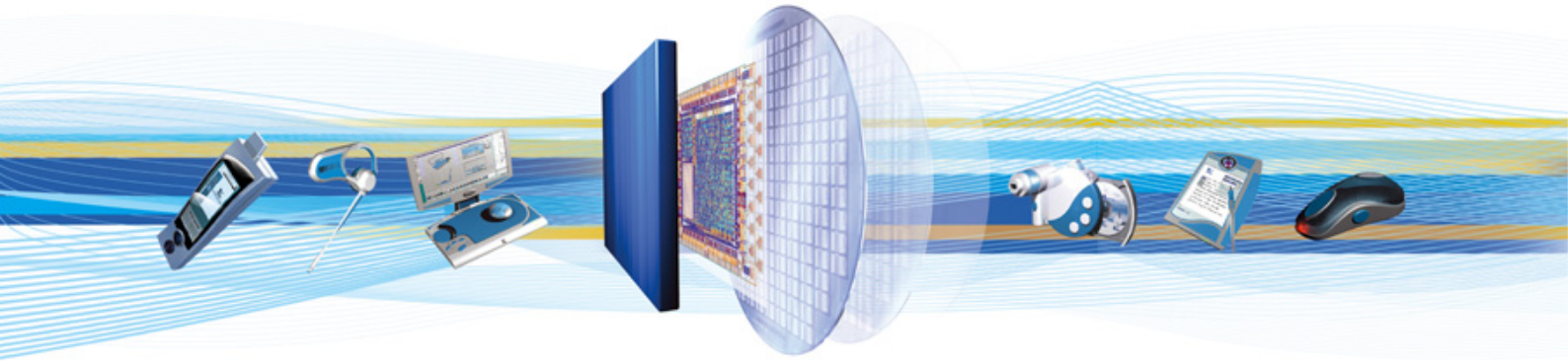
CSR Synergy Bluetooth 18.2.0

AVRCP - Audio Video Remote Control Profile

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 Audio Video Remote Control Profile component, henceforward called AVRCP. The AVRCP provides the services specified by the Advanced Audio Remote Control Profile revision 1.4 [AVRCP].

1.2 Assumptions

The following assumptions and preconditions are made:

- Only one instance of the AVRCP profile manager can be active at one time

2 Description

2.1 Introduction

The AVRCP provides the interface for an application that conforms to the Audio/Video Remote Control Profile. The AVRCP supports all mandatory requirements for the Audio/Video Control Transport Protocol Specification [AVCTP] and the Audio/Video Remote Control Profile [AVRCP].

The AVRCP supplies functionality for:

- Handling of service records
- Establishment and release of connections
- Exchange of pass-through commands and responses
- Exchange of metadata
- Support for browsing functionality
- AV/C and AVCTP message fragmentation and reassembly
- Automatic error response to incorrectly formatted AVRCP commands
- Connections to multiple remote devices simultaneously

The AVRCP provides support for both the controller side and the target side. However, some devices might only want to offer support for either the controller or the target. It is possible to remove the unsupported part of the AVRCP at compile time, saving code space. To remove the controller part, the code shall be compiled with the "EXCLUDE_AVRCP_CT_MODULE" flag define. Likewise, to remove the target part, the "EXCLUDE_AVRCP_TG_MODULE" shall be defined at compilation time.

2.2 Overall concept

The profile is designed for interfacing to multiple applications or tasks:

- One control application that handles initial configuration and connections
- One AVRCP controller applications for sending commands to a remote target (CT only)
- One or more media player applications for handling incoming commands from a controller (TG only)

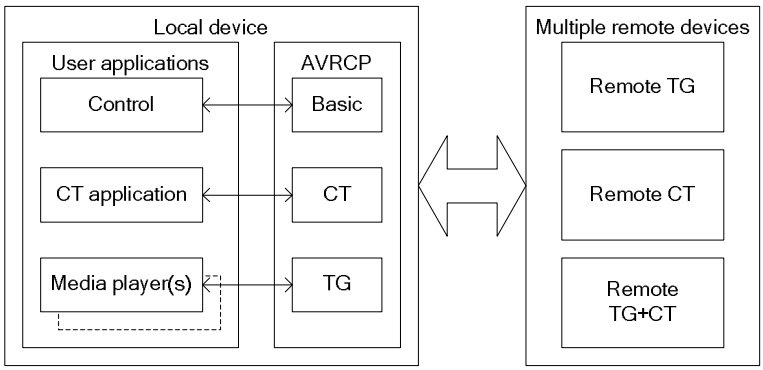


Figure 1: Overall concept

2.2.1 AVRCP Functionality Overview

Table 1 gives an overview of the functionality provided by the profile additional information:

