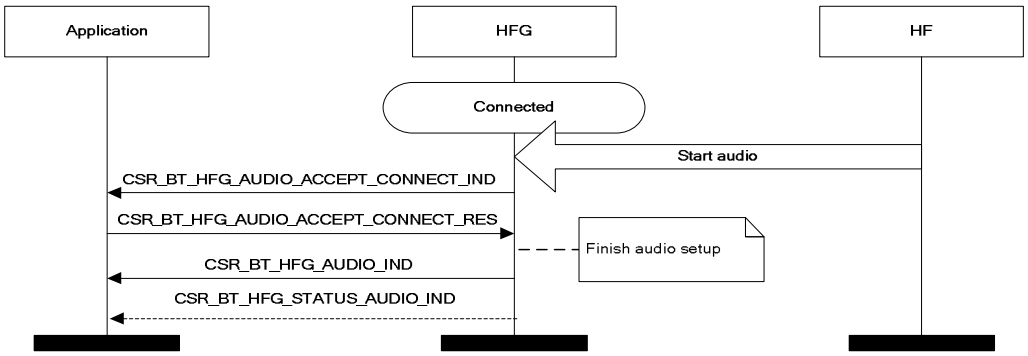


Figure 10: Incoming audio with dynamic PCM mapping

3.4.4 Outgoing Audio with Dynamic PCM Mapping

When requesting an outgoing audio connection the PCM slots are known in advance as they are passed to the HFG in the CSR_BT_HFG_AUDIO_CONNECT_REQ signal.

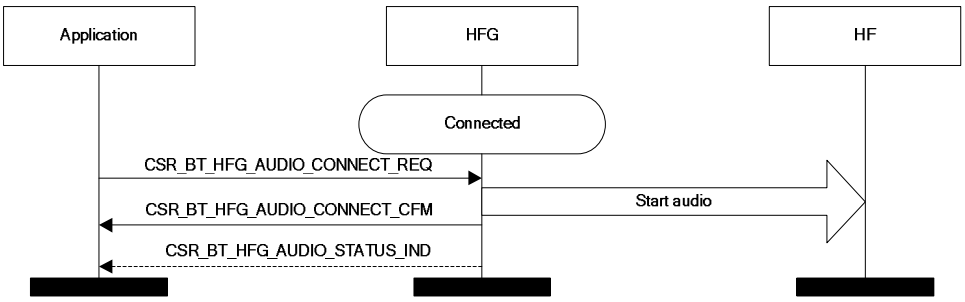


Figure 11: Outgoing audio initiation

3.4.5 Outgoing Call Set-Up

When the HF wants to initiate a new call it can be done, using one of the dial AT-commands. The application layer in the HFG will receive a dial out request in the CSR_BT_HFG_DIAL_IND. Outgoing calls can also be initiated by means of voice recognition which uses the CSR_BT_HFG_VOICE_RECOG_IND signal.

The table below shows what command is sent from the HFG to the application for number dial, memory dial, redial and voice recognition including the AT-command. It also shows the typical response the application uses in order to begin the call setup procedure.

Command	AT command	AT response
CSR_BT_HFG_DIAL_IND, dial specific number	ATD###	CSR_BT_HFG_DIAL_RES + CSR_BT_HFG_STATUS_INDICATOR_SET_REQ

CSR_BT_HFG_DIAL_IND, dial memory index	ATD>###	CSR_BT_HFG_DIAL_RES + CSR_BT_HFG_STATUS_INDICATOR_SET_REQ
CSR_BT_HFG_DIAL_IND, re-dial last number	AT+BLDN	CSR_BT_HFG_DIAL_RES + CSR_BT_HFG_STATUS_INDICATOR_SET_REQ
CSR_BT_HFG_VOICE_RECOG_IND	AT+BVRA	CSR_BT_HFG_VOICE_RECOG_RES + CSR_BT_HFG_STATUS_INDICATOR_SET_REQ + CSR_BT_HFG_AUDIO_CONNECT_REQ

Table 1: Outgoing calls setup and dial AT-commands

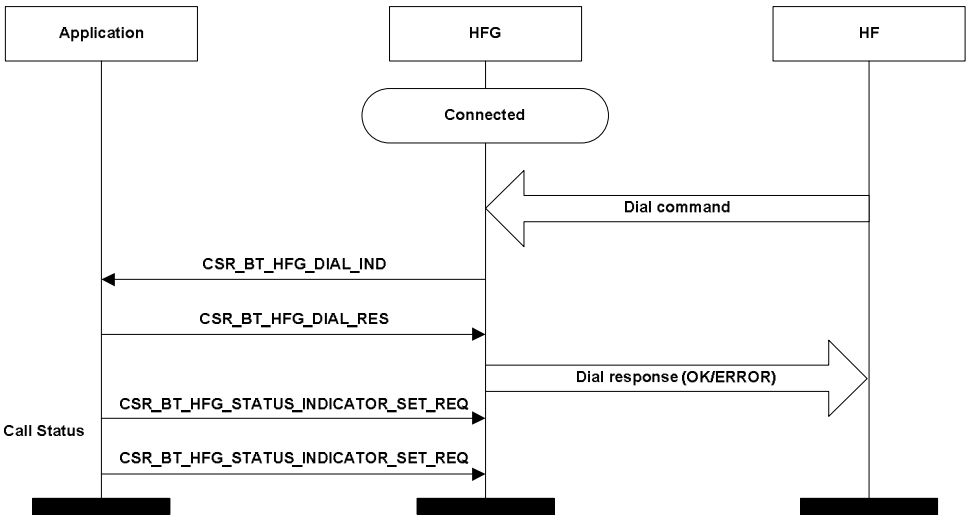


Figure 12: Outgoing call

During the call set-up process the application layer must continuously update the HF side with the current call and call setup status sending +CIEV status indicator updates via a CSR_BT_HFG_STATUS_INDICATOR_SET_REQ; further information can be found in [HF].

3.4.6 Call Handling with a Headset

The AT command interface available when connected to a headset is very limited compared to the AT command set available from a HF device. By using information about call status and ringing, the HFG is able to emulate the interface of a HF while connected to a HS. The interpretation of the headset button push in the HFG is described in Table 2.

Ring status	Call status	Audio On	Audio Off
Ringing	N/A	CSR_BT_HFG_ANSWER_IND	CSR_BT_HFG_ANSWER_IND
Not Ringing	Call not active	Transfer audio: Audio connection is turned off from the HFG	CSR_BT_HFG_VOICE_RECOG_IND or CSR_BT_HFG_DIAL_IND[1]
	Call active	CSR_BT_HFG_REJECT_IND	Transfer audio: Audio connection is turned on from the HFG

Table 2: Interpretations of the headset button push in the HFG

[1] The AT will depend on which features the HFG supports. If the HFG supports Voice Recognition activation this will be AT+BVRA+1 otherwise it will be AT+BLDN (Last number redial).

3.4.7 In-band Ringing to a Headset

When the application establishes an audio before sending the CSR_BT_HFG_RING_REQ the HFG will still send the unsolicited AT command RING but also expect the application to generate ring tones and sent it over the audio connection. This is implemented according to [ESR].

3.4.8 Status Indicators

The HFG supports a number of status indicators, which all are stored as long as the HFG is kept activated. The supported indicators are:

- Service indicator
- Call status indicator
- Call setup status indicator
- Signal strength indicator
- Roaming indicator
- Battery charge indicator
- Call held indicator

The status indicators are reported to the HF using the unsolicited result codes that are sent in the standard event reporting “+CIEV”. This signal is sent using the CSR_BT_HFG_STATUS_INDICATOR_SET_REQ signal. The CSR_BT_HFG_STATUS_INDICATOR_SET_REQ signal can be sent to the HFG whenever it has been activated and it will then update one or more connections depending on the value of the *connectionId* signal member, which is used for targeting specific connections.

One very suitable use of the CSR_BT_HFG_STATUS_INDICATOR_SET_REQ is whenever an application wants to update the signal strength or the battery charge status and this has to take place on all possible connections. The HFG will cache the last value sent to the HF for each connection, and will only send the unsolicited +CIEV event if there is any difference between the new and the old value. One exception to this rule is the call setup indicator, which must always be sent even if the value does not change.

Please refer to section 4.27 for further details of the signal and section 3.4.9 for a description of the *connectionId* concept.

3.4.9 Multiple Connections and Connection ID

The HFG is able to handle multiple simultaneous connections. This means that the application must have a way to target each specific connection, and a way to determine from what headset a specific command was sent from. For this purpose most of the HFG signal primitives contain a member named *connectionId* of the type `CsrBtHfgConnectionId`. The value of the member in signals sent to the application will always be the id of an actual connection.

When the application sends commands to the HFG, either the specific connection id can be used, or it can be set to a special value; *CSR_BT_HFG_CONNECTION_ALL*. The latter will send the command to *all* currently active connections. This is particularly useful for status indicator updates like battery- and signal strength changes. If a specific connection id is used, the command will only be sent to the targeted headset.

An example of multiple connections using CSR_BT_HFG_RING_REQ is given below.

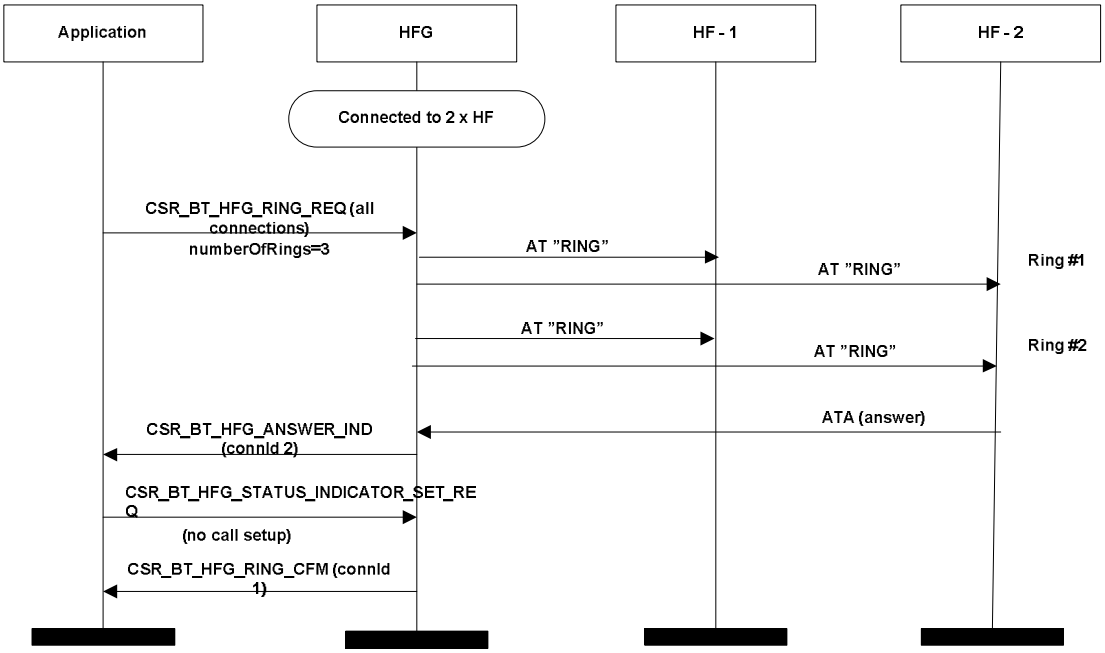


Figure 13: Ringing to multiple connections simultaneously

Figure 13 shows a simplified ringing process to two simultaneous connections, i.e. two headsets. For brevity the status indicators have been left out. In the sequence chart, the application can be seen to request the HFG to start ringing on all available connections using a CSR_BT_HFG_RING_REQ with the *connectionId* parameter set to CSR_BT_HFG_CONNECTION_ALL. The HFG receives the signal and starts the ringing process for both connections, with the number of rings set to 3.

After two “RING” AT-commands sent to both headsets, headset number 2 picks up the call. This automatically stops the ringing process for that connection. The headset sends the AT-command “ATA”, which is translated by the HFG into a CSR_BT_HFG_ANSWER_IND message. The message has the *connectionId* parameter set to *2*.

Even though headset 2 has answered the call, the ringing process for headset 1 continues until an indication that there is no longer an incoming call. The HFG sends a CSR_BT_HFG_RING_CFM to the application upon reception of this indication or when the configured number of rings has been sent, whatever happens first – the *connectionId* is in this case set to *1*.

3.4.10 Response and Hold

Please note that this section solely applies to handsets for CDMA or PDC networks.

Response and hold procedures allow for the possibility of putting an incoming call on hold from the HF or HFG and subsequently either accept or reject the call from the HF or HFG. This implies six different scenarios that are illustrated in Figure 14 to Figure 20.

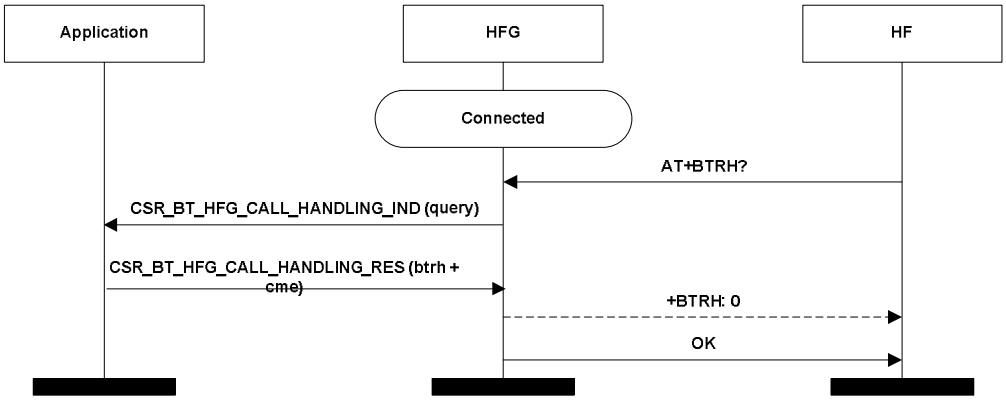


Figure 14: HF queries HFG for current "Response and Hold status"

It is important to note that the HFG uses the *call handling* signal primitive for both BTRH and three way calling.

Figure 14 indicates the procedure for the HF to request the response and hold state of the HFG. If the HFG is in a response and hold state it shall reply with '+BTRH: 0' followed by 'OK' otherwise it shall reply with only 'OK'.

The HFG profile includes a set of constants that eases the indication-and-response signalling. For example, the BTRH query will be sent in a CSR_BT_HFG_CALL_HANDLING_IND, and the reply is sent in a CSR_BT_HFG_CALL_HANDLING_RES with the BTRH code set to e.g. CSR_BT_HFG_BTRH_INCOMING_ON_HOLD and the result code set to CSR_BT_CME_SUCCESS.

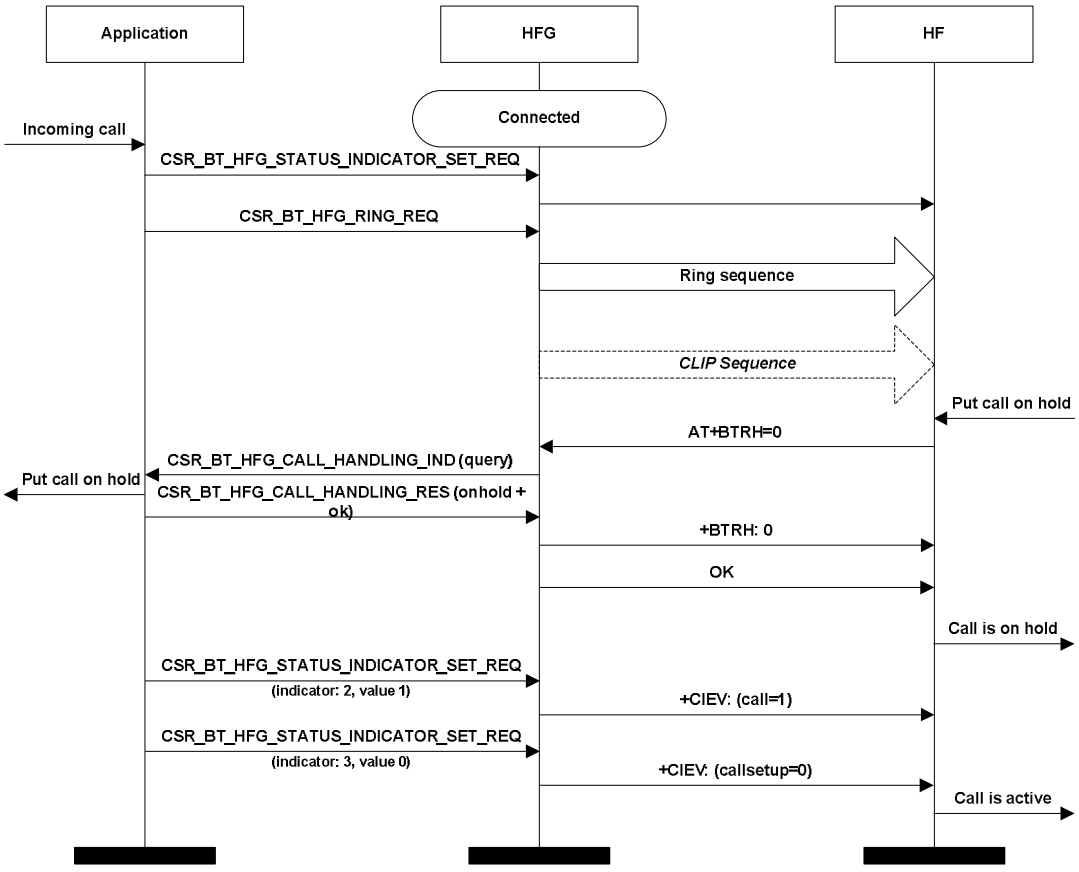


Figure 15: HF requests an incoming call to be put on hold

Figure 15 shows the procedure for the HF to request an incoming call to be put on hold. A precondition is that no calls are active or on hold in the HFG.

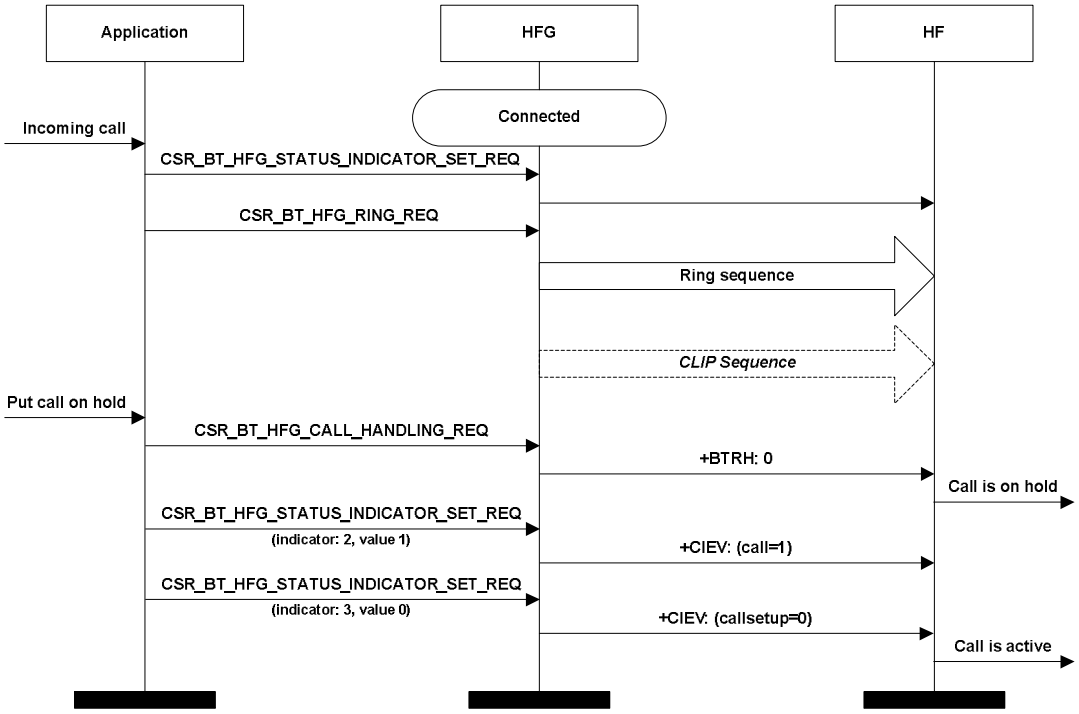


Figure 16: HFG requests an incoming call to be put on hold

Figure 16 displays the required procedure for the HFG to put an incoming call on hold. The same preconditions apply as in the previous scenario: No calls must be active or on hold in the HFG.