Functionality	Version	Addressed	Browsed	Timer (AVRCP_TIMER_)
Configuration	-	-	-	-
Activation/deactivation	-	-	-	-
Connection establishment	-	-	-	-
Pass-through commands	1.0 (1.3 for GN)	X	-	RCP
Notifications	1.3 (1.4 for some)	X	-	MTP
Player Application Settings	1.3	X	-	MTP
InformDisplayableCharset	1.3	X	-	MTC
InformBatteryStatusofCT	1.3	X	-	MTC
GetAttributes	1.3 (elem)/1.4 (item)	X (elem)	X (item)	MTP
Volume	1.4	X	-	MTC
Set Addressed/Browsed Player	1.4	-	-	MTC
Get Play Status	1.3	X	-	MTP
Get Folder Items	1.4	-	X	BROWSING
Change Path	1.4	-	X	BROWSING
Play Item	1.4	X	-	BROWSING
Add To Now Playing List	1.4	-	X	BROWSING
Search	1.4	-	X	BROWSING_SEARCH

Table 1: List of AVRCP functionality

The version indicates the minimum AVRCP version that must be supported by the two roles to use the functionality. The Addressed and Browsed columns indicates whether a command will be sent to the addressed or the browsed media player. The final column specifies which timers are used with the different commands. The timers in the two tables can be modified in csr_bt_usr_config.h.

Specification defined timers	Value in microsecond	Description
CSR_BT_AVRCP_TIMER_RCP	100000	Response time for any pass through command
CSR_BT_AVRCP_TIMER_MTC	200000	Response time for vendor dependent CONTROL
CSR_BT_AVRCP_TIMER_MTP	1000000	Response time for vendor dependent INTERIM and STATUS
Other timers		
CSR_BT_AVRCP_CT_TIMER_AIR_OVERHEAD	1500000	Additional time between commands and responses. The CT will issue a timeout if a response is not received within CSR_BT_AVRCP_CT_TIMER_AIR_OVERHEAD + the specification timer. The timer can be changed in csr_bt_usr_config.h.
CSR_BT_AVRCP_TIMER_BROWSING	1000000	Response time for browsing commands
CSR_BT_AVRCP_TIMER_BROWSING_SEARCH	20000000	Response time for search command (browsing)

Table 1: List of timers in AVRCP

2.3 Terms

2.3.1 Scope

The term scope is used in relation to browsing functionality and defines which part of a media player a command is intended for. The available scopes are listed in Table 2:

Scope	Define	Description
Media Player List	CSR_BT_AVRCP_SCOPE_MP_LIST	This scope covers the complete list of media players available on a target
Virtual File System	CSR_BT_AVRCP_SCOPE_MP_FS	The virtual file system associated to the media player
Search Results	CSR_BT_AVRCP_SCOPE_SEARCH	This scope will only be available after a successfully completed search procedure and covers the media items in the search result
Now Playing List	CSR_BT_AVRCP_SCOPE_NPL	The media items in the Now Playing List
Currently playing Item	CSR_BT_AVRCP_SCOPE_PLAYING	Deprecated, will be removed in Bt CSR Synergy 18.0.0.
Invalid	CSR_BT_AVRCP_SCOPE_INVALID	-

Table 2: Scopes

2.3.2 UIDs

Each item (media or folder) in the Virtual File System, Search Result or Now Playing List scope must have a unique number called the UID. It is the responsibility of the user applications to define and handle the UIDs.

The UID counter makes it possible for a target and controller to stay synchronised with relation to the currently assigned UIDs for items. If the UID is changed for one or more items, the UID Counter must be increased by one by the target.

Note: All UID parameters in API functions and primitives as described by this document expects the UID to be given as an array of 8 bytes with least significant byte first, i.e. little endian format.

2.3.3 Media Players

The target device must provide one or more media players that the controller device can access. The media player(s) can be of different types: music player(s), video player(s), FM radio(s), mobile TV tuner(s), etc..

3 Interface description

3.1 Configuration

Before the AVRCP module can be used it must be configured, which involves registration of one or two service records depending on which roles to support. Configuration can be initiated by sending a CSR_BT_AVRCP_CONFIG_REQ message by means of the `CsrBtAvrcpConfigReqSend` or the `CsrBtAvrcpConfigReqExtSend` function to the profile. This is illustrated in Figure 2.

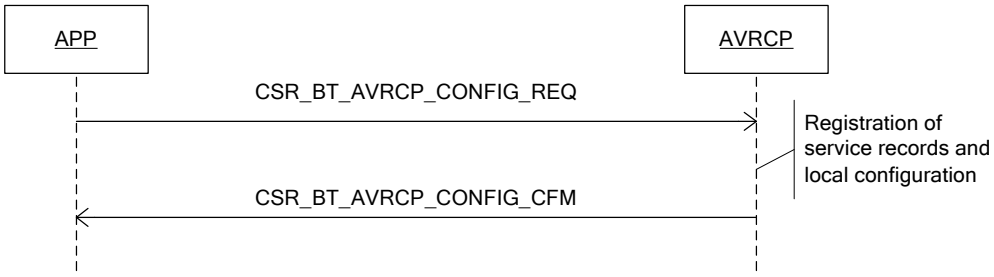


Figure 2: AVRCP configuration

The prototype for the `CsrBtAvrcpConfigReqSend` function is:

```
CsrBtAvrcpConfigReqSend(CsrSchedQid phandle, CsrUint32 globalConfig, CsrUint16
mtu, CsrBtAvrcpRoleDetails tgFeatures, CsrBtAvrcpRoleDetails ctFeatures)
```

The prototype for the `CsrBtAvrcpConfigReqExtSend` function is:

```
CsrBtAvrcpConfigReqExtSend(CsrSchedQid phandle, CsrUint32 globalConfig,CsrUint16
mtu, CsrBtAvrcpRoleDetails tgFeatures, CsrBtAvrcpRoleDetails ctFeatures,
CsrUint16 uidCount)
```

The `CsrBtAvrcpConfigReqExtSend` function is recommended for TG devices that are database aware. The TG profile keeps an updated value of the UID counter .Every time the TG application issued a command or a response containing the UID counter value, the local UID counter will be updated. Using the `CsrBtAvrcpConfigReqSend` function will set the UID counter to 0 to begin with at the TG side, while using the `CsrBtAvrcpConfigReqExtSend` function will set the local UID counter to the value given as parameter. This is irrelevant for CT devices, as they will always get the UID counter from the remote TG device.

The arguments for the primitive are described in Table 3.

Type	Argument	Description
CsrSchedQid	phandle	Queue ID of the control application
CsrUint32	globalConfig	Global configuration for AVRCP – set to CSR_BT_`AVRCP_CONFIG_GLOBAL_STANDARD` for now
CsrUint16	mtu	Maximum transmission unit to announce to a remote device during connection establishment (L2CAP configuration)
CsrBtAvrcpRoleDetails	tgFeatures	Features of the target – initialize parameter with `CsrBtAvrcpConfigRoleNoSupport()` or `CsrBtAvrcpConfigRoleSupport()`
CsrBtAvrcpRoleDetails	ctFeatures	Same as for tgFeatures
CsrUint16	uidCount	Start value of the “uid counter” . This is only relevant for TG devices.

Table 3: Arguments for the `CSR_BT_AVRCP_CONFIG_REQ_SEND` primitive

The two parameters `tgFeatures` and `ctFeatures` can be initialized by using one of the following helper functions depending of whether the specific role is supported or not:

```
void CsrBtAvrcpConfigRoleNoSupport(CsrBtAvrcpRoleDetails *details)

void CsrBtAvrcpConfigRoleSupport(CsrBtAvrcpRoleDetails *details,
   CsrBtAvrcpConfigRoleMask roleConfig, CsrBtAvrcpConfigSrVersion
   srAvrcpVersion, CsrBtAvrcpConfigSrFeatureMask srFeatures,
CsrCharString  *providerName, CsrCharString *serviceName)
```

The parameters for the helper functions listed in Table 4:

Type	Argument	Description
CsrBtAvrcpRoleDetails	*details	A pointer to the parameter to include in `CsrBtAvrcpConfigReqSend`
CsrBtAvrcpConfigRoleMask	roleConfig	Local configuration for the specific role: – `CSR_BT_AVRCP_CONFIG_ROLE_STANDARD`: Standard configuration –`CSR_BT_AVRCP_CONFIG_ROLE_TG_BATT_SUPPORT`: The TG can display the battery level of the CT
CsrBtAvrcpConfigSrVersion	srAvrcpVersion	AVRCP version to announce in the service record for the role – set to one of the following: – `CSR_BT_AVRCP_CONFIG_SR_VERSION_10` – `CSR_BT_AVRCP_CONFIG_SR_VERSION_13` – `CSR_BT_AVRCP_CONFIG_SR_VERSION_14`
CsrBtAvrcpConfigSrFeatureMask	srFeatures	Features and categories to announce in the service record for the specific role – create a mask from the following defines: – `CSR_BT_AVRCP_CONFIG_SR_FEAT_CAT1_PLAY_REC` – `CSR_BT_AVRCP_CONFIG_SR_FEAT_CAT2_MON_AMP` – `CSR_BT_AVRCP_CONFIG_SR_FEAT_CAT3_TUNER` – `CSR_BT_AVRCP_CONFIG_SR_FEAT_CAT4_MENU` – `CSR_BT_AVRCP_CONFIG_SR_FEAT_PAS` – `CSR_BT_AVRCP_CONFIG_SR_FEAT_GROUP_NAV` – `CSR_BT_AVRCP_CONFIG_SR_FEAT_BROWSING` – `CSR_BT_AVRCP_CONFIG_SR_FEAT_MULTIPLE_MP`
CsrCharString	*providerName	NUL-terminated provider name to announce in the service record
CsrCharString	*serviceName	NUL-terminated service name to announce in the service record