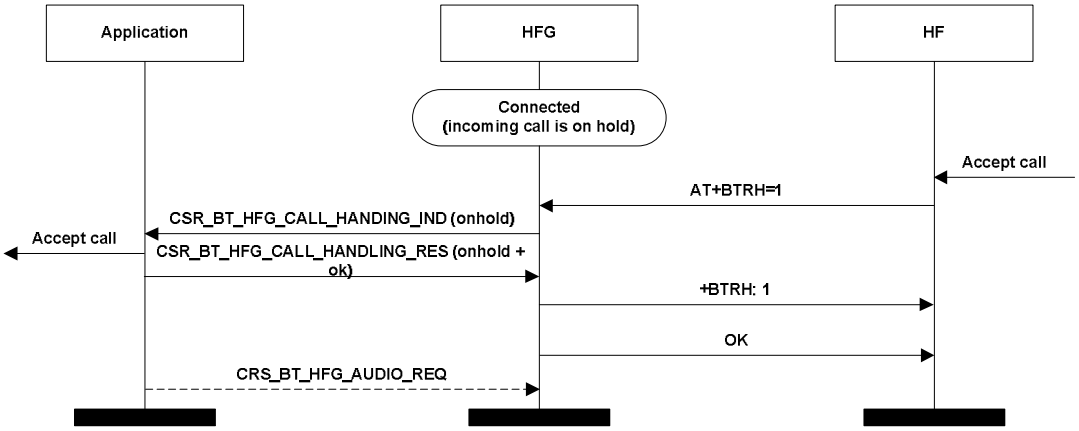


Figure 17: HF accepts a held incoming call

The procedure for the HF to accept a held call is illustrated in Figure 17. The HFG must conclusively open an audio connection if one is not open already.

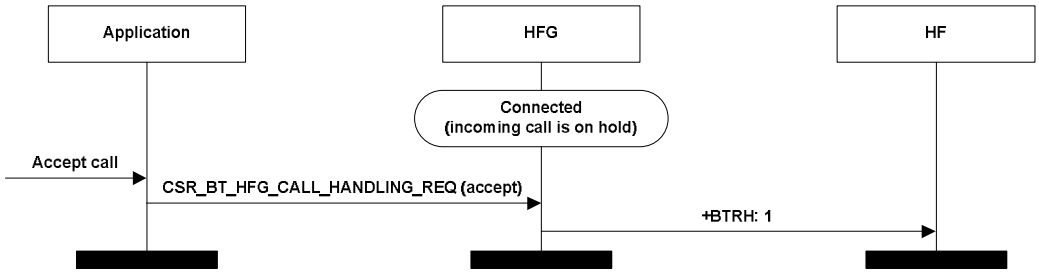


Figure 18: HFG accepts a held incoming call

Figure 18 describes the required procedure for the HFG to accept a held call.

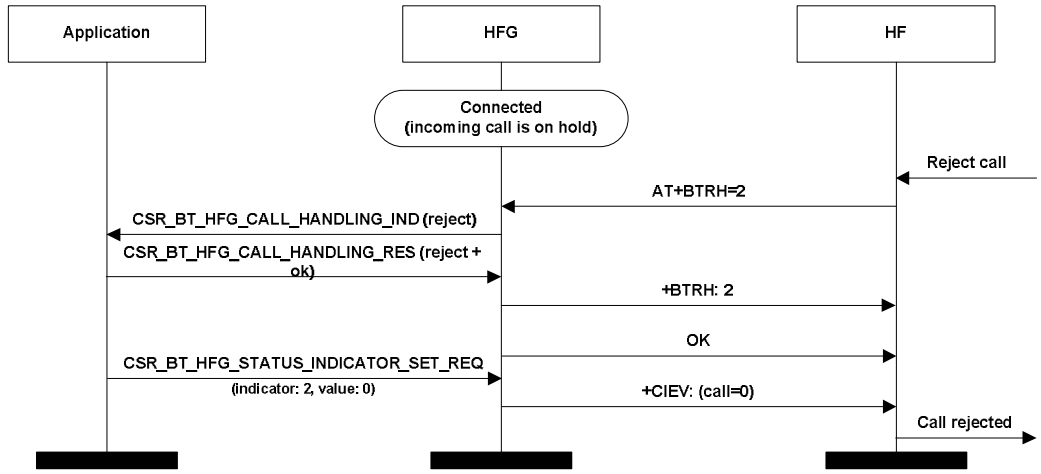


Figure 19: HF rejects a held incoming call

The procedure of rejecting a held call from the HF is depicted in Figure 19.

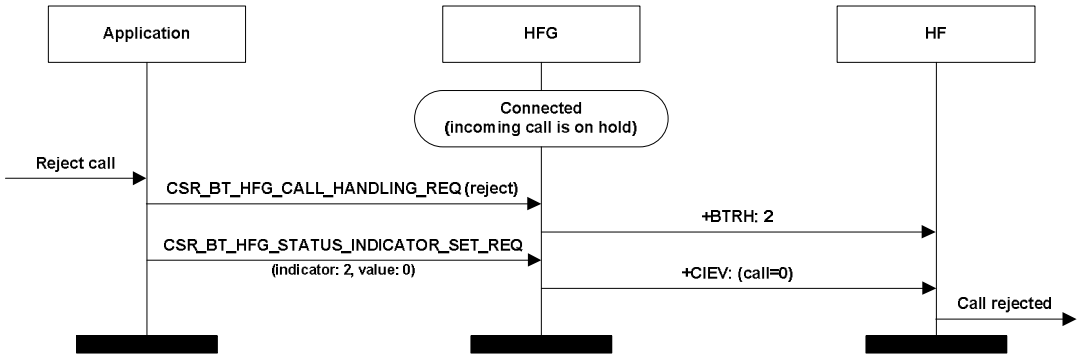


Figure 20: HFG rejects a held incoming call

Figure 20 displays the actions required to reject a held call from the HFG.

Support functions exist for making it more convenient to send the different AT-commands from the HFG. These are described in chapter 3.7.5 in relation to the signal they deal with. Further details about the response and hold features of the HFP can be found in [HF].

3.4.11 Three Way Call Handling

Three way calling in the hands-free profile is handled partly by means of the CHLD AT-commands and partly by means of status indicators. The use-cases for three way calling are quite complex and are outside the scope of

the HFG API specification to deal with all of them. Below is covered the typical use of three way calling and how the HFG profile signals fit into the scheme:

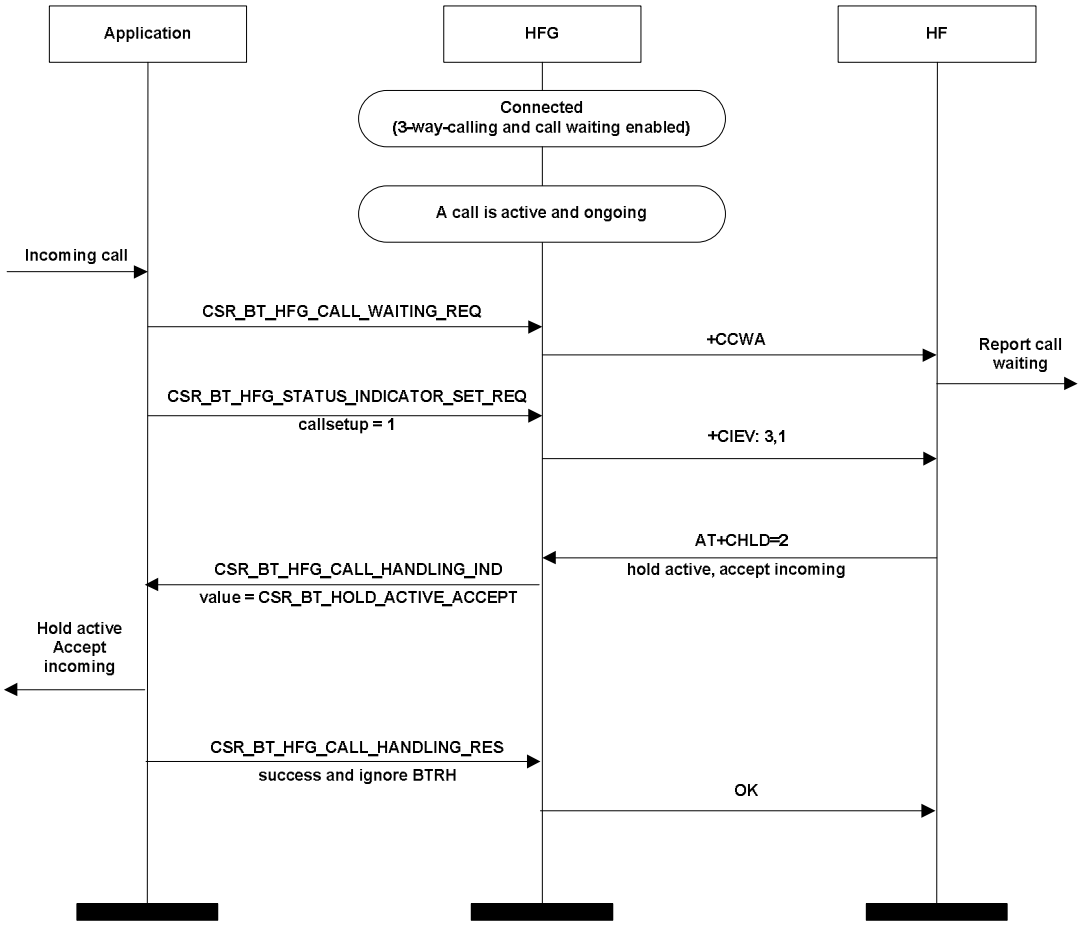


Figure 21: Typical use of three way calling

As can be seen from Figure 21, a call is ongoing while another party calls the gateway. The HFG notifies the HF about the waiting call using the CSR_BT_HFG_CALL_WAITING_REQ, and sets the *call setup* status indicator to *incoming call present*. The user picks up the call via the headset placing the current active call on hold, which is the command “AT+CHLD=2”. This message is relayed to the application as a CSR_BT_HFG_CALL_HANDLING_REQ with the value set to the special value CSR_BT_HOLD_ACTIVE_ACCEPT. The gateway puts the active call on hold and answers the other. As a call (and audio link) was already present, there is no need to establish that. The gateway finally finishes the three-way-calling procedure by the CSR_BT_HFG_CALL_HANDLING_RES with a success code, which is sent to the headset.

3.4.12 AT-Command Handling

Depending on which “AT mode” the application chooses when the HFG is activated, the handling of AT commands will take place differently. The HFG is capable of interpreting the entire set of the AT-commands defined in the hands-free profile. It can operate in four different modes as described in 4.2:

1. FULL mode. Where the HFG will handle all AT commands and map some of them into signal primitives. Commands not present in the table below will be answered with “ERROR” without announcement to the application.

2. SEMI mode. Similar to FULL mode but commands not included I the table below will be forwarded to the application in this case, if received after service level connection establishment.

3. TRANSPARENT mode. Where the HFG will forward all AT commands received to the application and not handle any of them. This applies for AT-commands needed for service level connection establishment as well. Only application developers familiar with the Hands free profile specification should use this mode.

4. USER CONFIG mode. Where the application can choose which AT-command it wants to handle and which ones the HFG shall handle exactly as it does in the SEMI and FULL modes. Only AT-commands received after service level connection establishment will be forwarded to the application.

AT-command	HFG signal primitive mapped to	Description	Default (for USER CONFIG mode)	Index (for USER CONFIG mode)
ATA	CSR_BT_HFG_ANSWER_IND	Answer call	0	1
AT+CHUP	CSR_BT_HFG_REJECT_IND	Hang-up / reject call	0	5
AT+CKPD=	N/A (handled in the HFG profile)	Button press (old HS)	1	4
AT+VGM=	CSR_BT_HFG_MIC_GAIN_IND	Microphone gain	1	2
AT+VGS=	CSR_BT_HFG_SPEAKER_GAIN_IND	Speaker gain	1	3
AT+ATD	CSR_BT_HFG_DIAL_IND	Dial number	0	9
AT+ATD>	CSR_BT_HFG_DIAL_IND	Speed / memory dial	0	8
AT+BLDN	CSR_BT_HFG_DIAL_IND	Re-dial last number	1	10
AT+CHLD=?	N/A (handled in the HFG profile)	3-way calling support	0	6
AT+CHLD=	CSR_BT_HFG_CALL_HANDLING_IND	3-way calling command	0	7
AT+BRSF=	N/A (handled in the HFG profile)	Supported features	1	11
AT+BTRH?	CSR_BT_HFG_CALL_HANDLING_IND	Response and hold support	1	12
AT+BTRH=	CSR_BT_HFG_CALL_HANDLING_IND	Response and hold command	1	13
AT+CIND?	N/A (handled in the HFG profile)	Status indicator update	1	14
AT+CIND=?	N/A (handled in the HFG profile)	Status indicator support	1	15
AT+CMER=	N/A (handled in the HFG profile)	Status notification setup	1	16
AT+CMEE=	N/A (handled in the HFG profile)	Extended error codes	0	17
AT+CCWA=	N/A (handled in the HFG profile)	Call waiting notification	0	18
AT+CLIP=	N/A (handled in the HFG profile)	Calling line notification	0	19
AT+CLCC	CSR_BT_HFG_CALL_LIST_IND	Obtain call list	0	20
AT+CNUM	CSR_BT_HFG_SUBSCRIBER_NUMBER_IND	Obtain subscriber number	0	21
AT+COPS?	N/A (handled in the HFG profile)	Operator name setup	0	22
AT+COPS=	CSR_BT_HFG_OPERATOR	Operator name query	0	23
AT+BVRA=	CSR_BT_HFG_VOICE_RECOG_IND / RES	Voice recognition toggle	1	24

AT-command	HFG signal primitive mapped to	Description	Default (for USER CONFIG mode)	Index (for USER CONFIG mode)
AT+BIA=	N/A (handled in the HFG profile)	Indicator notification activation	1	31
AT+NREC=	CSR_BT_HFG_NOISE_ECHO_IND	Noise/echo reduction	1	32
AT+VTS=	CSR_BT_HFG_GENERATE_DTMF_IND	DTMF tone generation	0	33
AT+BINP=	CSR_BT_HFG_BT_INPUT_IND	Bluetooth input (voice tag)	1	34
AT+CSR=	CSR_BT_HFG_C2C_SF_IND	CSR indicator setup	1	25
AT+CSRBATT=	CSR_BT_HFG_C2C_BATTERY_IND	CSR battery level	1	26
AT+CSRGETSMS=	CSR_BT_HFG_C2C_SMS_GET_IND	CSR SMS request	1	27
AT+CSRPWR=	CSR_BT_HFG_C2C_POWER_IND	CSR power level	1	28
AT+CSRSF=	CSR_BT_HFG_C2C_SF_IND	CSR supported features	1	29
AT+CSRFN=	CSR_BT_HFG_STATUS_AUDIO_IND	CSR Codec negotiation	1	30
Other commands	CSR_BT_HFG_AT_CMD_IND	AT command not handled by the HFG	1	0

Table 3: HFG manager upstream interpreted AT-commands

Other commands	CSR_BT_HFG_AT_CMD_IND	AT command not handled by the HFG	1	0

Table 3 lists all AT-commands known to the HFG, together with the HFG message which the AT-command is processed into in case the HFG shall handle the AT command. If the HFG signal primitive lists a “N/A”, this indicates that the HFG is able to process the specific AT-command entirely without notifying the application.

The HFG manager will interpret and validate the commands before sending the indication to the application layer. This implies then that the application layer does not need to send any OK or ERROR command to the HF side. The data payload is dynamically allocated and must be freed again by the application layer to prevent memory leaks.

Even though the HFG profile supports all HF Profile defined AT-commands and responses, it is also possible for the HFG application to send generic AT-responses or other unsolicited commands using the CSR_BT_HFG_AT_CMD_REQ. The application layer must compile the AT string itself and include it in the signal. Besides the CSR_BT_HFG_AT_CMD_REQ a number of specific commands are supplied in the HFG interface; these are:

The “Index” column in table 3 indicates the value or index that shall be used in connection with the primitive CSR_BT_HFG_CONFIG_SINGLE_AT_CMD_REQ (see 4.41) when operating in USER CONFIG mode, in order to choose whether to let the HFG handle a particular AT-command or not.
The “default” column indicates whether the command shall be handled by the HFG or whether it shall be forwarded to the application unhandled when operating in USER CONFIG mode. Value ‘1’ means the HFG handles the AT command; value ‘0’ indicates that it is forwarded to the application. Commands not known to the HFG will be answered with the “ERROR” response code if the default value is ‘1’ and forwarded to the application, otherwise.

Explanatory examples.

1. **Reception of a known AT command: "AT+VGM=7"**

   a. **In FULL mode**

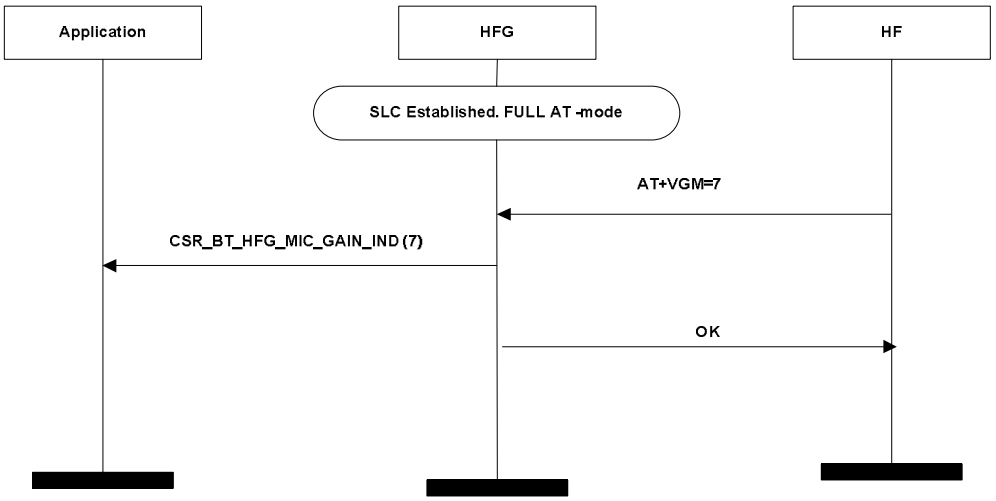


Figure 22: Handling of "AT+VGM=7" in FULL mode

   b. **In SEMI mode**

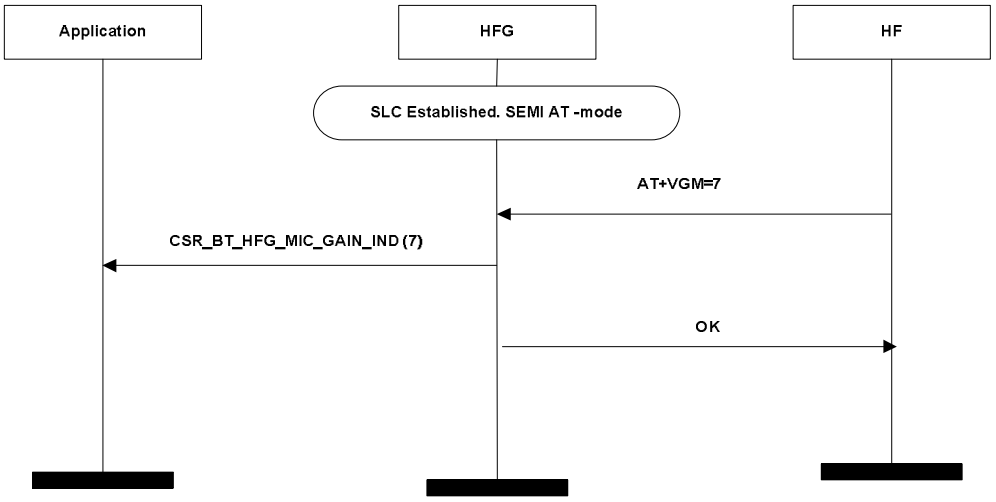


Figure 23: Handling of "AT+VGM=7" in SEMI mode

c. In TRANSPARENT mode

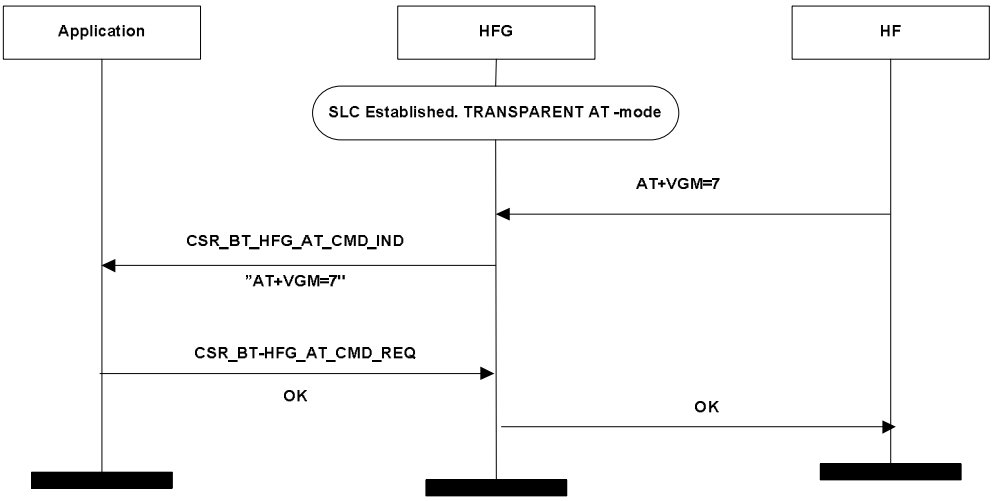


Figure 24: Handling of "AT+VGM=7" in TRANSPARENT mode

d. In USER CONFIG mode

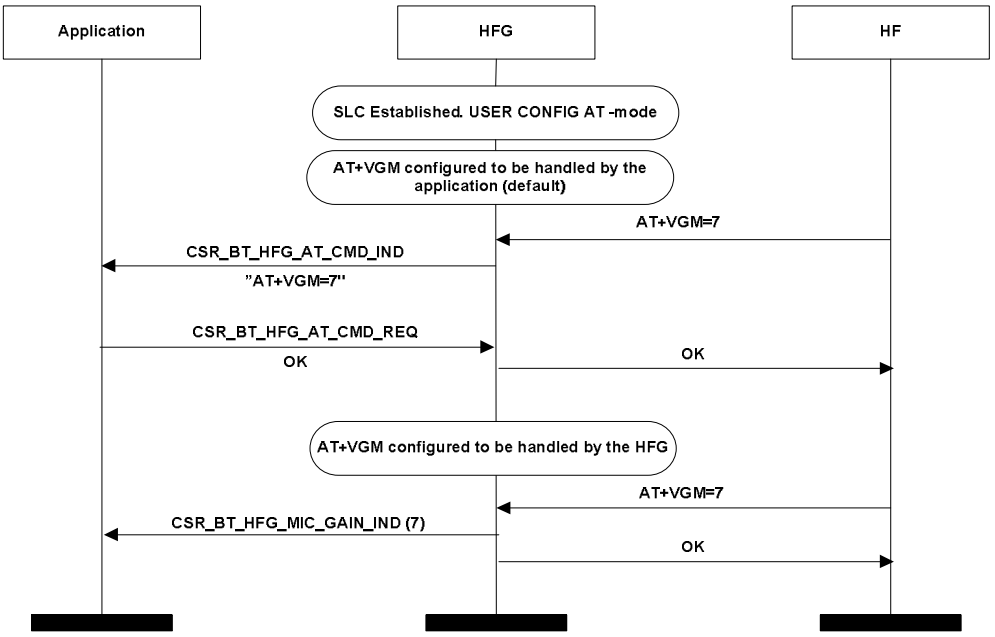


Figure 25: Handling of "AT+VGM=7" in USER CONFIG mode

2. **Reception of an unknown AT command: “AT+GMM”**

a. **In FULL mode**

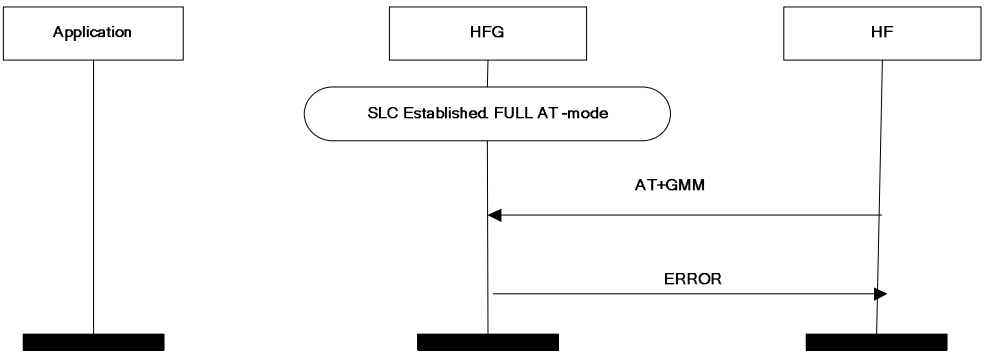


Figure 26: Handling of “AT+GMM” in FULL mode

b. **In SEMI mode**

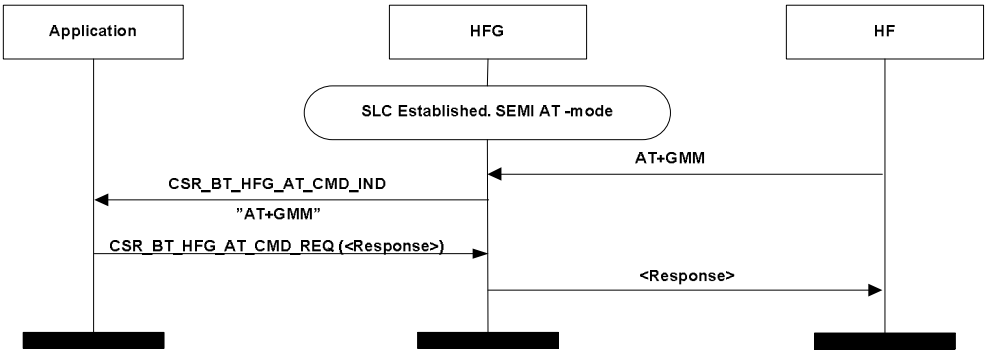


Figure 27: Handling of “AT+GMM” in SEMI mode

c. **In TRANSPARENT mode**

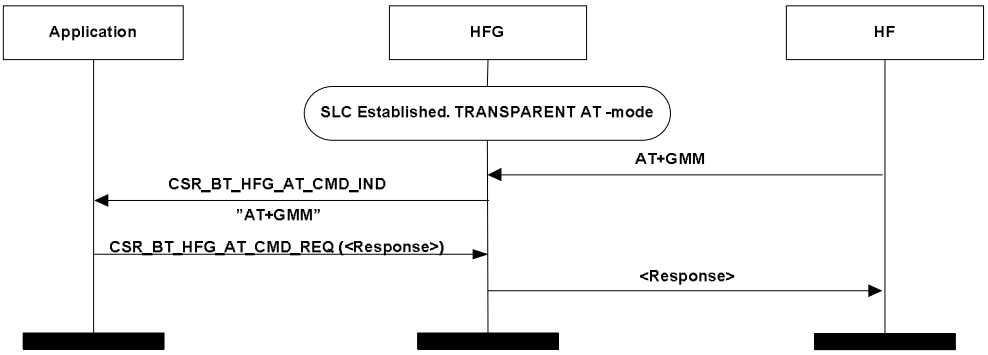


Figure 28: Handling of “AT+GMM” in TRANSPARENT mode

d. In USER CONFIG mode

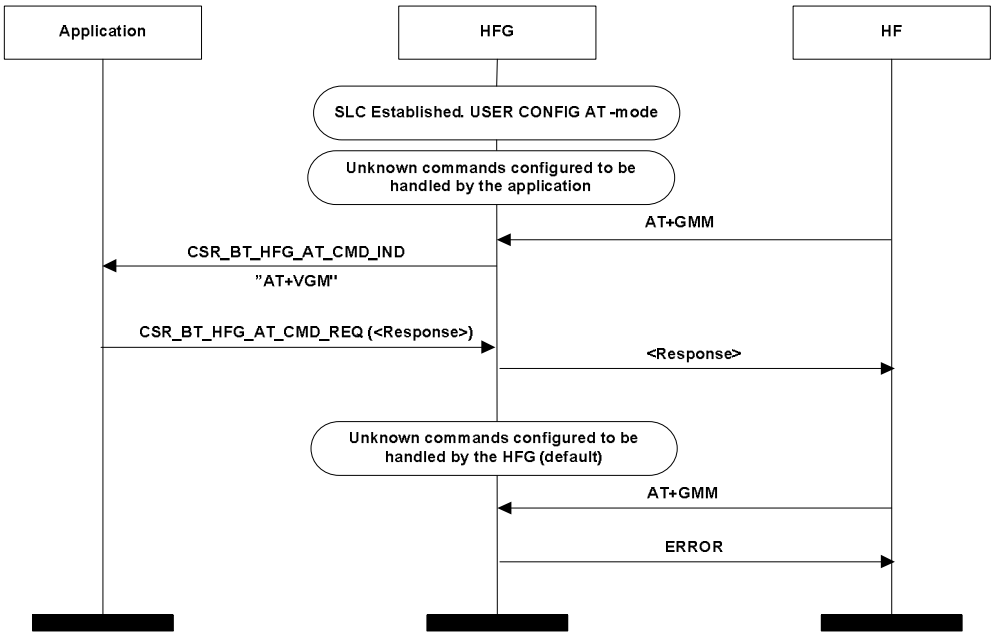


Figure 29: Handling of "AT+GMM" in USER CONFIG mode

AT-response	HFG signal primitive	Description
OK	N/A (handled in the HFG profile)	Command successful
ERROR	N/A (handled in the HFG profile)	Command error
RING	CSR_BT_HFG_RING_REQ	Ring
+VGS	CSR_BT_HFG_SPEAKER_GAIN_REQ	Speaker gain
+VGM	CSR_BT_HFG_MIC_GAIN_REQ	Microphone gain
+BRSF	N/A (handled in the HFG profile)	Supported features
+CLIP	CSR_BT_HFG_RING_REQ	Calling line identification
+COPS	CSR_BT_HFG_OPERATOR_RES	Operator name response
+CIEV	CSR_BT_HFG_STATUS_INDICATOR_SET_REQ	Status indicator update
+CHLD	CSR_BT_HFG_CALL_HANDLING_RES	BTRH / 3-way calling
+BTRH	CSR_BT_HFG_CALL_HANDLING_RES / REQ	BTRH / 3-way calling
+BSIR	CSR_BT_HFG_INBAND_RINGING_REQ	Toggle inband ringing
+BINP	CSR_BT_HFG_BT_INPUT_RES	Bluetooth input (voice tag)
+CLCC	CSR_BT_HFG_CALL_LIST_RES	Call list
+CNUM	CSR_BT_HFG_SUBSCRIBER_NUMBER_RES	Subscriber number response
+CCWA	CSR_BT_HFG_CALL_WAITING_REQ	Call waiting notification
+CME ERROR	N/A (handled in the HFG profile)	Extended error code
+BVRA	CSR_BT_HFG_VOICE_RECOG_REQ	Toggle voice recognition
+CIND	N/A (handled in the HFG profile)	Status indicator update
+CSRSF	CSR_BT_HFG_C2C_SF_REQ	CSR supported features
+CSRTXT	CSR_BT_HFG_C2C_TXT_REQ	CSR unsolicited TTS text
+CSRGETSMS	CSR_BT_HFG_C2C_SMS_ARRIVE_REQ	CSR SMS arrival

Table 4: AT-responses and unsolicited commands

The CSR_BT_HFG_STATUS_INDICATOR_SET_REQ command is interpreted in the HFG manager and may be used for rejecting an ongoing call process; thus the call process can be interrupted by sending the CSR_BT_HFG_STATUS_INDICATOR_SET_REQ with the value set to ‘no call set-up’.

The CSR_BT_HFG_RING_REQ command will ensure retransmission of the ring signal for the specified number of times with the specified ring interval. Ringing will automatically be interrupted upon answer or rejected from either the HF side or from the application layer.

Finally, a number of the responses may need to be sent several times as the HFG may have multiple subscriber numbers etc. These signal primitives all have a special member named *final* of the type *CsrBool*. This is used for telling the HFG whether or not the given signal is the last in a multi-signal sequence.

3.4.13 Unsupported and Disabled HF/HFG Features

Depending on the supported features of both the HFG and the HF to which a connection exists, different subsets of AT-commands may be available. In order to make it easier to write the HFG application, the HFG is able to filter out and ignore any signals sent to it that the HF does not support.

This means that even if the HF does not support *call waiting notifications* and the application sends a CSR_BT_HFG_CALL_WAITING_REQ to the HFG; the HFG will silently ignore and discard the message.

3.5 Service Level Connection Release

This section describes the different scenarios a fully established service level connection (SLC) can be released in. If a pending SLC setup is to be released, please see the description of the CSR_BT_HFG_CANCEL_CONNECT_REQ in section 3.3.4.

3.5.1 Connection Release

A service level connection is released by sending a CSR_BT_HFG_DISCONNECT_REQ to the HFG manager. A disconnect request is always confirmed with a CSR_BT_HFG_DISCONNECT_IND (and a CSR_BT_HFG_STATUS_IND, since the status of the connection has changed to LINK_STATUS_DISCONNECTED as defined in csr_bt_profiles.h).

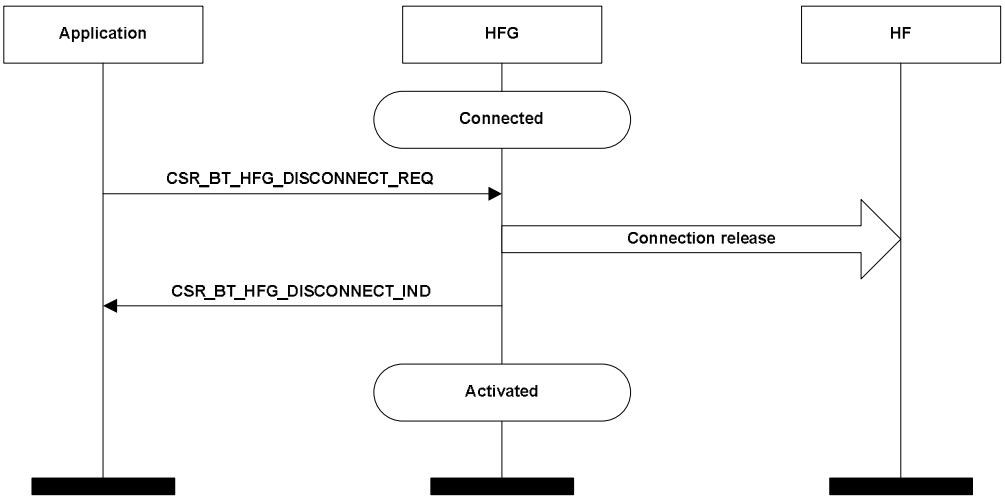


Figure 30: Connection release sequence

The application layer is not allowed to issue any new signals to the HFG until the disconnect request is confirmed.

The disconnect request is connection specific and will only release the connection specified through the parameter *connectionId*. It cannot be used for cancelling an attempt to establish a service level connection, please see section 3.3.4 for information about how to perform cancelling.

3.5.2 Peer Side Connection Release (Normal Release)

The HF at the remote end may at any time release the service level connection. A peer side connection release is indicated in the CSR_BT_HFG_DISCONNECT_IND result code.

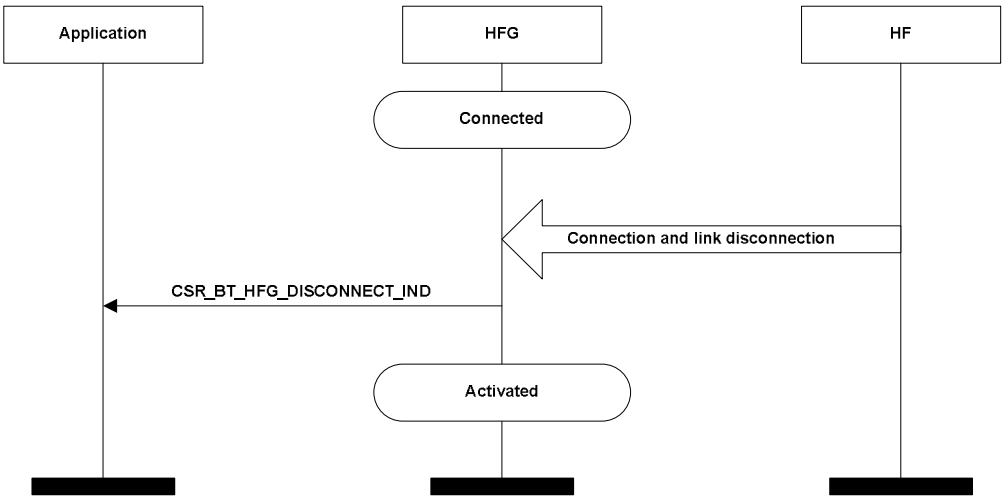


Figure 31: Peer side connection release sequence

The release is indicated to the application layer in a CSR_BT_HFG_DISCONNECT_IND. When the CSR_BT_HFG_DISCONNECT_IND is received for the last ongoing connection no further signaling is possible

until a new service level connection is established (see 3.3). After receiving the disconnect indication, the application layer cannot assume that the HF part has maintained the state and history of the call.

3.5.3 Link Loss Recovery (Abnormal Release)

It is possible that the physical link is closed due to an abnormal situation, e.g. radio interference or the devices getting out of coverage from each other. The HFG manager will handle a link loss by informing the application layer via a CSR_BT_HFG_DISCONNECT_IND with a release reason indicating an abnormal situation. After receiving the CSR_BT_HFG_DISCONNECT_IND, the application layer cannot assume that the HF part has maintained the state and history of the call.

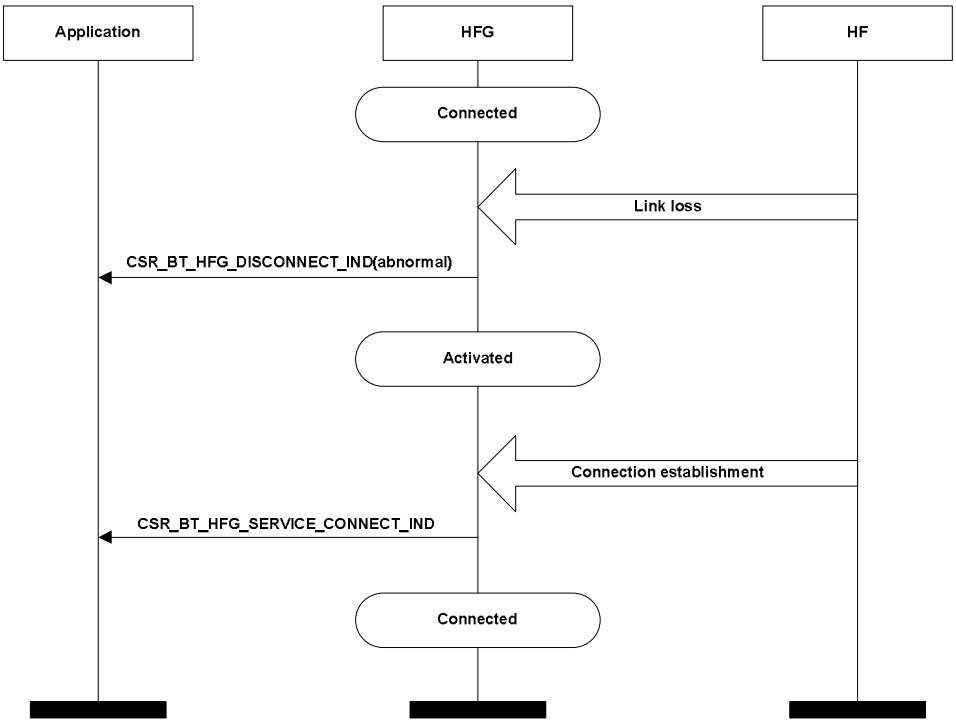


Figure 32: Peer side abnormal connection release sequence

After a disconnect, for whatever reason, the HFG manager enters the activated state, and is as such connectable and must be ready to receive a CSR_BT_HFG_SERVICE_CONNECT_IND message. To leave the activated state the application layer must explicitly instruct the HFG manager by issuing the deactivate request, as described in section 3.2.

3.6 Wide-Band Speech

The Wide-Band Speech (WBS) feature is an enhancement introduced in the HFP Bluetooth specification, version 1.6. To have access to this feature using the CSR Synergy BT implementation, the following conditions shall be met:

- The CSR chip used shall have DSP support
- The CSR Synergy BT code shall be compiled with the option “CSR_DSPM_ENABLE=1”
- The flag “CSR_BT_HFG_SUPPORT_CODEC_NEGOTIATION” shall be included in the “supportedFeatures” field of the HFG_ACTIVATE_REQ primitive.

Only if the remote device supports WBS too, will it be possible to establish a WBS audio connection between the two devices. The WBS feature mandates support for two audio formats: CVSD and mSBC, where the first is a narrow band format and the second is a wide band format.

3.7 CSR2CSR Features

CSR2CSR is the designation of a range of features usable between devices using respectively CSR Synergy Bluetooth for the HFG and BlueLab for the HF. Generally, these features provide the possibility of easily transferring information between two devices otherwise not easily possible. Currently the supported features consist of the following:

- Sending unsolicited text from the HFG to the HF
- Receiving battery level and power status from the HF
- Sending SMS (text message) notifications and messages to the HF
- Announcing incoming calls on the HF with both number and name of caller
- (**Deprecated: do not use!**)Negotiating the audio codec and/or sample rate to use for a determined audio connection (Auristream).

Before any CSR2CSR related functionalities can be used, they must be activated in the HFG and a service level connection must be established to a CSR2CSR capable HF.

3.7.1 Initialisation

The HFG supports the CSR2CSR features per default when it has been activated, but all features start disabled. This means that before any of the features can be enabled and used, a headset capable of utilising the CSR2CSR features must be connected to the HFG.

Upon connection of a CSR2CSR capable headset, a special AT-command will be sent to the HFG. The HFG will translate this message into a CSR_BT_HFG_C2C_SF_IND message, which indicates an enabled and supported CSR2CSR feature. After this, the HFG can enable the feature using a CSR_BT_HFG_C2C_SF_REQ, which will be sent to the headset, and the HFG and HF will at this point know which features are supported, and which are enabled.

Both CSR2CSR supported features (SF) signal primitives contain a number of indicators and corresponding values as listed in Table 5.

Indicator number	Indicator description	Indicator values
1	Support for presentation of name of callers	0: Disabled (default) 1: Enabled
2	Support for sending unsolicited text	0: Disabled (default) 1: Enabled
3	Support for sending SMS notifications and messages	0: Disabled (default) 1: Enabled
4	Support for presenting battery level	0: Disabled (default) 1: Enabled
5	Support for presenting power status	0: Disabled (default) 1: Enabled
6	Support for ADPCM negotiation	0: Disabled (default) 1: CVSD 2: 2-bit ADPCM 4: 4-bit ADPCM

7	Support for sample rate negotiation	0: Disabled (default) 1: 8 Khz sample rate 2: 16 Khz sample rate

Table 5: CSR2CSR indicator numbers and corresponding values

Please note that other indicator numbers and values are reserved for future use.

Note: Indicators number 6 and 7 in the table above are deprecated and will be removed from the implementation. It is recommended not to use them (give them the "Disabled" value as long as they are present in the code).

3.7.2 Example of CSR2CSR Feature Exchange

In Figure 33 an example of a CSR2CSR feature exchange is depicted.

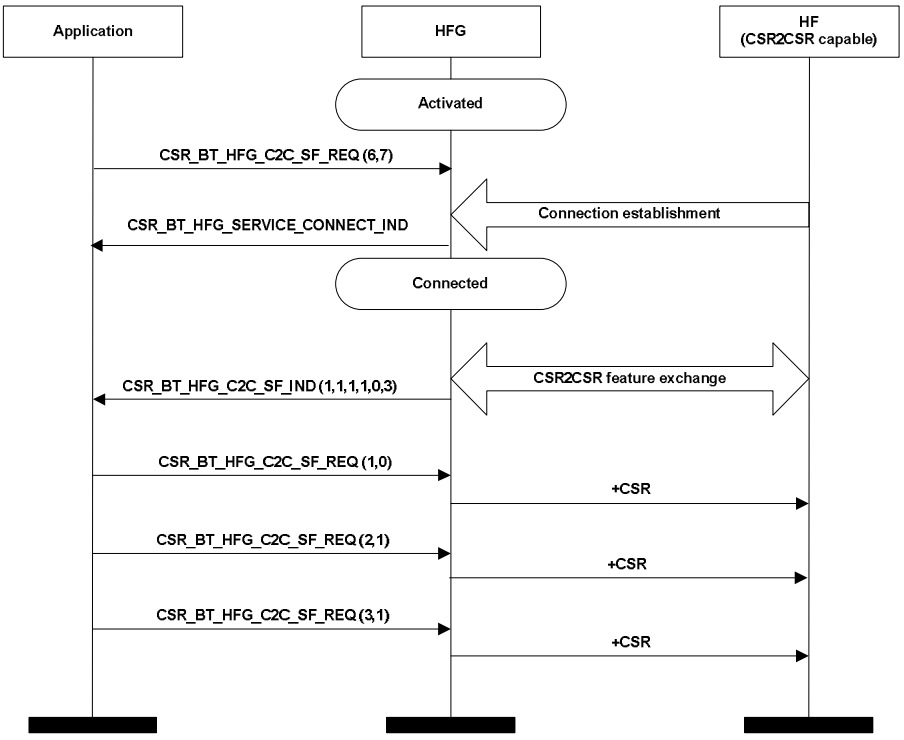


Figure 33: Example of CSR2CSR feature exchange

The application can configure, enable or disable any of the CSR2CSR supported features at any time after activation of the HFG profile.

After the SLC has been established, the HF will initiate the CSR2CSR supported features exchange. The HF supported features are sent to the application in a CSR_BT_HFG_C2C_SF_IND, which contains an array of indicator values as described in Table 5.

Following HF supported features, the application can use to enable support for presentation of names, unsolicited text and SMS messages.

In the example in Figure 33, the headset supports all features except *power status*, and CVSD is chosen as audio coding scheme as the indicator array consists of 4 ones,1 zero and 1 one. However, the HFG only enables

indicator 1, 2 and 3 which means that after the feature exchange, the HF and HFG will use the CSR2CSR features of *presentation of names*, *unsolicited text support* and finally *SMS notification and text support*.

If an audio connection is to be established towards the HF, the HFG will automatically negotiate the settings (ADPCM and/or sample rate) according to the HF capabilities and the preferences given by the application (see chapter 3.7.6 for details).

Note: The ADPCM and sample rate negotiation through the CSR to CSR proprietary AT commands is deprecated and will be removed from the implementation. It is recommended not to use it or indicate support for it to remote devices.

3.7.3 Sending Unsolicited Text from the HFG to the HF

This feature is useful with a handsfree device equipped with a display or a text-to-speech engine, as some kind of MMI is required to present the text either audibly or visually. In these cases the unsolicited text from the HFG can either be converted to speech or presented using a display on the HF. Unsolicited text can be sent from the HFG to the HF by using the CSR_BT_HFG_C2C_TXT_REQ primitive at any time after an established service level connection and if CSR2CSR is activated in the HFG and supported by the HF.

3.7.4 Receiving Battery Level and Power Status from the HF

By using this feature, the battery level and power status of the HF, can be sent to the HFG. The power status indicates whether the HF is powered by battery or by an external source. Support for battery level and power status can be activated separately. If the battery level or power status of the HF changes, the application will receive respectively a CSR_BT_HFG_C2C_BATTERY_IND or a CSR_BT_HFG_C2C_POWER_IND indication.

3.7.5 Sending SMS (text message) Notifications and Messages to the HF

The HFG can inform the HF about incoming text messages using the CSR_BT_HFG_C2C_SMS_ARRIVE_REQ. The HF can then notify the user visually or by speech and subsequently the user can choose to retrieve the entire message from the HFG. If the user requests the entire message, a CSR_BT_HFG_C2C_SMS_GET_IND indication will be sent to the application on the HFG, which can then send the message to the HF using CSR_BT_HFG_C2C_SMS_TXT_REQ.

3.7.6 Negotiating audio codec and sample rate

Note: This feature is deprecated and will be removed from the implementation. It is recommended not to use it or indicate support for it to remote devices.

If the HFG has been configured to it, it will automatically try to negotiate the audio settings to use during an audio connection if the HF has announced that it has support for it. If the negotiation fails, the HFG will fall back to use SCO settings for the connection.

Both the codec and sample rate are negotiated together. If the HF supports only codec negotiation, then that will be the only thing negotiated, and the sample rate used for the connection will be 8Khz.

The HFG will establish the audio connection itself, if the HF starts audio settings negotiation and both devices agree on the settings to use.

4 Hands-Free Gateway Primitives

This section gives an overview of the primitives and parameters in the interface. Detailed information can be found in the corresponding csr_bt_hfg_prim.h file.

To enable a fast integration process, library functions, which build each of the signals that are to be sent to the Hands-Free Gateway, are available. These are described in the library file csr_bt_hfg_lib.h, which also defines the valid information elements that are to be included when sending a signal.

4.1 List of All Primitives

Primitives	Reference
CSR_BT_HFG_ACTIVATE_REQ	See section 4.2
CSR_BT_HFG_DEACTIVATE_REQ	See section 4.3
CSR_BT_HFG_DEACTIVATE_CFM	See section 4.3
CSR_BT_HFG_SERVICE_CONNECT_REQ	See section 4.4
CSR_BT_HFG_SERVICE_CONNECT_IND	See section 4.4
CSR_BT_HFG_CANCEL_CONNECT_REQ	See section 4.5
CSR_BT_HFG_DISCONNECT_REQ	See section 4.6
CSR_BT_HFG_DISCONNECT_IND	See section 4.6
CSR_BT_HFG_AUDIO_CONNECT_REQ	See section 4.7
CSR_BT_HFG_AUDIO_CONNECT_CFM	See section 4.7
CSR_BT_HFG_AUDIO_CONNECT_IND	See section 4.7
CSR_BT_HFG_AUDIO_ACCEPT_CONNECT_IND	See section 4.8
CSR_BT_HFG_AUDIO_ACCEPT_CONNECT_RES	See section 4.8
CSR_BT_HFG_AUDIO_DISCONNECT_REQ	See section 4.9
CSR_BT_HFG_AUDIO_DISCONNECT_CFM	See section 4.9
CSR_BT_HFG_AUDIO_DISCONNECT_IND	See section 4.9
CSR_BT_HFG_SECURITY_IN_REQ	See section 4.10
CSR_BT_HFG_SECURITY_IN_CFM	See section 4.10

Primitives	Reference
CSR_BT_HFG_SECURITY_OUT_REQ	See section 4.10
CSR_BT_HFG_SECURITY_OUT_CFM	See section 4.10
CSR_BT_HFG_CONFIG_SNIFF_REQ	See section 4.11
CSR_BT_HFG_CONFIG_AUDIO_REQ	See section 4.12
CSR_BT_HFG_STATUS_LP_IND	See section 4.13
CSR_BT_HFG_STATUS_AUDIO_IND	See section 4.14
CSR_BT_HFG_RING_REQ	See section 4.15
CSR_BT_HFG_RING_CFM	See section 4.15
CSR_BT_HFG_ANSWER_IND	See section 4.16
CSR_BT_HFG_REJECT_IND	See section 4.17
CSR_BT_HFG_CALL_WAITING_REQ	See section 4.18
CSR_BT_HFG_CALL_HANDLING_REQ	See section 4.19
CSR_BT_HFG_CALL_HANDLING_IND	See section 4.19
CSR_BT_HFG_CALL_HANDLING_CFM	See section 4.19
CSR_BT_HFG_DIAL_IND	See section 4.20
CSR_BT_HFG_DIAL_RES	See section 4.20
CSR_BT_HFG_SPEAKER_GAIN_REQ	See section 4.21
CSR_BT_HFG_SPEAKER_GAIN_IND	See section 4.21
CSR_BT_HFG_MIC_GAIN_REQ	See section 4.22
CSR_BT_HFG_MIC_GAIN_IND	See section 4.22
CSR_BT_HFG_AT_CMD_REQ	See section 4.23
CSR_BT_HFG_AT_CMD_IND	See section 4.23
CSR_BT_HFG_OPERATOR_IND	See section 4.24
CSR_BT_HFG_OPERATOR_RES	See section 4.24
CSR_BT_HFG_CALL_LIST_IND	See section 4.25
CSR_BT_HFG_CALL_LIST_RES	See section 4.25
CSR_BT_HFG_SUBSCRIBER_NUMBER_IND	See section 4.26
CSR_BT_HFG_SUBSCRIBER_NUMBER_RES	See section 4.26
CSR_BT_HFG_STATUS_INDICATOR_SET_REQ	See section 4.27
CSR_BT_HFG_INBAND_RINGING_REQ	See section 4.28
CSR_BT_HFG_GENERATE_DTMF_IND	See section 4.29
CSR_BT_HFG_NOISE_ECHO_IND	See section 4.30
CSR_BT_HFG_BT_INPUT_IND	See section 4.31
CSR_BT_HFG_BT_INPUT_RES	See section 4.31
CSR_BT_HFG_VOICE_RECOG_REQ	See section 4.32
CSR_BT_HFG_VOICE_RECOG_IND	See section 4.32
CSR_BT_HFG_C2C_SF_REQ	See section 4.33
CSR_BT_HFG_C2C_SF_IND	See section 4.33
CSR_BT_HFG_C2C_TXT_REQ	See section 4.34
CSR_BT_HFG_C2C_SMS_ARRIVE_REQ	See section 4.35
CSR_BT_HFG_C2C_SMS_GET_IND	See section 4.36
CSR_BT_HFG_C2C_SMS_TXT_REQ	See section 4.37

Primitives	Reference
CSR_BT_HFG_C2C_BATTERY_IND	See section 4.38
CSR_BT_HFG_C2C_POWER_IND	See section 4.39
CSR_BT_HFG_MANUAL_INDICATOR	See section 4.40
CSR_BT_HFG_CONFIG_SINGLE_ATCMD_REQ	See section 4.41
CSR_BT_HFG_CONFIG_SINGLE_ATCMD_CFM	See section 4.41
CSR_BT_HFG_CONFIG_ATCMD_HANDLING_REQ	See section 4.42
CSR_BT_HFG_CONFIG_ATCMD_HANDLING_CFM	See section 4.42
CSR_BT_HFG_GET_C2C_ADPCM_LOCAL_SUPPORTED_REQ	See section 4.43
CSR_BT_HFG_GET_C2C_ADPCM_LOCAL_SUPPORTED_IND	See section 4.43
CSR_BT_HFG_DEREGISTER_TIME_REQ	See section 4.44
CSR_BT_HFG_DEREGISTER_TIME_CFM	See section 4.44

Table 6: List of all primitives

4.2 CSR_BT_HFG_ACTIVATE

Primitives	type	phandle	atMode	numConnections	*serviceName	supportedFeatures	callConfig	hfgConfig
CSR_BT_HFG_ACTIVATE_REQ	✓	✓	✓	✓	✓	✓	✓	✓

Table 7: The CSR_BT_HFG_ACTIVATE Primitive

Description

This signal is used for registering the service record with the supported features available in the Hands-Free Gateway profile and making the device connectable. This message shall be the first message to be sent. The primitive provides means to control the advanced behaviour of the HFG-profile, the maximum number of simultaneous connections etc.

Parameters

type — The signal identity, CSR_BT_HFG_ACTIVATE_REQ.

phandle — The identity of the calling process, to which all indications shall be sent.

atMode — This defines how the AT-command parser shall function. Allowed values are:

- *CSR_BT_HFG_AT_MODE_FULL*
  The HFG will process *all* AT-commands, including unknown commands. This is the recommended mode.
- *CSR_BT_HFG_AT_MODE_SEMI*
  The HFG will process all known AT-commands. Unknown commands are forwarded to the application in a HFG_AT_CMD_IND
- *CSR_BT_HFG_AT_MODE_TRANSPARENT*
  The HFG AT parser is completely disabled, and all AT-commands are forwarded to the application in CSR_BT_HFG_AT_CMD_IND primitives.
- *CSR_BT_HFG_AT_MODE_USER_CONFIG*
  The HFG will process those AT-commands that the application has configured it to, and will forward all others to the application, after SLC establishment. See chapter 3.4.12 for details

numConnections — Maximum number of simultaneous connections – allowed values are 1 and 2.

*serviceName — The service name string of the HFG to be set in the service record. Note that the HFG will free the pointer after the primitive has been received. A value of *NULL* will use the default.

supportedFeatures — This parameter defines the Hands-Free Profile specification *supported features* bitmask. The bits must be OR'ed together to form a bitmask, and the allowed bits are defined below (and in the csr_bt_hf.h header file):

- CSR_BT_HFG_SUPPORT_THREE_WAY_CALLING
- CSR_BT_HFG_SUPPORT_EC_NR_FUNCTION

- CSR_BT_HFG_SUPPORT_VOICE_RECOGNITION
- CSR_BT_HFG_SUPPORT_INBAND_RINGING
- CSR_BT_HFG_SUPPORT_ATTACH_NUMBER_TO_VOICE_TAG
- CSR_BT_HFG_SUPPORT_ABILITY_TO_REJECT_CALL
- CSR_BT_HFG_SUPPORT_ENHANCED_CALL_STATUS
- CSR_BT_HFG_SUPPORT_ENHANCED_CALL_CONTROL
- CSR_BT_HFG_SUPPORT_EXTENDED_ERROR_CODES
- CSR_BT_HFG_SUPPORT_CODEC_NEGOTIATION

Note: It is recommended to enable *all* features. Unused bits *shall* be left to 0.

callConfig

The call configuration parameter controls what kinds of enhanced call control the HFG supports when the AT parser is in full or semi mode. The parameter is a bitmask that can be OR'ed together using the following values:

- CSR_BT_HFG_CRH_DISABLE_BTRH
  Disable response-and-hold (AT+BTRH and +BTRH)
- CSR_BT_HFG_CRH_DISABLE_CHLD_0
  Disable 'release all held calls' (AT+CHLD=0)
- CSR_BT_HFG_CRH_DISABLE_CHLD_1
  Disable 'release active call' (AT+CHLD=1)
- CSR_BT_HFG_CRH_DISABLE_CHLD_1X
  Disable 'release specific call' (AT+CHLD=1,n)
- CSR_BT_HFG_CRH_DISABLE_CHLD_2
  Disable 'hold active accept' (AT+CHLD=2)
- CSR_BT_HFG_CRH_DISABLE_CHLD_2X
  Disable 'private consultation mode' (AT+CHLD=2,n)
- CSR_BT_HFG_CRH_DISABLE_CHLD_3
  Disable 'add call' (AT+CHLD=3)
- CSR_BT_HFG_CRH_DISABLE_CHLD_4
  Disable 'connect two calls' (AT+CHLD=4)

Note: The disabling of the different features shall be set dependent of which features the product supports, e.g. many GSM phones support the most of the CHLD features while CDMA one-based products very often support the BTRH feature. However, all the features can be enabled by default and then it is up to the application to decide in each situation whether the requested command can be executed. Unused bits *shall* be left to 0.

hfgConfig

The HFG profile configuration is used for controlling advanced features. The parameter is a bitmask that can be OR'ed together using the following values:

- CSR_BT_HFG_CNF_LP_STATUS
  Subscribe to the special CSR_BT_HFG_STATUS_LP_IND primitives
- CSR_BT_HFG_CNF_AUDIO_STATUS

Subscribe to the special CSR_BT_HFG_STATUS_AUDIO_IND primitives

- CSR_BT_HFG_CNF_DISABLE_C2C
  Disable support for the CSR2CSR features completely
- CSR_BT_HFG_CNF_DISABLE_BIA
  Disable support for the special AT+BIA command
- CSR_BT_HFG_CNF_MANUAL_INDICATORS
  Halt the SLC AT startup sequence and require the application to answer the CIND manually using the CSR_BT_ HFG_MANUAL_INDICATORS_IND/RES signal.
- CSR_BT_HFG_CNF_ DISABLE_COPS
  Disable support for operator name request in the app. The HFG answers appropriately to the HF.
- CSR_BT_HFG_CNF_ ENABLE_EXTENDED_AUDIO
  The HFG sends an extended audio indication message to the application upon establishment of an audio connection, including information about the codec and sample rate used for the connection.
- CSR_BT_HFG_CFN_EXIT_LP_ON_AUDIO
  The HFG will try to exit LP mode when audio connection is established and enter it again if enabled upon audio disconnection
- CSR_BT_HFG_CNF_DISABLE_ESCO_TO_OLD_DEVICES
  The HFG will only try to establish SCO connections (not eSCO) to devices supporting HFP versions older than 1.05, where eSCO was first specified.

Note: Setting these flags may change the default behavior of the HFG. Unused bits *shall* be left to 0.

Library Function

```
void CsrBtHfgActivateReqSend( CsrSchedQid            phandle,
                              CsrBtHfgAtParserMode   atMode,
                              CsrUint8               numConnections,
                              CsrCharString          *serviceName,
                              CsrUint32              supportedFeatures,
                              CsrUint32              callConfig,
                              CsrUint32              hfgConfig);
```

All parameters are as described in the primitive details above.

4.3 CSR_BT_HFG_DEACTIVATE

Primitives \ Parameters	type
CSR_BT_HFG_DEACTIVATE_REQ	✓
CSR_BT_HFG_DEACTIVATE_CFM	✓

Table 8: The CSR_BT_HFG_DEACTIVATE Primitives

Description

Interrupt an already ongoing activity in the HFG manager. Depending on the actual state of the HFG manager this may include disconnection of an established service level connection. When the application has received the CSR_BT_HFG_DEACTIVATE_CFM message, the profile has been deactivated and it is no longer possible to use the HFG (until the application activates it again using CSR_BT_HFG_ACTIVATE_REQ).

Parameters

type The signal identity, CSR_BT_HFG_DEACTIVATE_REQ/CFM.

Library Function

```
void CsrBtHfgDeactivateReqSend(  void   )
```

4.4 CSR_BT_HFG_SERVICE_CONNECT

Primitives \ Parameters	type	deviceAddr	connectionType	connectionId	serviceName	supportedFeatures	resultCode	resultSupplier
CSR_BT_HFG_SERVICE_CONNECT_REQ	✓	✓	✓					
CSR_BT_HFG_SERVICE_CONNECT_IND	✓	✓	✓	✓	✓	✓	✓	✓

Table 9: The CSR_BT_HFG_SERVICE_CONNECT Primitives

Description

Setup and establish a service level connection between the HFG and the HF specified by the Bluetooth® device address. Please note that before a complete service level connection is established a number of AT-commands must be exchanged (see 3.1). In case the remote device does not start the AT command exchange sequence, the HFG device has a connection guard timer to ensure that the connection is released eventually and the application gets an answer to its connection request. The timer is defined in the file csr_bt_usr_config.h with the name “CSR_BT_HFG_CONNECTION_GUARD_TIME” and is set to 30 seconds per default.

Note that the service connection indication is used both for locally requested (outgoing) connections and when a headset connects to the HFG (incoming connections).

Please note that it is illegal to issue multiple connections to the same device, unless the previous connection and/or connection attempt has been completed.

Parameters

type — The signal identity, CSR_BT_HFG_SERVICE_CONNECT_REQ/IND.

deviceAddr — The Bluetooth® device address to which a connection must be established.

connectionType — The type of connection to connect to, or the type of connection that has just been established. The possible values are:

- CSR_BT_HFG_CONNECTION_UNKNOWN
  Only possible in the REQ – this will make the HFG autodetect the type of connection and is the recommended value.
- CSR_BT_HFG_CONNECTION_HFG
  Connection to or from a Hands-Free device
- CSR_BT_HFG_CONNECTION_AG
  Connection to or from a Headset device

connectionId — This is the connection index, which is used for uniquely identifying this particular connection from other headsets. It is vital that the application saves this id for future reference.

serviceName — Text defined by the HF service record as a zero-terminated string. Note that this member is passed as an array.

supportedFeatures — The supported features bitmask of the hands-free device to which a connection has

been established. The possible values are defined by the Hands-Free Specification. Compiler defines for easy usage can be found in the csr_bt_hf.h header file, and are:

- CSR_BT_HF_SUPPORT_EC_NR_FUNCTION
- CSR_BT_HF_SUPPORT_CALL_WAITING_THREE_WAY_CALLING
- CSR_BT_HF_SUPPORT_CLI_PRESENTATION_CAPABILITY
- CSR_BT_HF_SUPPORT_VOICE_RECOGNITION
- CSR_BT_HF_SUPPORT_REMOTE_VOLUME_CONTROL
- CSR_BT_HF_SUPPORT_ENHANCED_CALL_STATUS
- CSR_BT_HF_SUPPORT_ENHANCED_CALL_CONTROL

resultCode	The result code of the operation. Possible values depends on the value of resultSupplier. If eg. the resultSupplier == CSR_BT_SUPPLIER_CM then the possible result codes can be found in csr_bt_cm_prim.h. All values which are currently not specified in the respective prim.h file are regarded as reserved and the application should consider them as errors.
resultSupplier	This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h

Library Function

```
void CsrBtHfgServiceConnectReqSend(deviceAddr_t  deviceAddr,
                                   CsrBtHfgConnection connectionType)
```

All parameters are as described above in the signal primitive details.

4.5 CSR_BT_HFG_CANCEL_CONNECT

Parameters / Primitives	type	deviceAddr
CSR_BT_HFG_CANCEL_CONNECT_REQ	✓	✓

Table 10: The CSR_BT_HFG_CANCEL_CONNECT primitive

Description

The CSR_BT_HFG_CANCEL_CONNECT_REQ primitive is used for cancelling an ongoing connection establishment. It is used for cancelling the connection attempt right after the CSR_BT_HFG_SERVICE_CONNECT_REQ. When the connection has been cancelled the HFG profile manager will send a CSR_BT_HFG_SERVICE_CONNECT_IND with result code of *CSR_BT_CANCELLED_CONNECT_ATTEMPT*. If the CSR_BT_HFG_CANCEL_CONNECT_REQ is received in the HFG after a CSR_BT_HFG_SERVICE_CONNECT_IND is sent, the HFG will disconnect from the device with the specified device address and send a CSR_BT_HFG_DISCONNECT_IND with a result code of *CSR_BT_CANCELLED_CONNECT_ATTEMPT*.

Parameters

type	The signal identity, CSR_BT_HFG_CANCEL_CONNECT_REQ.
deviceAddr	The Bluetooth® device address to which a connection must be established.

Library Function

```
void CsrBtHfgCancelConnectReqSend (deviceAddr_t  deviceAddr)
```

All parameters are as described above in the signal primitive details.

4.6 CSR_BT_HFG_DISCONNECT

Primitives \ Parameters	type	connectionId	reasonCode	reasonSupplier	localTermianted	deviceAddr
CSR_BT_HFG_DISCONNECT_REQ	✓	✓				
CSR_BT_HFG_DISCONNECT_IND	✓	✓	✓	✓	✓	✓

Table 11: The CSR_BT_HFG_DISCONNECT Primitives

Description

This signal will disconnect an established service level connection. The application will receive a disconnect indication when the connection has been released. Note that the same indication is used for both locally requested disconnect and when the headset disconnects from the HFG. After the connection has been released, it will automatically accept new connections.

Parameters

type	The signal identity, CSR_BT_HFG_DISCONNECT_REQ/IND.
connectionId	The identification number of the connection which is to be or has been disconnected.
reasonCode	The reason code of the operation. Possible values depends on the value of reasonSupplier. If eg. the reasonSupplier == CSR_BT_SUPPLIER_CM then the possible reason codes can be found in csr_bt_cm_prim.h. All values which are currently not specified in the respective prim.h files are regarded as reserved and the application should consider them as errors.
reasonSupplier	This parameter specifies the supplier of the reason given in reasonCode. Possible values can be found in csr_bt_result.h
localTerminated	TRUE if termination of connection happened on request from the local host; FALSE otherwise.
deviceAddr	The Bluetooth® device address of the connection that has been released.

Library Function

```
void CsrBtHfgDisconnectReqSend(  CsrBtHfgConnectionId      connectionID     )
```

The parameter is as described in the signal primitive above.

4.7 CSR_BT_HFG_AUDIO_CONNECT

Primitives \ Parameters	type	connectionId	scoHandle	pcmSlot	pcmRealloc	linkType	txInterval	weSco	rxPacketLength	txPacketLength	airMode	codecUsed	sampleRate	resultCode	resultSupplier
CSR_BT_HFG_AUDIO_CONNECT_REQ	✓	✓		✓	✓										
CSR_BT_HFG_AUDIO_CONNECT_IND	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
CSR_BT_HF_AUDIO_CONNECT_CFM	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Table 12: CSR_BT_HFG_AUDIO_CONNECT Primitives

Description

Request for audio establishment. The audio must be controlled by the application due to the interface for external audio, e.g. GSM.

If the application has issued the CSR_BT_HFG_CONFIG_AUDIO_REQ primitive to set the audio quality prior to the CSR_BT_HFG_AUDIO_CONNECT_REQ message, then the settings given in the configuration message will be used by the HFG. Thus, the application decides the audio quality. Please note that a higher quality requires more bandwidth and more power consumption from the battery. If the CSR_BT_HFG_CONFIG_AUDIO_REQ has not been issued, it is up to the profile to decide the best quality available at the given time.

Parameters

Type	The signal identity, CSR_BT_HFG_AUDIO_CONNECT_REQ/IND.
connectionId	The connection to request audio on.
scoHandle	Sco handle if routed internally
pcmSlot	Pcm slot to use
pcmRealloc	If TRUE, then reallocate automatically if pcm-slot chosen is in use
LinkType	Link type used in the audio connection (0x00 = SCO, 0x02 = eSCO)
txInterval	Transmission interval in baseband slots (Set to zero for SCO links)
WeSco	Retransmission window in baseband slots (Set to 0 for SCO links)
rxPacketLength	RX packet length in bytes for eSCO connection (Set to 0 for SCO links)
txPacketLength	TX packet length in bytes for eSCO connection (Set to 0 for SCO links)
AirMode	The air mode: 0x00 = my-law; 0x01 = A-law; 0x02 = CVSD; 0x03 = Transparent data
codecUsed	Codec used for the audio connection established
sampleRate	Sample rate used for the audio connection established

resultCode	The result code of the operation. Possible values depend on the value of resultSupplier. If e.g. the resultSupplier == CSR_BT_SUPPLIER_CM then the possible result codes can be found in csr_bt_cm_prim.h. All values which are currently not specified in the respective prim.h file are regarded as reserved and the application should consider them as errors.
resultSupplier	This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h

Library Function

```
void CsrBtHfgAudioConnectReqSend(CsrBtHfgConnectionId connectionId, CsrUint8
                                 pcmSlot,              CsrBool pcmRealloc)
```

Example

Here is an example of how to send the request signal using the library function.

```
CsrBtHfgAudioConnectReqSend(conId, 0x01, FALSE)
```

Examples for generic (e)SCO packet bit patterns:

The packet type support for negotiating the audio quality is specified in a 2 octet bit pattern as shown here.

						3EV5	2EV5	3EV3	2EV3	EV5	EV4	EV3	HV3	HV2	HV1

Note that the EDR eSCO packets (2EV3, 3EV3, 2EV5 and 3EV5) have their enable bits inverted. The grayed out section are bits reserved for future use and their setting is irrelevant for the application interfacing the hands-free profile manager. In the examples below the reserved bits are set to 1.

Enable all:
Should all packets be available for while negotiating the quality setting for the audio link between devices, the HV bits and EV bits should be set. The fact that the EDR eSCO packets have their enable bits inverted means that they should not be set to enable them.

Set HV1 + HV2 + HV3 + EV3 + EV4 + EV5 and not 2EV3 + 3EV3 + 2EV5 + 3EV5
1111110000111111 = 0xFC3F

For setting this bit pattern, the following packet definitions from hci.h should be combined:

```
HCI_ESCO_PKT_HV1 | HCI_ESCO_PKT_HV2 | HCI_ESCO_PKT_HV3 | HCI_ESCO_PKT_EV3 |
HCI_ESCO_PKT_EV4 | HCI_ESCO_PKT_EV5
```

Specifying only SCO:
Should only SCO packets be available, the HV and the EDR eSCO bits should be set.

Set HV1 + HV2 + HV3 and 2EV3 + 3EV3 + 2EV5 + 3EV5
1111111111000111 = 0xFFC7

For setting this bit pattern, the following packet definitions from hci.h should be combined:

```
HCI_ESCO_PKT_HV1 | HCI_ESCO_PKT_HV2 | HCI_ESCO_PKT_HV3 | HCI_ALL_MR_ESCO
```

Specifying only eSCO:
Should only the eSCO packet types be available, the EV bits should be set and the EDR eSCO bits should not.

Set EV3 + EV4 + EV5 and not 2EV3 + 3EV3 + 2EV5 + 3EV5
1111110000111000 = 0xFC38

For setting this bit pattern, the following packet definitions from hci.h should be combined:

HCI_ESCO_PKT_HV1 | HCI_ESCO_PKT_HV2 | HCI_ESCO_PKT_HV3

Description of the type CsrBtHfgAudioLinkParameterListConfig;

packetType	Specifies which SCO/eSCo packet types to use
txBandwitdh	Specifies the maximum Transmission bandwidth to use
rxBandwitdth	Specifies the maximum Receive bandwidth to use
maxLatency	Specifies the maximum Latency to use.
voiceSettings	Specifies the voice coding used in the transmission. Eg. Csvd or Transparent.
reTxtEffort	Specifies the Retransmission setting to use.

4.8 CSR_BT_HFG_AUDIO_ACCEPT_CONNECT

Primitives \ Parameters	type	connectionId	linkType	acceptResponse	*acceptParameters	acceptParamteresLength	pcmSlot	pcmReassign
CSR_BT_HFG_AUDIO_ACCEPT_CONNECT_IND	✓	✓	✓					
CSR_BT_HFG_AUDIO_ACCEPT_CONNECT_RES	✓			✓	✓	✓	✓	✓

Table 13: CSR_BT_HFG_AUDIO_ACCEPT_CONNECT Primitives

Description

The CSR_BT_HFG_AUDIO_ACCEPT_CONNECT_IND is sent to the application layer during the initiation of a remotely initiated audio connection. The application layer must then send a CSR_BT_HFG_AUDIO_ACCEPT_CONNECT_RES to the HF profile in order to decide whether to accept an incoming audio connection, and if so what settings to accept and which PCM port the incoming connection must be mapped to.

Note that the dynamic PCM port selection will only work if PSKEY_HOSTIO_MAP_SCO_PCM is set to FALSE.

Note also that the contents of the “acceptParameters” may be overruled with default values in the HF profile in case an eSCO/SCO connection exists beforehand, as the settings already used for the existing connection might have an influence on what settings are possible in the new one.

Parameters

Type	The signal identity, CSR_BT_HFG_AUDIO_ACCEPT_CONNECT_IND/RES.
connectionId	The identification number of the connection.
LinkType	Specifies SCO/eSCO
acceptResponse	The HCI response code from profile can be one of the following: HCI_SUCCESS,HCI_ERROR_REJ_BY_REMOTE_NO_RES, HCI_ERROR_REJ_BY_REMOTE_PERS Note: If this is not HCI_SUCCESS then the incoming SCO/eSCO connection will be rejected
acceptParameters	Specifies which SCO/eSCO parameters to accept. If NULL the default ACCEPT SCO parameters from csr_bt_usr_config.h or CSR_BT_HFG_CONFIG_AUDIO_REQ are used. This pointer is of the type CsrBtHfgAudioIncomingAcceptParameters and is described below.
acceptParametersLength	Shall be 1 if acceptParameters != NULL, otherwise 0
pcmSlot	Map this audio connection to the specified PCM slot. Range is 0-4, where 0 means that audio will be routed through CSR Synergy Bluetooth, and 1-4 corresponds to PCM slot 1-4 on the BlueCore chip. If the parameter is set to CSR_BT_PCM_DONT_CARE defined in csr_bt_profiles.h, the first available PCM slot will be assigned.
pcmRealloc	Automtically assign another pcm-slot if pcmSlot given in this response is already in

use. The resulting pcmSlot will be informed in the CSR_BT_HFG_AUDIO_IND.

Library Function

```
void CsrBtHfgAudioAcceptConnectResSend(CsrBtHfgConnectionId      connectionId,
                                       hci_error_t acceptResponse,
                                       CsrBtHfgAudioIncomingAcceptParameters
                                       *acceptParameters, CsrUint8
                                       acceptParametersLength
                                       CsrUint8   pcmSlot,
                                       CsrBool   pcmRealloc)
```

Example

Here is an example of how to send the request signal using the library function.

```
   CsrBtHfgAudioAcceptConnectResSend(indPrim->connectionId, HCI_SUCCESS, NULL,
1,TRUE)
```

Description of the type CsrBtHfgAudioIncomingAcceptParameters:

packetTypes	Specifies which SCO/eSCo packet types to accept
txBandwitdh	Specifies the maximum Transmission bandwidth to accept
rxBandwitdth	Specifies the maximum Receive bandwidth to accept
maxLatency	Specifies the maximum Latency to accept
contentFormat	Specifies which SCO/eSCO content format to accept
reTxtEffort	Specifies the Retransmission setting to accept.

4.9 CSR_BT_HFG_AUDIO_DISCONNECT

Primitives \ Parameters	type	connectionId	scoHandle	resultCode	resultSupplier	reasonCode	reasonSupplier
CSR_BT_HFG_AUDIO_DISCONNECT_REQ	✓	✓					
CSR_BT_HFG_AUDIO_DISCONNECT_CFM	✓	✓	✓	✓	✓		
CSR_BT_HFG_AUDIO_DISCONNECT_IND	✓	✓	✓			✓	✓

Table 14: CSR_BT_HFG_AUDIO_DISCONNECT Primitives

Description

The application can release an audio connection by means of the CSR_BT_HFG_AUDIO_DICONNECT_REQ primitive. When the connection is actually released, the profile will send a confirm message back to the application. However, if the audio connection is released by the remote device, or due to a link loss, the profile will send a CSR_BT_HFG_DISCONNECT_IND to the application instead, to inform it about the release of the connection.

Parameters

type	The signal identity, CSR_BT_HFG_AUDIO_DICONNECT_REQ /IND/CFM
connectionId	The identification number of the connection.
scoHandle	Sco handle if routed internally.NB: If this parameter is set as 0xFFFF in the CSR_BT_HFG_AUDIO_DISCONNECT_REQ the profile will disconnect all established audio connections to the associated connectionId and it will send a CSR_BT_HFG_AUDIO_DISCONNECT_CFM for each audio connection that is disconnected. Otherwise it should be set to the right scoHandle as received in a respective CSR_BT_HFG_AUDIO_CONNECT_IND/CFM.
resultCode	The result code of the operation. Possible values depend on the value of resultSupplier. If e.g. the resultSupplier == CSR_BT_SUPPLIER_CM then the possible result codes can be found in csr_bt_cm_prim.h. All values which are currently not specified in the respective prim.h file are regarded as reserved and the application should consider them as errors.
resultSupplier	This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h
reasonCode	The reason code of the operation. Possible values depend on the value of reasonSupplier. If e.g. the reasonSupplier == CSR_BT_SUPPLIER_CM then the possible reason codes can be found in csr_bt_cm_prim.h. All values which are currently not specified in the respective prim.h files are regarded as reserved and the application should consider them as errors.
reasonSupplier	This parameter specifies the supplier of the reason given in reasonCode. Possible values can be found in csr_bt_result.h

Here is an example of how to send the request signal using the library function.

```
CsrBtHfgAudioDisconnectReqSend(indPrim->connectionId)
```

4.10 CSR_BT_HFG_SECURITY_IN / _OUT

Primitives \ Parameters	type	appHandle	secLevel	resultCode	resultSupplier
CSR_BT_HFG_SECURITY_IN_REQ	✓	✓	✓		
CSR_BT_HFG_SECURITY_IN_CFM	✓			✓	✓
CSR_BT_HFG_SECURITY_OUT_REQ	✓	✓	✓		
CSR_BT_HFG_SECURITY_OUT_CFM	✓			✓	✓

Table 15: The CSR_BT_HFG_SECURITY_IN and CSR_BT_HFG_SECURITY_OUT Primitives

Description

Applications that want to change the enforcement to a specific profile security level, i.e. authentication, encryption and/or authorisation, can use this API to setup the security level for *new* connections. Note that this API is for the local device only and can be used from within any state.

The *CSR_BT_SECURITY_IN_REQ* signal sets up the security level for new incoming connections. Already established or pending connections are not altered.

The *CSR_BT_SECURITY_OUT_REQ* signal sets up the security level for new outgoing connections. Already established and pending connections are not altered. Note that *authorisation* should not be used for outgoing connections as that may be confusing for the user – there is really no point in requesting an outgoing connection and afterward having to authorise as both are locally-only decided procedures.

Note, that any attempts to set security to a less secure level than the mandatory security level will be rejected. See csr_bt_profiles.h for mandatory security settings. The default settings used by CSR Synergy Bluetooth are set to require authentication and encryption.

Note that if MITM protection is requested and the remote device does not have the required IO capabilities, pairing/bonding will fail and connections to the remote device *cannot* be made. See [SC] for further details.

Parameters

type — Signal identity CSR_BT_HFG_SECURITY_IN/OUT_REQ/CFM.

appHandle — Application handle to which the confirm message is sent.

secLevel — The application must specify one of the following values:

- CSR_BT_SEC_DEFAULT : Use default security settings
- CSR_BT_SEC_MANDATORY : Use mandatory security settings
- CSR_BT_SEC_SPECIFY : Specify new security settings

If CSR_BT_SEC_SPECIFY is set the following values can be OR'ed additionally:

- CSR_BT_SEC_AUTHORISATION: Require authorisation
- CSR_BT_SEC_AUTHENTICATION: Require authentication
- CSR_BT_SEC_ SEC_ENCRYPTION: Require encryption (implies

authentication)

- CSR_BT_SEC_MITM: Require MITM protection (implies encryption)

resultCode	The result code of the operation. Possible values depend on the value of resultSupplier. If e.g. the resultSupplier == CSR_BT_SUPPLIER_CM then the possible result codes can be found in csr_bt_cm_prim.h. All values which are currently not specified in the respective prim.h file are regarded as reserved and the application should consider them as errors.
resultSupplier	This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h

4.11 CSR_BT_HFG_CONFIG_SNIFF

Parameters / Primitives	type	mask
CSR_BT_HFG_CONFIG_SNIFF_REQ	✓	✓

Table 16: The CSR_BT_HFG_CONFIG_SNIFF Primitives

Description

Note: This signal is intended for advanced low-power mode control of the HFG, and is *not* required for normal operation. The HFG will automatically use low-power modes without any user intervention.

It has very advanced setups and it may be necessary to control the low-power modes in great detail. This signal primitive allows enabling/disabling the use of park and sniff-modes. Please note that there is no confirm signal for this request.

Parameters

type — Signal identity CSR_BT_HFG_CONFIG_SNIFF_REQ.

mask — A bitmask used for controlling the advanced low power modes. The bitmask is constructed by OR'ing together any of the following defines:

- CSR_BT_HFG_PWR_DISCONNECT_ON_NO_LP
  Disconnect the SLC if the HFG is unable to enter sniff-mode after 3 attempts
- CSR_BT_HFG_PWR_DISABLE_PARK_ACP
  Do not accept park mode requests from remote device
- CSR_BT_HFG_PWR_DISABLE_SNIFF_REQ
  Do not request sniff mode
- CSR_BT_HFG_PWR_DISABLE_SNIFF_ACP
  Do not accept sniff mode requests from remote device

4.12 CSR_BT_HFG_CONFIG_AUDIO

Primitives \ Parameters	type	connectionId	audioType	*audioSettingLen	audioSetting
CSR_BT_HFG_CONFIG_AUDIO_REQ	✓	✓	✓	✓	✓

Table 17: The CSR_BT_HFG_CONFIG_AUDIO Primitive

Description

Note: This signal is intended for advanced audio (eSCO and SCO) control of the HFG, and is *not* required for normal operation. The HFG will automatically use the best possible audio configuration with emphasis on latency and power usage – the default eSCO parameters are defined in the *csr_bt_usr_config.h* header file.

If special (e)SCO parameters are required, a set of user-defined settings can be sent to the HFG using this signal primitive. The HFG will then, when initiating the next (e)SCO try with the user-supplied setup. If the remote side rejects the setup, the Hands-Free Specification fallback settings are tried in order (S3, S2, S1 and SCO).

The *audioType*, *audioSetting* and *audioSettingLen* concept is shared with the CSR_BT_HFG_STATUS_AUDIO_IND signal in section 4.14.

Parameters

type — Signal identity CSR_BT_HFG_CONFIG_AUDIO_REQ.

connectionId — The connection index for which the audio setup should be changed.

audioType — This parameter designates what *datatype* the *audioSetting* points to. The allowed values are:

- CSR_BT_HFG_AUDIO_RETRANSMISSION
  Pointer type shall be of type *CsrBtHfgAudioRetransmission* and point to the new *retranmission effort* value for an eSCO link.
- CSR_BT_HFG_AUDIO_MAX_LATENCY
  Pointer type shall be of type *CsrBtHfgAudioMaxLatency* and point to the highest allowed latency value for an eSCO or SCO link.
- CSR_BT_HFG_AUDIO_SUP_PACKETS
  Pointer type shall be of type *CsrBtHfgAudioSupPackets* and point to a bitmask that identifies what eSCO and SCO packet types are allowed. This setting is also known as the *audio quality*.
- CSR_BT_HFG_AUDIO_TX_BANDWIDTH
  Pointer type shall be of type *CsrBtHfgAudioTxBandwidth* and point to the transmit bandwidth value.
- CSR_BT_HFG_AUDIO_RX_BANDWIDTH
  Pointer type shall be of type *CsrBtHfgAudioRxBandwidth* and point to the receive bandwidth value.
- CSR_BT_HFG_AUDIO_CODEC_SETUP
  Pointer type shall be of type *CsrBtHf*gAudioCodecType and point to the codec value preferred among those allowed. At connection time, the HFG will negotiate

this setting with the remote device if it also supports it..

- CSR_BT_HFG_AUDIO_SAMPLE_RATE_SETUP
  Pointer type shall be of type CsrBtHfgAudioSampleRateType and point to the sample rate value preferred among those allowed.

- CSR_BT_HFG_AUDIO_DSPM_STREAMS_CONFIG
  Pointer type shall be of type CsrBtHfgAudioDspmStreamConfig. With this configuration type, the application can decide where the audio shall be routed to and configure the audio. This is only relevant if the chip uses DSP. Refer to the chip app note for details. This configuration is only handled in the HFG code if the “CSR_USE_DSPM” define is enabled.

- CSR_BT_HFG_AUDIO_DSPM_STREAM_SINGLE_CONFIG
  Pointer type shall be of type CsrBtHfgAudioDspStreamSingleConfig. The same restrictions as for the “CSR_BT_HFG_AUDIO_DSPM_STREAMS_CONFIG” apply. This is used to configure one single feature AFTER the audio connection is established. For example, the audio gain. To find out what features can be set refer to the chip documentation.

audioSettingLen	16-bit value with the length of the pointer in bytes, which must also be in accordance with *audioType*.
*audioSetting	Pointer to the audio setting which is to be setup. The pointer must point to the type specified by the *audioType* signal. The HFG will free this pointer after is has been read.

The CSR_BT_HFG HFG_AUDIO_CODEC_SETUP and the CSR_BT_HFG_AUDIO_SAMPLE_RATE_SETUP settings may be passed from the application to the HFG at any time after activation of the HFG. However, these settings will only take effect if the remote device connecting to has support for Auristream too. If the desired settings cannot be agreed upon at audio connection setup, or if the remote device does not support Auristream, an audio connection will be tried established following the HFP 1.5 specification. After audio connection establishment, the primitive CSR_BT_HFG HFG_AUDIO_SETTINGS_IND will be sent to the application right after the CSR_BT_HFG HFG_AUDIO_IND or CSR_BT_HFG HFG_AUDIO_EXTENDED_IND message, in order to inform the application about the codec and sample rate actually used for the connection established.
If the CSR_BT_HFG HFG_AUDIO_CODEC_SETUP or the CSR_BT_HFG HFG_AUDIO_SAMPLE_SETUP are delivered to the HFG with connection Id field set to 0xFF, then the settings given will be set and used for all connections. The CSR_BT_HFG_AUDIO_DSPM_STREAMS_CONFIG and CSR_BT_HFG_AUDIO_DSPM_STREAM_SINGLE_CONFIG are relevant if the Wide-Band Speech feature is supported.

Supported packet types

When setting up the eSCO and SCO supported packet type, the packet type support for negotiating the audio quality is specified in a 2 octet bit pattern as shown below.

						3EV5	2EV5	3EV3	2EV3	EV5	EV4	EV3	HV3	HV2	HV1

Note that the EDR eSCO packets (2EV3, 3EV3, 2EV5 and 3EV5) have their enable bits inverted. The grayed out section are bits reserved for future use and their setting is irrelevant for the application interfacing the hands-free gateway profile manager. In the examples below the reserved bits are set to 1.

Enable all:

Should all packets be available while negotiating the quality setting for the audio link between devices, the HV bits and EV bits should be set. The fact that the EDR eSCO packets have their enable bits inverted means that they should not be set to enable them.

Set HV1 + HV2 + HV3 + EV3 + EV4 + EV5 and not 2EV3 + 3EV3 + 2EV5 + 3EV5

1111110000111111 = 0xFC3F

For setting this bit pattern, the following packet definitions from csr_bt_hci.h should be combined:

(HCI_ESCO_PKT_HV1 | HCI_ESCO_PKT_HV2 | HCI_ESCO_PKT_HV3 |
HCI_ESCO_PKT_EV3 | HCI_ESCO_PKT_EV4 | HCI_ESCO_PKT_EV5)

Specifying only SCO:

Should only SCO packets be available, the HV and the EDR eSCO bits should be set.

Set HV1 + HV2 + HV3 and 2EV3 + 3EV3 + 2EV5 + 3EV5

1111111111000111 = 0xFFC7

For setting this bit pattern, the following packet definitions from csr_bt_hci.h should be combined:

(HCI_ESCO_PKT_HV1 | HCI_ESCO_PKT_HV2 |
HCI_ESCO_PKT_HV3 | HCI_ALL_MR_ESCO)

Specifying only eSCO:

Should only the eSCO packet types be available, the EV bits should be set and the EDR eSCO bits should not.

Set EV3 + EV4 + EV5 and not 2EV3 + 3EV3 + 2EV5 + 3EV5

1111110000111000 = 0xFC38

For setting this bit pattern, the following packet definitions from csr_bt_hci.h should be combined:

(HCI_ESCO_PKT_HV1 | HCI_ESCO_PKT_HV2 | HCI_ESCO_PKT_HV3)

4.13 CSR_BT_HFG_STATUS_LP

Primitives \ Parameters	type	connectionId	currentMode	wantedMode	oldMode	remoteReq	resultCode	resultSupplier
CSR_BT_HFG_STATUS_LP_IND	✓	✓	✓	✓	✓	✓	✓	✓

Table 18: The CSR_BT_HFG_STATUS_LP Primitive

Description

This signal will only be sent to the application if the CSR_BT_HFG_CNF_LP_STATUS bit has been set in the *hfgConfig* for the CSR_BT_HFG_ACTIVATE_REQ, see section 4.2. The signal should only be used for debugging, and it cannot be used for low-power control of any kind.

When CSR_BT_HFG_CNF_LP_STATUS is enabled, the application must *always* be able to receive and process a CSR_BT_HFG_STATUS_LP_IND signal, no matter what state the application, profile or connection is in.

The signal is sent every time the HFG detects a low power mode change on any of the active connections. The *mode* parameters are defined in the *csr_bt_cm_prim.h* header file.

Parameters

type	Signal identity CSR_BT_HFG_STATUS_LP_IND.
connectionId	The connection index for which a LP mode change has been detected.
currentMode	The LP mode which has just been entered.
oldMode	The LP mode which has just been left.
wantedMode	The LP mode which the HFG would like to enter.
remoteReq	This boolean parameter indicates whether the LP mode transition was requested by the remote device (TRUE), or it was initiated locally (FALSE).
resultCode	The result code of the operation. Possible values depend on the value of resultSupplier. If e.g. the resultSupplier == CSR_BT_SUPPLIER_CM then the possible result codes can be found in csr_bt_cm_prim.h. All values which are currently not specified in the respective prim.h file are regarded as reserved and the application should consider them as errors.
resultSupplier	This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h

4.14 CSR_BT_HFG_STATUS_AUDIO_IND

Primitives \ Parameters	type	connectionId	audioType	*audioSettingLen	audioSetting
CSR_BT_HFG_STATUS_AUDIO_IND	✓	✓	✓	✓	✓

Table 19: The CSR_BT_HFG_STATUS_AUDIO_IND Primitive

Description

This signal will only be sent to the application if the CSR_BT_HFG_CNF_AUDIO_STATUS bit has been set in the *hfgConfig* for the CSR_BT_HFG_ACTIVATE_REQ, see section 4.2. The signal should only be used for debugging, and it cannot be used for fine grained control of the (e)SCO setup.

When CSR_BT_HFG_CNF_AUDIO_STATUS is enabled, the application must *always* be able to receive and process a CSR_BT_HFG_STATUS_AUDIO_IND signal, no matter what state the application, profile or connection is in.

The signal is sent every time an incoming or outgoing eSCO or SCO connection is established or disconnected, and contains all available information about the details of the given (e)SCO connection. Some parameters may be left as 0 (zero) if the specific parameter is not available.

The signal is also sent to the application when the hands free device and the gateway agree on a specific audio coding scheme to use over the air.

The *audioType*, *audioSetting* and *audioSettingLen* concept is shared with the CSR_BT_HFG_CONFIG_AUDIO_IND signal in section 4.12.

Parameters

type — Signal identity CSR_BT_HFG_CONFIG_AUDIO_REQ.

connectionId — The connection index for which the audio setup should be changed.

audioType — This parameter designates what *datatype* the *audioSetting* points to. The allowed values are:

- CSR_BT_HFG_AUDIO_SCO_STATUS
  Pointer type shall be of type *CsrBtHfgAudioScoStatus* and point to the structure that describes (e)SCO parameters.

Warning: It required for the application to inspect the *audioType* parameter before trying to decode the *audioSetting* pointer. Unknown values of the *audioType* parameter shall mean that the application shall ignore, but still CsrPfree(), the *audioSetting* pointer.

audioSettingLen — 16-bit value with the length of the pointer in bytes, which must also be in accordance with *audioType*.

*audioSetting — Pointer to the audio status information. The pointer must point to the type specified by the *audioType* signal. The shall CsrPfree() this pointer after it has been used.

4.15 CSR_BT_HFG_RING

Primitives \ Parameters	type	connectionId	repetitionRate	numOfRings	*number	*name	numType
CSR_BT_HFG_RING_REQ	✓	✓	✓	✓	✓	✓	✓
CSR_BT_HFG_RING_CFM	✓	✓					

Table 20: The CSR_BT_HFG_RING Primitives

Description

Initiate ringing towards the HF or HS side. The HFG manager will ensure repetition of the ring signal for the specified number of times with the given interval.

If the remote device is a HF and it has enabled CLIP the HFG manager will automatically send the A-number if available, otherwise it will be ignored. The same is true for the calling party name, which is a special CSR2CSR feature, see section 4.33 for more information about how to enable the feature.

The CSR_BT_HFG_RING_CFM will only be sent to the application if the call is not answered or rejected while ringing.

Parameters

type — The signal identity, CSR_BT_HFG_RING_REQ/CFM.

connectionId — The connection index for which ringing should start or have just stopped

repetitionRate — Time interval, in seconds, between sending the ring signal to the HF side. Note that the first ring is sent immediately.

numOfRings — Number of rings to be sent, with a few exceptions:

- **0** will stop any ongoing ringing and send a CSR_BT_HFG_RING_CFM to the application
- **1-254** will send the specified number of rings, until either the ringing stops via a CSR_BT_HFG_RING_CFM, or the call is either answered (CSR_BT_HFG_ANSWER_IND) or rejected (CSR_BT_HFG_REJECT_IND).
- **255** will ring forever, or until the call is either answered, rejected or the ringing is stopped by sending a new CSR_BT_HFG_RING_REQ with a *numOfRings* of 0.

*number — A zero-terminated string containing the calling party's number in ASCII. The HFG will automatically add quotes around the string, if such are not already present.

*name — The name (e.g. phonebook name) of the calling party. This is a zero-terminated ASCII-string. The HFG will automatically add quotes around the string if they are not already present.

numType — The "number type" as specified by the Hands-Free Profile specification.

Library Function

```
void CsrBtHfgRingReqSend( CsrBtHfgConnectionId   connectionId,
                          CsrUint16      repetitionRate,
                          CsrUint8       numOfRings,
                          CsrCharString  *number,
                          CsrCharString  *name,
                          CsrUint8       numType   )
```

All parameters are as described above in the signal primitive details.

4.16 CSR_BT_HFG_ANSWER

Primitives \ Parameters	type	connectionId
CSR_BT_HFG_ANSWER_IND	✓	✓

Table 21: The CSR_BT_HFG_ANSWER Primitive

Description

This signal indicates that the HF has accepted the incoming call and has answered the call request.

Parameters

type	The signal identity, CSR_BT_HFG_ANSWER_IND.
connectionId	The identification number of the connection for which the answer is intended.

4.17 CSR_BT_HFG_REJECT

Primitives \ Parameters	type	connectionId
CSR_BT_HFG_REJECT_IND	✓	✓

Table 22: The CSR_BT_HFG_REJECT Primitive

Description

This signal notifies the application of the rejection of an incoming call or interruption of already ongoing call process. The service level connection is *not* affected by the reject. This signal can only be received if the connection is established to a HF device.

Parameters

type	The signal identity, CSR_BT_HFG_REJECT_IND.
connectionId	The identification number of the connection intended for rejection.

4.18 CSR_BT_HFG_CALL_WAITING

Primitives \ Parameters	type	connectionId	*number	*name	numType
CSR_BT_HFG_CALL_WAITING_REQ	✓	✓	✓	✓	✓

Table 23: The CSR_BT_HFG_CALL_WAITING primitive

Description

Call waiting is used for notifying the headset about incoming calls when there are currently active calls. It is safe for the application to send this message even if the headset does not support the call waiting notification – in this case, the HFG will simply ignore the message sent from the application.

Parameters

type	The signal identity, CSR_BT_HFG_CALL_WAITING_REQ.
connectionId	The connection index to which the 'call waiting' notification should be sent to.
*number	The zero-terminated ASCII-string containing the number of the calling party.
*name	The zero-terminated ASCII-string containing the name (e.g. phonebook name) of the calling party.
numType	The Hands-Free Profile 'number type' for the calling party number.

Library function

```
void CsrBtHfgCallWaitingReqSend(CsrBtHfgConnectionId   connectionId,
                                CsrCharString   *number,
                                CsrCharString   *name,
                                CsrUint8        numType   )
```

All parameters are as described above in the signal primitive details.

4.19 CSR_BT_HFG_CALL_HANDLING

Primitives \ Parameters	type	connectionId	btrh	cmeeCode	value	index
CSR_BT_HFG_CALL_HANDLING_REQ	✓	✓	✓			
CSR_BT_HFG_CALL_HANDLING_IND	✓	✓			✓	✓
CSR_BT_HFG_CALL_HANDLING_RES	✓	✓	✓	✓		

Table 24: The CSR_BT_HFG_CALL_HANDLING primitives

Description

The call handling signals are used for controlling the three way calling and response-and-hold features in the Hands-Free Profile.

The CSR_BT_HFG_CALL_HANDLING_REQ is used solely for sending response-and-hold (BTRH) commands to the headset. The CSR_BT_HFG_CALL_HANDLING_IND is used for notifying the application about three-way-calling and response-and-hold requests made by the headset. CSR_BT_HFG_CALL_HANDLING_RES is used for answering these requests.

The most important aspect in dealing with three-way-calling commands is to decode the *value* correctly. In the header file *csr_bt_hf.h*, the different values can be found. Depending on what *value* is sent, the *index* can be used for targeting specific call indexes, as is the case with e.g. CHLD=1X and CHLD=2X.

Note: The *btrh* parameter must be set to the special value *CSR_BT_HFG_BTRH_IGNORE* unless the command or response is used for BTRH operations. Failure to do so may leave the HFG and/or HF in an undefined state. Likewise, the *cmeeCode* must be set to the special CSR_BT_CME_SUCCESS value in order for an operation to succeed.

Parameters

type — The signal identity, CSR_BT_HFG_CALL_HANDLING_REQ/IND/RES.

connectionId — The HFG connection for which the call handling is taking place

btrh — This value is only used when requesting or responding to BTRH (response and hold) commands. The possible values are defined in *csr_bt_hfg_prim.h* and are:

- CSR_BT_HFG_BTRH_INCOMING_ON_HOLD
  Put incoming call on hold. Also used in the response to indicate that at least one call is on hold.
- CSR_BT_HFG_BTRH_ACCEPT_INCOMING
  Accept incoming call command.
- CSR_BT_HFG_BTRH_REJECT_INCOMING
  Reject incoming call command.
- CSR_BT_HFG_BTRH_IGNORE
  This value shall be used when *not* replying to BTRH commands, or in the response when no calls are on hold.

cmeeCode	The extended error code (CME) response. The header file csr_bt_hf.h defines all available CME error codes. In case of success, the special *CSR_BT_CME_SUCCESS* must be used. **Note:** If extended error codes have not been enabled by the HF, the HFG will automatically ignore this result and return the simple ERROR response.
value	The value denotes the three-way-calling or response-and-hold opcode. The header file *csr_bt_hf.h* defines the possible values as: • CSR_BT_RELEASE_ALL_HELD_CALL This covers the AT command AT+CHLD=0 • CSR_BT_RELEASE_ACTIVE_ACCEPT This covers the AT command AT+CHLD=1 • CSR_BT_RELEASE_SPECIFIED_CALL This covers the AT command AT+CHLD=1X • CSR_BT_HOLD_ACTIVE_ACCEPT This covers the AT command AT+CHLD=2 • CSR_BT_REQUEST_PRIVATE_WITH_SPECIFIED This covers the AT command AT+CHLD=2X • CSR_BT_ADD_CALL This covers the AT command AT+CHLD=3 • CSR_BT_CONNECT_TWO_CALLS This covers the AT command AT+CHLD=4 • CSR_BT_BTRH_PUT_ON_HOLD This covers the AT command AT+BTRH=0 • CSR_BT_BTRH_ACCEPT_INCOMING This covers the AT command AT+BTRH=1 • CSR_BT_BTRH_REJECT_INCOMING This covers the AT command AT+BTRH=2 • CSR_BT_BTRH_READ_STATUS This covers the AT query status command AT+BTRH?
index	The index is used for targeting a specific call index if the *value* parameter is either CSR_BT_RELEASE_SPECIFIED_CALL or CSR_BT_REQUEST_PRIVATE_WITH_SPECIFIED. Otherwise the *index* shall be ignored by the application.

Library function

```
void CsrBtHfgCallHandlingReqSend(CsrBtHfgConnectionId   connectionId,
                                 CsrBtHfgBtrhResponse   btrh      )


void CsrBtHfgCallHandlingResSend(CsrBtHfgConnectionId   connectionId,
                                 CsrUint16    cmeeCode,
                                 CsrBtHfgBtrhResponse   btrh      )
```

The parameters for both functions are as defined above in the signal primitive description.

4.20 CSR_BT_HFG_DIAL

Parameters / Primitives	type	connectionId	command	*number	cmeeCode
CSR_BT_HFG_DIAL_IND	✓	✓	✓	✓	
CSR_BT_HFG_DIAL_RES	✓	✓			✓

Table 25: The CSR_BT_HFG_DIAL primitives

Description

The CSR_BT_HFG_DIAL primitives are used for controlling dialing requests sent from the headset. This includes number dial, memory/speed dial and redial-last-dialed-number. The exact type of dial request is stored in the *command* parameter.

Parameters

type — The signal identity, CSR_BT_HFG_DIAL_IND/RES.

connectionId — The HFG connection index this call request or response is for/from.

command — The dial command issued by the headset. The possible values are defined in the header file csr_bt_hfg_prim.h and are

- CSR_BT_HFG_DIAL_NUMBER
  Number dial requested for the number stored in the *number* parameter. This covers the AT command “ATD”.
- CSR_BT_HFG_DIAL_MEMORY
  Memory or speed dial requested for the memory index stored in the *number* parameter. This command covers the AT command “ATD>”
- CSR_BT_HFG_DIAL_REDIAL
  The last number dialed by the gateway should be re-dialed. This command covers the AT command “AT+BLDN”.

*number — When using number of memory dialing, this parameter contains a null-terminated ASCII string of the number/index to be called. After use it shall be CsrPfree()’ed by the application.

cmeeCode — If the network is not able to perform dialing at this time, or the gateway is not in a state where calls are possible, the extended error code shall be returned in the response. If the call can be initiated the value shall be set to CSR_BT_CME_SUCCESS.

Note: If extended error codes (CME) have not been enabled by the headset, the HFG will automatically ignore this result and return the simple ERROR response.

Library function

```
void CsrBtHfgDialresSend(    CsrBtHfgConnectionId   connectionId,
                             CsrUint16    cmeeCode      )
```

All parameters are as described above in the signal primitive details.

4.21 CSR_BT_HFG_SPEAKER_GAIN

Primitives \ Parameters	type	connectionId	gain
CSR_BT_HFG_SPEAKER_GAIN_REQ	✓	✓	✓
CSR_BT_HFG_SPEAKER_GAIN_IND	✓	✓	✓

Table 26: The CSR_BT_HFG_SPEAKER_GAIN Primitives

Description

These signals are used for controlling the speaker gain from both the HFG and the HF.

Parameters

type	The signal identity, CSR_BT_HFG_SPEAKER_GAIN_REQ/IND.
connectionId	The identification number of the connection for which the change of speaker gain is intended or has occurred.
gain	Value for volume setting. The range must be in the interval 0 to 15 (both inclusive). Note that this is a real value not an ASCII-character.

Library Function

```
void CsrBtHfgSpeakerGainReqSend(CsrBtHfgConnectionIdconnectionId,
                                CsrUint8    gain      )
```

All parameters are as described above in the signal primitive details.

4.22 CSR_BT_HFG_MIC_GAIN

Primitives \ Parameters	type	connectionId	gain
CSR_BT_HFG_MIC_GAIN_REQ	✓	✓	✓
CSR_BT_HFG_MIC_GAIN_IND	✓	✓	✓

Table 27: The CSR_BT_HFG_MIC_GAIN Primitives

Description

These signals are used for controlling the microphone gain from both the HFG and the HF side.

Parameters

type	The signal identity, CSR_BT_HFG_MIC_GAIN_REQ/IND.
connectionId	The identification number of the connection for which a microphone gain change is intended or has occurred.
gain	Value for volume setting. The range must be in the interval 0 to 15 (both inclusive). Note that this is a real value, not an ASCII character.

Library Function

```
void CsrBtHfgMicGainReqSend(  CsrBtHfgConnectionId  connectionId,
                              CsrUint8    gain      )
```

All parameters are as described above in the signal primitive details.

4.23 CSR_BT_HFG_AT_CMD

Primitives \ Parameters	type	connectionId	*command	cmeeEnabled
CSR_BT_HFG_AT_CMD_IND	✓	✓	✓	✓
CSR_BT_HFG_AT_CMD_REQ	✓	✓	✓	

Table 28: The CSR_BT_HFG_AT_CMD Primitives

Description

If the AT parser mode in the CSR_BT_HFG_ACTIVATE_REQ (see section 4.2) has been set to either semi or transparent mode, some or all AT-commands sent from the HS/HF may be forwarded directly to the application in form of a CSR_BT_HFG_AT_CMD_IND signal. If the application needs to response to these messages, or need to send special AT-commands not covered by the HFG, the CSR_BT_HFG_AT_CMD_REQ is used.

The HFG will not process any of the data in the *command* parameter except add or remove tailing zero-terminators when sending/receiving data to/from the lower protocol layers.

Parameters

type	The signal identity, CSR_BT_HFG_AT_CMD_IND/REQ.
connectionId	The identification number of the connection for which the AT commands is intended or from which it originates.
command	The raw AT-command as a zero-terminated ASCII-string. If the data is received by the application, the data must be CsrPfree()'ed after use. When sending the request to the HFG, the HFG will automatically CsrPfree() the data after use.
cmeeEnabled	When the AT parser is set to the semi-mode, it may be necessary for the application to know whether or not extended error codes (CME) have been enabled. If CME is enabled, the value will be TRUE, otherwise FALSE. Note that the value will always be FALSE when in transparent mode and the application itself has to keep track of the CMEE mode.

Library Function

```
void CsrBtHfgAtCmdReqSend(CsrBtHfgConnectionId   connectionId, CsrCharString
                          *command)
```

All parameters are as described above in the signal primitive details.

4.24 CSR_BT_HFG_OPERATOR

Primitives \ Parameters	type	connectionId	mode	*operatorName	cmeeCode
CSR_BT_HFG_OPERATOR_IND	✓	✓			
CSR_BT_HFG_OPERATOR_RES	✓	✓	✓	✓	✓

Table 29: The CSR_BT_HFG_OPERATOR primitives

Description

The HF may request the network operator or service provider name of the HFG. When the HFG received the CSR_BT_HFG_OPERATOR_IND it must respond with the CSR_BT_HFG_OPERATOR_RES. If no network is available, it can return an empty name. The signals cover the AT-commands “AT+COPS” and “+COPS”.

Parameters

type	The signal identity, CSR_BT_HFG_OPERATOR_IND / RES.
connectionId	The HFG connection index for which the indication/response is for
mode	The current mode as specified by the Hands-Free Profile specification
*operatorName	The operator name as a zero-terminated ASCII string. If no name is available the NULL value shall be passed. The HFG will automatically CsrPfree() this data after usage.
cmeeCode	The extended error code (CME) response. The header file *csr_bt_hf.h* defines all available CME error codes. In case of success, the special *CSR_BT_CME_SUCCESS* must be used. **Note:** If extended error codes have not been enabled by the HF, the HFG will automatically ignore this result and return the simple ERROR response.

Library Function

```
void CsrBtHfgOperatorResSend (CsrBtHfgConnectionId  connectionId, CsrUint8
                              mode,
                              CsrCharString   *operatorName, CsrUint16
                              cmeeCode)
```

All parameters are as described above in the signal primitive details.

4.25 CSR_BT_HFG_CALL_LIST

Primitives \ Parameters	type	connectionId	final	idx	dir	stat	mode	mpy	*number	numType	cmeeCode
CSR_BT_HFG_CALL_LIST_IND	✓	✓									
CSR_BT_HFG_CALL_LIST_RES	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Table 30: The CSR_BT_HFG_CALL_LIST primitives

Description

These signals are used for transferring the current list of calls upon indication of the HF.

As more than a single call may be present, it is possible to send several response messages directly after one another. The *final* parameter is used for denoting that the given response signal is the last.

If the call list is empty, a response must still be sent, but the *number* parameter must be set to the NULL value and the *final* flag must be TRUE.

Parameters

type — The signal identity, CSR_BT_HFG_CALL_LIST_IND/RES.

final — This boolean value must be set to TRUE when sending the last call entry in the list. Otherwise, the value shall be FALSE.

idx — The call index, starting from 1.

dir — The direction of the call. Set to 0 for outgoing calls and 1 for incoming calls.

stat — The status of the call. This value can be one of the following:

- **0,** Active
- **1,** Held
- **2,** Dialing
- **3,** Alerting
- **4,** Incoming
- **5,** Waiting

mode — The mode of the call. Set to 0 for voice calls, 1 for data and 2 for FAX.

mpy — Multiparty calls. Set to 0 for non-multiparty and 1 for multiparty calls.

*number — The A-number of the calling party. Set to NULL to denote that the call list is empty. The format is a zero-terminated ASCII-string.

numType	The number type as specified by the Hands-Free Profile specification.
cmeeCode	The extended error code (CME) response. The header file *csr_bt_hf.h* defines all available CME error codes. In case of success, the special *CSR_BT_CME_SUCCESS* must be used. **Note:** If extended error codes have not been enabled by the HF, the HFG will automatically ignore this result and return the simple ERROR response.

Library Function

```
void CsrBtHfgCallListResSend (   CsrBtHfgConnectionId      connectionId,
                                 CsrBool              final,
                                 CsrUint8          idx,
                                 CsrUint8          dir,
                                 CsrUint8          stat,
                                 CsrUint8          mode,
                                 CsrUint8          mpy,
                                 CsrCharString     *number,
                                 CsrUint8          numType,        CsrUint16
                                    cmeeCode  )
```

All parameters are as described above in the signal primitive details.

4.26 CSR_BT_HFG_SUBSCRIBER_NUMBER

Primitives \ Parameters	type	connectionId	final	*number	numType	service	cmeeCode
CSR_BT_HFG_SUBSCRIBER_NUMBER_IND	✓	✓					
CSR_BT_HFG_SUBSCRIBER_NUMBER_RES	✓	✓	✓	✓	✓	✓	✓

Table 31: The CSR_BT_HFG_SUBSCRIBER_NUMBER primitives

Description

These signals are used for indicating and transferring the subscriber number to the HF upon request.

Note that multiple numbers can be transferred using the *final* parameter. This flag must be set to TRUE for the last signal, and must be FALSE otherwise.

Parameters

type	The signal identity, CSR_BT_HFG_SUBSCRIBER_NUMBER_IND / RES.
connectionId	The HFG connection index for which the subscriber number has be sent
final	This boolean value must be set to TRUE when sending the last call entry in the list. Otherwise, the value shall be FALSE.
*number	The zero-terminated ASCII-string containing the network subscriber number.
numType	The number type according to the Hands-Free Profile.
service	The service the particular subscriber number relates to. This can be either 4 for voice or 5 for FAX.
cmeeCode	The extended error code (CME) response. The header file csr_bt_hf.h defines all available CME error codes. In case of success, the special *CSR_BT_CME_SUCCESS* must be used. **Note:** If extended error codes have not been enabled by the HF, the HFG will automatically ignore this result and return the simple ERROR response.

Library Function

```
void CsrBtHfgSubscriberNumberResSend(CsrBtHfgConnectionId  connectionId,
                                     CsrBool final, CsrCharString *number,
                                     CsrUint8  numType,
                                     CsrUint8  service,  CsrUint16 cmeeCode)
```

All parameters are as described above in the signal primitive details.

4.27 CSR_BT_HFG_STATUS_INDICATOR_SET

Primitives \ Parameters	type	connectionId	indicator	value
CSR_BT_HFG_STATUS_INDICATOR_SET_REQ	✓	✓	✓	✓

Table 32: The CSR_BT_HFG_STATUS_INDICATOR_SET Primitive

Description

This signal is used for updating the indicator values. The signal can be sent directly after the HFG profile has been activated. It is not necessary to have any established service level connections as the indicator and value are cached in the HFG.

It is possible to send status indicator updates to multiple connections at the same time using the special CSR_BT_HFG_CONNECTION_ALL value for the *connectionId* parameter.

The HFG keeps track of the status indicators that have been sent to the HF devices and only send updates to the device if the value for the given indicator has actually changed. One exception to this is the *call held indicator*, which is always sent to the HF side.

Note that this signal covers both the AT-command “+CIND” and “+CIEV”.

Parameters

type — The signal identity, CSR_BT_HFG_STATUS_INDICATOR_SET_REQ.

connectionId — The HFG connection index for which the status indicator update is sent

indicator — The indicator values can be dynamically assigned, but the HFG implementation uses static numbers which ease the use considerably. The following defines are found in the header file *csr_bt_hf.h*:

- CSR_BT_SERVICE_INDICATOR
- CSR_BT_CALL_STATUS_INDICATOR
- CSR_BT_CALL_SETUP_STATUS_INDICATOR
- CSR_BT_CALL_HELD_INDICATOR
- CSR_BT_SIGNAL_STRENGTH_INDICATOR
- CSR_BT_ROAM_INDICATOR
- CSR_BT_BATTERY_CHARGE_INDICATOR
- CSR_BT_GATHERED_CALL_INDICATORS

The CSR_BT_GATHERED_CALL_INDICATORS has been added to allow sending information of several indicators in one single RFCOMM frame.

value

The value of the indicator is dependent upon the indicator being changed. The allowed values are defined in the [HFP], and the header file csr_bt_hf.h has compiler defines for all of them:

- CSR_BT_SERVICE_INDICATOR
  - CSR_BT_NO_SERVICE_VALUE
  - CSR_BT_SERVICE_PRESENT_VALUE
- CSR_BT_CALL_STATUS_INDICATOR
  - CSR_BT_NO_CALL_ACTIVE_VALUE
  - CSR_BT_CALL_ACTIVE_VALUE
- CSR_BT_CALL_SETUP_STATUS_INDICATOR
  - CSR_BT_NO_CALL_SETUP_VALUE
  - CSR_BT_INCOMING_CALL_SETUP_VALUE
  - CSR_BT_OUTGOING_CALL_SETUP_VALUE
  - CSR_BT_OUTGOING_REMOTE_ALERT_VALUE
- CSR_BT_CALL_HELD_INDICATOR
  - CSR_BT_NO_CALL_HELD_VALUE
  - CSR_BT_CALL_HELD_RETRIEVE_OTHER_CALL_VALUE
  - CSR_BT_CALL_HELD_NO_ACTIVE_CALL_VALUE
- CSR_BT_SIGNAL_STRENGTH_INDICATOR
  - CSR_BT_SIGNAL_STRENGTH_LEVEL_0 ... 5
- CSR_BT_ROAM_INDICATOR
  - CSR_BT_ROAM_OFF
  - CSR_BT_ROAM_ON
- CSR_BT_BATTERY_CHARGE_INDICATOR
  - CSR_BT_BATTERY_CHARGE_LEVEL_0 ... 5
- CSR_BT_GATHERED_CALL_INDICATORS
  - CSR_BT_NO_CALL_ACTIVE_NO_CALL_SETUP
  - CSR_BT_CALL_ACTIVE_NO_CALL_SETUP
  - CSR_BT_CALL_ACTIVE_CALL_SETUP

Library Function

```
void CsrBtHfgStatusIndicatorSetReqSend(CsrBt  HfgConnectionId_t   connectionId,
                                       CsrUint8     indicator,
                                       CsrUint8     value     )
```

All parameters are as described above in the signal primitive details.

4.28 CSR_BT_HFG_INBAND_RINGING_REQ

Primitives \ Parameters	type	connectionId	inband
CSR_BT_HFG_INBAND_RINGING_REQ	✓	✓	✓

Table 33: The CSR_BT_HFG_INBAND_RINGING primitive

Description

Usually “in-band ringing” is enabled per request of the HFG’s supported features (see the CSR_BT_HFG_ACTIVATE_REQ in section 4.2), but it is possible to enable/disable the feature online with this signal.

Parameters

type	The signal identity, CSR_BT_HFG_INBAND_RINGING_REQ.
connectionId	The HFG connection index to toggle inband ringing for
inband	Boolean value set to the new status of inband ringing, i.e. TRUE for on or FALSE for off.

Library Function

```
void CsrBtHfgInbandRingingReqSend(  CsrBtHfgConnectionId   connectionId,
                                    CsrBool         inband    )
```

All parameters are as described above in the signal primitive details.

4.29 CSR_BT_HFG_GENERATE_DTMF

Parameters / Primitives	type	connectionId	dtmf
CSR_BT_HFG_GENERATE_DTMF_IND	✓	✓	✓

Table 34: The CSR_BT_HFG_GENERATE_DTMF primitive

Description

The application will receive this message if the headset wants the HFG to generate a dialing tone, known as a DTMF. The tone itself is a single character covering the number 0-9 and the special characters #, * and +.

Parameters

type	The signal identity, CSR_BT_HFG_GENERATE_DTMF_IND.
connectionId	The HFG connection index that is requiring a tone generation
dtmf	The character for which a tone should be generated. The character is a single ASCII character in the range “0” to “9” and the two special characters “#” (hash) and “*” (asterisk).

4.30 CSR_BT_HFG_NOISE_ECHO

Primitives \ Parameters	type	connectionId	nrec
CSR_BT_HFG_NOISE_ECHO_IND	✓	✓	✓

Table 35: The CSR_BT_HFG_NOISE_ECHO primitive

Description

The noise reduction and echo cancellation feature is enabled in the supported features bitmask of the CSR_BT_HFG_ACTIVATE_REQ signal. However, it is possible for the handsfree device to toggle this feature on and off during an ongoing connection, which will show up in the application as a CSR_BT_HFG_NOICE_ECHO_IND signal. This signal is sent to the application whenever the HF wants to toggle the NR/EC function.

Parameters

type	The signal identity, CSR_BT_HFG_NOISE_ECHO_IND.
connectionId	The HFG connection index for which NR/EC should be toggled
nrec	The new state of the NR/EC function. TRUE means on, FALSE is off.

4.31 CSR_BT_HFG_BT_INPUT

Parameters / Primitives	type	connectionId	request	cmeeCode	*response
CSR_BT_HFG_BT_INPUT_IND	✓	✓	✓		
CSR_BT_HFG_BT_INPUT_RES	✓	✓		✓	✓

Table 36: The CSR_BT_HFG_BT_INPUT primitives

Description

A special AT-command known at “AT+BINP” and the response “+BINP” allows for complex forms of Bluetooth input. These signals are used for transferring these requests/responses to the application.

Parameters

type — The signal identity, CSR_BT_HFG_BT_INPUT_IND/RES.

connectionId — The HFG connection index for this indication / response.

request — The ‘data request code’. The only allowed values are currently:

- **1:** Phone number corresponding to the last voice tag recorded in the HF

cmeeCode — If the application does not know how to handle the given *request* code, it shall return the appropriate extended error code (CME). If the application was able to process the command successful and the *response* is meaningful, the special code of CSR_BT_CME_SUCCESS shall be returned.

*response — The response is a custom zero-terminated ASCII-string that depends on the *request* code. The HFG profile will free this pointer after use.

Library Function

```
void CsrBtHfgBtInputResSend( CsrBtHfgConnectionId   connectionId,
                             CsrUint16 cmeeCode, CsrCharString*response )
```

All parameters are as described above in the signal primitive details.

4.32 CSR_BT_HFG_VOICE_RECOG

Primitives \ Parameters	type	connectionId	bvra	cmeeCode
CSR_BT_HFG_VOICE_RECOG_REQ	✓	✓	✓	
CSR_BT_HFG_VOICE_RECOG_IND	✓	✓	✓	
CSR_BT_HFG_VOICE_RECOG_RES	✓	✓		✓

Table 37: The CSR_BT_HFG_VOICE_RECOG primitives

Description

If voice recognition has been enabled, a voice recognition sequence can be started and stopped by both the headset and the HFG. These signals are just to initiate or end such procedures, and to reply whether or not the voice recognition request from the HF could be started.

Note: These signals merely control the voice recognition activation, but *not* the audio connection. Please see the CSR_BT_HFG_AUDIO primitives in section 4.7, 4.8 and 4.9 for information on how to establish and close audio connections.

Parameters

type	The signal identity, CSR_BT_HFG_VOICE_RECOG_REQ/IND.
connectionId	The HFG connection index for which voice recognition should be started/stopped.
bvra	Boolean value that states the state of the voice recognition, i.e. TRUE for on and FALSE for off.
cmeeCode	If the application does not know how to handle the given *request* code, it shall return the appropriate extended error code (CME). If the application was able to process the command successful and the *response* is meaningful, the special code of CSR_BT_CME_SUCCESS shall be returned.

Library Function

```
void CsrBtHfgVoiceRecogReqSend( CsrBtHfgConnectionId      connectionId,
                                CsrBool             bvra      )

void CsrBtHfgVoiceRecogResSend( CsrBtHfgConnectionId      connectionId,
                                CsrUint16       cmeeCode  )
```

All parameters are as described above in the signal primitive details.

4.33 CSR_BT_HFG_C2C_SF

Primitives \ Parameters	type	connectionId	number	value	*indicators	indicatorsLength
CSR_BT_HFG_C2C_SF_REQ	✓	✓	✓	✓		
CSR_BT_HFG_C2C_SF_IND	✓	✓			✓	✓

Table 38: The CSR_BT_HFG_C2C_SF primitives

Description

This signal is used for two things. The CSR_BT_HFG_C2C_SF_IND signal reports what CSR2CSR features the headset supports and would like to enable. The request is used for enabling/disabling a particular CSR2CSR feature such that it will be used when the headset supports it.

Note that this signal can be sent as soon as the HFG profile has been activated and before any service level connections are present. In that case the value of the 'connectionId' field shall be set to 0xFF. The HFG will store the desired setting and automatically enable it if the application has previously requested a feature.

Note: In order for a feature to be usable both the HFG and the HF must enable it, so it may BE necessary for the application to keep track of the HF indicators. If the application sends CSR2CSR requests that have not been agreed on by both the HF and HFG, the HFG will ignore the messages.

Note: The ADPCM and sample rate negotiation feature is deprecated and will be removed from the implementation. It is recommended not to use it or indicate support for it to remote devices.

Parameters

type — Signal identity CSR_BT_HFG_C2C_SF_REQ/IND.

connectionId — The HFG connection index (0xFF meaning all connections)

number — The CSR2CSR indicator index which should be enabled/disabled. The following compiler defines can be found in the header file *csr_bt_hf.h*:

- CSR_BT_C2C_NAME_IND
  Index 1, transfer names with call waiting and rings (clip)
- CSR_BT_C2C_TXT_IND
  Index 2, support for unsolicited text
- CSR_BT_C2C_SMS_IND
  Index 3, support for SMS arrival notification and text transfer
- CSR_BT_C2C_BAT_IND
  Index 4, battery charge level notifications
- CSR_BT_C2C_PWR_IND
  Index 5, power status notifications
- CSR_BT_C2C_ADPCM_IND

Index 6, audio codec settings. Can be set to a bitmask built with the values:

- 0x00 Disabled (default)
- 0x01 CVSD
- 0x02 2-bit ADPCM
- 0x04 4-bit ADPCM

- CSR_BT_C2C_SAMPLE_RATE_IND
  Index 7, audio data sampling rate. Can be set to a bitmask built with the values:
  - 0x00 Disabled (default)
  - 0x01 8 KHz
  - 0x02 16 KHz

value	The value of the given indicator. A value of “0” (zero) means off while a “1” (one) is on. In the case of the CSR_BT_C2C_ADPCM_IND and CSR_BT_C2C_SAMPLE_RATE_IND, the values can be any of the above mentioned, or 0 if none is supported. Note that the value is transferred as a real value, *not* an ASCII character.
*indicators	When the application receives the supported/enabled features from the HF, they are packed into an array of characters. This is the pointer for that array, and must be CsrPfree()’ed after use by the application. The indicator indexes are the same as explained above in the *number* parameter. **Note:** It is critical that the application does not try to decode indexes that exceed the *indicatorsLength* or indicator indexes that are unknown for the application.
indicatorsLength	The length of the *indicators* array. There shall be made no assumptions on the length of the array, as new CSR2CSR features may be added in the future without warning.

Library Function

```
void CsrBtHfgC2cSfReqSend(   CsrBtHfgConnectionId      connectionId,
                             CsrUint8        number,
                             CsrUint8        value     )
```

All parameters are as described above in the signal primitive details.

4.34 CSR_BT_HFG_C2C_TXT

Primitives \ Parameters	type	connectionId	*text
CSR_BT_HFG_C2C_TXT_REQ	✓	✓	✓

Table 39: CSR_BT_HFG_C2C_TXT primitive

Description

The CSR_BT_HFG_C2C_TXT_REQ primitive is used for sending unsolicited text to the HF. The primitive will be ignored by the HFG manager if support for unsolicited text is not activated in the HFG or if the HF does not support it.

Parameters

type	Signal identity CSR_BT_HFG_C2C_TXT_REQ.
connectionId	Identification of the connection from which the indication originates.
*text	Pointer to a zero-terminated ASCII-string. The HFG will CsrPfree() the pointer after use. The HFG will automatically add beginning and trailing quotes to the string, with one exception: **Warning:** If the string body itself contains special characters, including quotes, the application must escape these and *manually* add the heading and trailing quotes for the string. In this case, the HFG will just pass the raw string to the AT-command.

Library Function

```
void CsrBtHfgC2cTxtReqSend(  CsrBtHfgConnectionId   connectionId,
                             CsrCharString      *text     )
```

All parameters are as described above in the signal primitive details.

4.35 CSR_BT_HFG_C2C_SMS_ARRIVE

Primitives \ Parameters	type	connectionId	smsIndex	*number	*name
CSR_BT_HFG_C2C_SMS_ARRIVE_REQ	✓	✓	✓	✓	✓

Table 40: CSR_BT_HFG_C2C_SMS_ARRIVE primitive

Description

When a SMS has been received at the handset, a notification can be sent to a CSR2CSR capable HF using the CSR_BT_HFG_C2C_SMS_ARRIVE_REQ primitive.

Parameters

type	The signal identity, CSR_BT_HFG_C2C_SMS_ARRIVE_REQ.
connectionId	Identification of the connection from which the indication originates.
smsIndex	Unique identifier for the specific SMS-message.
*number	The number the SMS message originates from, encoded as a zero-terminated ASCII-string.
*name	The name of the person the SMS originates from, encoded as a zero-terminated ASCII-string.

Library Function

```
void CsrBtHfgC2cIncSmsReqSend(   CsrBtHfgConnectionId   connectionId,
                                 CsrUint16       smsIndex,
                                 CsrUint8     *number,
                                 CsrUint8     *name);
```

All parameters are as described above in the signal primitive details.

4.36 CSR_BT_HFG_C2C_SMS_GET

Primitives \ Parameters	type	connectionId	smsIndex
CSR_BT_HFG_C2C_SMS_GET_IND	✓	✓	✓

Table 41: The CSR_BT_HFG_C2C_SMS_GET primitive

Description

The CSR_BT_HFG_C2C_SMS_GET_IND is sent to the application when a connected CSR2CSR capable HF device requests a SMS message.

Parameters

type	Signal identity CSR_BT_HFG_C2C_SMS_GET_IND.
connectionId	Identification of the connection from which the indication originates.
smsIndex	Unique identifier for the specific SMS-message. The index corresponds to an index previously sent to the HF using a CSR_BT_HFG_C2C_SMS_ARRIVE_REQ primitive.

4.37 CSR_BT_HFG_C2C_SMS_TXT

Primitives \ Parameters	type	connectionId	*smsText
CSR_BT_HFG_C2C_SMS_TXT_REQ	✓	✓	✓

Table 42: The CSR_BT_HFG_C2C_SMS_TXT primitive

Description

The CSR_BT_HFG_C2C_SEND_SMS_REQ is used for sending a complete SMS message to the HF. This primitive should only be used after the reception of a CSR_BT_HFG_C2C_SMS_GET_IND.

Parameters

type	Signal identity CSR_BT_HFG_C2C_SEND_SMS_REQ.
connectionId	Identification of the connection from which the indication originates.
*smsText	Pointer to the message, encoded as a zero-terminated ASCII-string. The HFG will automatically add beginning and trailing quotes to the string, with one exception: **Warning:** If the string body itself contains special characters, including quotes, the application must escape these and *manually* add the heading and trailing quotes for the string. In this case, the HFG will just pass the raw string to the AT-command.

Library Function

```
void CsrBtHfgC2cSmsTxtReqSend(  CsrBtHfgConnectionId   connectionId,
                                CsrCharString      *smsText     )
```

All parameters are as described above in the signal primitive details.

4.38 CSR_BT_HFG_C2C_BATTERY

Parameters / Primitives	type	connectionId	batteryLevel
CSR_BT_HFG_C2C_BATTERY_IND	✓	✓	✓

Table 43: CSR_BT_HFG_C2C_BATTERY primitive

Description

If support for battery level is activated in the HFG and supported by the HF a CSR_BT_HFG_C2C_BATTERY_IND primitive will be sent to the application each time the battery level of a connected CSR2CSR capable HF changes.

Parameters

type	Signal identity CSR_BT_HFG_C2C_BATTERY_IND.
connectionId	Identification of the connection from which the indication originates.
batteryLevel	Battery level of the HF. Range from 0 to 9.

4.39 CSR_BT_HFG_C2C_POWER

Primitives \ Parameters	type	connectionId	powerStatus
CSR_BT_HFG_C2C_POWER_IND	✓	✓	✓

Table 44: CSR_BT_HFG_C2C_POWER primitive

Description

If support for power status is activated in the HFG and supported by the HF a CSR_BT_HFG_C2C_POWER_IND primitive will be sent to the application each time the power status of a connected CSR2CSR capable HF changes.

Parameters

Type	Signal identity CSR_BT_HFG_C2C_POWER_IND.
connectionId	Identification of the connection from which the indication originates.
powerStatus	Power status of the connected HF. 1: Powered by battery, 2: Externally powered.

4.40 CSR_BT_HFG_MANUAL_INDICATOR

Primitives \ Parameters	type	connectionId	*indicators	indicatorsLength
CSR_BT_HFG_MANUAL_INDICATOR_IND	✓	✓		
CSR_BT_HFG_MANUAL_INDICATOR_RES	✓	✓	✓	✓

Table 45: The CSR_BT_HFG_MANUAL_INDICATOR Primitives

Description

Note: This signal is intended for legacy applications that have been designed such that they are only able to send status indicator updates to the HFG profile when a connection is present. The HFG has been designed such that this old scheme is no longer necessary.

It is highly recommended **not** to use this signal, and in order to use this scheme it must be enabled via the CSR_BT_HFG_CNF_MANUAL_INDICATOR flag in the CSR_BT_HFG_ACTIVATE_REQ signal.

If enabled, the HFG will halt the AT startup sequence when the HF has requested the full status indicator settings via AT+CIND?. The HFG then sends the CSR_BT_HFG_MANUAL_INDICATOR_IND to the application, which must reply *immediately* using the CSR_BT_HFG_MANUAL_INDICATOR_RES with the full set of status indicators. The AT sequence is then continued as usual.

Parameters

Type	Signal identity CSR_BT_HFG_MANUAL_INDICATOR_RES.
connectionId	The connection index for which the audio setup should be changed.
*indicators	Pointer to an array of the status indicators as defined in the *csr_bt_hf.h*. It is up the the application to allocate this structure.
indicatorsLength	Length of the *indicators* pointer in bytes.

Library function

In order to ease the use of the response signal, the following library function exists, which will automatically allocate the *indicators* array and fill in the values are the correct places:

```
void CsrBtHfgManualIndicatorResSend( CsrBtHfgConnectionId   connectionId,
                                     CsrUint8        service,
                                     CsrUint8        callStatus,
                                     CsrUint8        callSetup,
                                     CsrUint8        callHeld,
                                        CsrUint8        signalStrength,
                                     CsrUint8        roam,
                                     CsrUint8        battery   );
```

4.41 CSR_BT_HFG_CONFIG_SINGLE_ATCMD

Primitives \ Parameters	type	pHandle	idx	sendToApp	result
CSR_BT_HFG_CONFIG_SINGLE_ATCMD_REQ	✓	✓	✓	✓	
CSR_BT_HFG_CONFIG_SINGLE_ATCMD_CFM	✓				✓

Table 46: CSR_BT_HFG_CONFIG_SINGLE_ATCMD primitive

Description

When the HFG is activated and the mode "*CSR_BT_HFG_AT_MODE_USER_CONFIG*" (see 4.2 for details) has been chosen, it is possible for the application to select single AT commands that it chooses to handle itself. It can do so by sending the CSR_BT_HFG_CONFIG_SINGLE_ATCMD_REQ message to HFG. The HFG will send a CSR_BT_HFG_CONFIG_SINGLE_ATCMD_CFM primitive as answer with a proper result code.

Parameters

type — The signal identity, CSR_BT_HFG_CONFIG_SINGLE_ATCMD_REQ

pHandle — The identity of the calling process, to which all indications shall be sent.

Idx — Index of the AT-command selected (see **3.4.12**)

sendToApp — Boolean to indicate whether the command shall be handled at HFG (1) or forwarded to the application (0)

result — Result of the operation. Possible result codes are:

- CSR_BT_HFG_SUCCESS
  The operation was successfully performed
- CSR_BT_HFG_OUT_OF_BOUNDS
  The index given does not map to an actual AT-command in the table
- CSR_BT_HFG_WRONG_ATMODE
  The HFG is not in CSR_BT_HFG_AT_MODE_USER_CONFIRM.
- CSR_BT_HFG_UNKNOWN_ERROR
  Operation failed due to other errors than the two described above

Library Function

```
void CsrBtHfgConfigSingleAtcmdReqSend(CsrSchedQid  phandle,
                                      CsrUint8 idx,
                                      CsrBool sendToApp);
```

All parameters are as described above in the signal primitive details.

Examples:

To let the application handle the AT command “AT+VGS=”, the following function call must be issued:

`CsrBtHfgConfigSingleAtcmdReqSend` (phandle, 2, 0);

If the HFG shall handle “AT+BINP=” :

`CsrBtHfgConfigSingleAtcmdReqSend` (phandle, 33, 1);

4.42 CSR_BT_HFG_CONFIG_ATCMD_HANDLING

Primitives \ Parameters	type	pHandle	bitwiseIndicators	result
CSR_BT_HFG_CONFIG_ATCMD_HANDLING _REQ	✓	✓	✓	
CSR_BT_HFG_CONFIG_ATCMD_HANDLING _CFM	✓		✓	✓

Table 47: CSR_BT_HFG_CONFIG_ ATCMD_HANDLING primitive

Description

When the HFG is activated and the mode "*CSR_BT_HFG_AT_MODE_USER_CONFIG*" (see 4.2 for details) has been chosen, the application can choose to handle some (or all) AT commands itself. It can do so by sending the CSR_BT_HFG_CONFIG_ATCMD_HANDLING_REQ message to HFG. The HFG will send a CSR_BT_HFG_CONFIG_ATCMD_HANDLING_CFM primitive as answer with a proper result code and the actual settings.

Parameters

type	The signal identity, CSR_BT_HFG_CONFIG_SINGLE_ATCMD_REQ
pHandle	The identity of the calling process, to which all indications shall be sent.
idx	Index of the AT-command selected
bitwiseIndicators	Pointer to an array of bytes where each bit represents one AT command. If a bit has value 0 the corresponding AT-command will be handled by the application.
result	Result of the operation. Possible result codes are: ○ CSR_BT_HFG_SUCCESS The operation was successfully performed ○ CSR_BT_HFG_OUT_OF_BOUNDS The index given does not map to an actual AT-command in the table ○ CSR_BT_HFG_WRONG_ATMODE The HFG is not in HFG_AT_MODE_USER_CONFIRM ○ CSR_BT_HFG_UNKNOWN_ERROR Operation failed due to other errors than the two described above

Library Function

```
void CsrBtHfgConfigAtcmdHandlingReqSend (CsrSchedQid  phandle,
                                         CsrUint8 *bitwiseIndicators);
```

All parameters are as described above in the signal primitive details.

4.43 CSR_BT_HFG_GET_C2C_ADPCM_LOCAL_SUPPORTED

Primitives \ Parameters	type	result
CSR_BT_HFG_GET_C2C_ADPCM_LOCAL_SUPPORTED_REQ	✓	
CSR_BT_HFG_GET_C2C_ADPCM_LOCAL_SUPPORTED_IND	✓	✓

Table 48: CSR_BT_HFG_GET_C2C_ADPCM_LOCAL_SUPPORTED Primitive

Description

Note: This primitive is deprecated and will be removed from the implementation. It is recommended not to use it.

The HFG application can find out whether the HW/FW platform supports ADPCM audio by means of this primitive. This operation should be performed prior to trying to change the codec settings.

Parameters

type	Signal identity CSR_BT_HFG_GET_C2C_ADPCM_LOCAL_SUPPORTED_REQ/IND
result	TRUE if ADPCM audio is supported; FALSE otherwise

Library Function

```
void CsrBtHfgGetC2CAdpcmLocalSupportedReqSend(void)
```

4.44 CSR_BT_HFG_DEREGISTER_TIME

Primitives \ Parameters	type	waitSeconds	result
CSR_BT_HFG_DEREGISTER_TIME_REQ	✓	✓	
CSR_BT_HFG_DEREGISTER_TIME_CFM	✓		✓

Table 49: CSR_BT_HFG_DEREGISTER_TIME Primitive

Description

Whenever a connection is established from a remote device, the HFG profile will remove the service record used for that connection from the local service database. This is done to avoid other remote devices trying to connect to the local channel in that specific service record. However, some devices try to read some of the features stored in the service record even after the connection has been established. Therefore, the HFG application may want to keep the service record for some time after connection for interoperability purposes. The profile provides an interface to do that: the CSR_BT_HFG_DEREGISTER_TIME_REQ and CFM messages allow the application to determine the number of seconds that the profile must wait before removing the service record from the local service database. Per default, this time is 0, and the service record will be removed immediately after connection establishment.

BEWARE: As long as the service record exists in the service database, any other remote devices will be able to get information from it, and try to connect to the local device with the information in it. These attempts to connect will fail as long as there is an active connection on it!

Parameters

type	Signal identity CSR_BT_HFG_DEREGISTER_TIME_REQ /CFM
waitSeconds	Number of seconds to wait. If 0, the service record is removed immediately (default)
result	Result of the operation: CSR_BT_HFG_SUCCESS if the operation succeeds

Library Function

void CsrBtHfgSetDeregisterTimeReqSend (waitSeconds)

5 Document References

Document	Reference
Bluetooth® Core Specification v.1.1, v.1.2 and v.2.0	[BT]
CSR Synergy Bluetooth, SC – Security Controller API Description, Document no. api-0102-sc	[SC]
The Bluetooth® Specification, Hands-free Profile, ver. 1.5, 2005-08-01	[HF]
Specification of the Bluetooth System, Profiles part K:6, ver. 1.2 2008-Dec-18	[HS]
Bluetooth® Hands-Free Profile Application Guidelines	[CCAP]
CSR Synergy Bluetooth. CM – Connection Manager API Description, doc. no. api-0101-cm	[CM]
CSR Synergy Bluetooth, SD - Service Discovery API Description, doc. no. api-0103-sd	[SD]
Errata Service Release to Specification version 1.1, profile section K:6	[ESR]

Terms and Definitions

BlueCore®	Group term for CSR's range of Bluetooth wireless technology chips
Bluetooth®	Set of technologies providing audio and data transfer over short-range radio connections
CSR	Cambridge Silicon Radio
HFG	Hands-free Gateway Profile
UniFi™	Group term for CSR's range of chips designed to meet IEEE 802.11 standards

Document History

Revision	Date	History
1	26 SEP 11	Ready for release 18.2.0

TradeMarks, Patents and Licences

Unless otherwise stated, words and logos marked with ™ or ® are trademarks registered or owned by CSR plc or its affiliates. Bluetooth® and the Bluetooth logos are trademarks owned by Bluetooth SIG, Inc. and licensed to CSR. Other products, services and names used in this document may have been trademarked by their respective owners.

The publication of this information does not imply that any licence is granted under any patent or other rights owned by CSR plc.

CSR reserves the right to make technical changes to its products as part of its development programme.

While every care has been taken to ensure the accuracy of the contents of this document, CSR cannot accept responsibility for any errors.

Life Support Policy and Use in Safety-critical Compliance

CSR's products are not authorised for use in life-support or safety-critical applications. Use in such applications is done at the sole discretion of the customer. CSR will not warrant the use of its devices in such applications.

Performance and Conformance

Refer to www.csrsupport.com for compliance and conformance to standards information.