Table 4: Arguments for the `CsrBtAvrcpConfigRoleNoSupport` and `CsrBtAvrcpConfigRoleSupport` helper functions

When the AVRCP module has finished handling the CSR_BT_AVRCP_CONFIG_REQ, it will confirm the request by sending a CSR_BT_AVRCP_CONFIG_CFM message, which contains the parameters found in Table 5:

Type	Parameter	Description
CsrBtAvrcpPrim	type	Signal identity – always CSR_BT_AVRCP_CONFIG_CFM
CsrBtResultCode	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.
CsrBtSupplier	resultSupplier	This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h

Table 5: Parameters in a CSR_BT_AVRCP_CONFIG_CFM message

3.2 Activation and incoming Connections

In order for the AVRCP module to accept incoming connections it must be activated by sending the CSR_BT_AVRCP_ACTIVATE_REQ message to the profile by means of the `CsrBtAvrcpActivateReqSend` function.

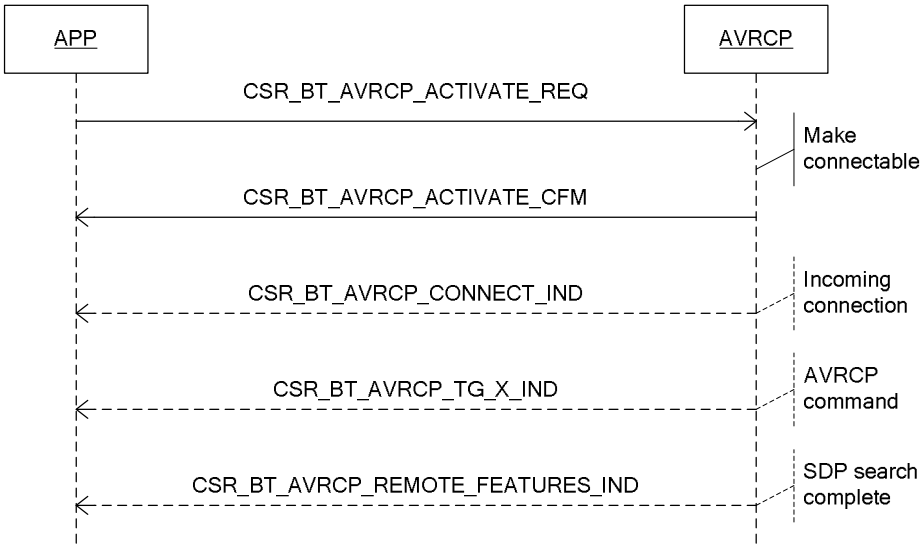


Figure 3: AVRCP activation and incoming connections

If a remote device establishes a connection to the AVRCP module, a CSR_BT_AVRCP_CONNECT_IND message will be sent to the control application. At this time, any target media players should be ready to receive AVRCP commands from a remote controller. Immediately following the establishment of an incoming connection, the AVRCP module will initiate SDP search against the remote device to determine the capabilities of the remote target and controller roles. When the SDP search is complete, a CSR_BT_AVRCP_REMOTE_FEATURES_IND message will be sent to the control application. Please refer to Figure 3 for an overview of the sequence.

The prototype for the `CsrBtAvrcpActivateReqSend` function is:

```
CsrBtAvrcpActivateReqSend(CsrUint8 maxIncoming)
```

The single argument to the function is described in Table 7.

Type	Argument	Description
CsrUint8	maxIncoming	Maximum number of simultaneous connections

Table 6: Arguments for the `CsrBtAvrcpActivateReqSend` function

When the AVRCP module has finished handling the CSR_BT_AVRCP_ACTIVATE_REQ, it will confirm the request by sending a CSR_BT_AVRCP_ACTIVATE_CFM message, which contains the parameters found in Table 7:

Type	Parameter	Description
CsrBtAvrcpPrim	type	Signal identity – always CSR_BT_AVRCP_ACTIVATE_CFM
CsrBtResultCode	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.
CsrBtSupplier	resultSupplier	This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h

Table 7: Parameters in a CSR_BT_AVRCP_ACTIVATE_CFM message

NOTE: The activation operation will fail if the TG role is supported and no media players are registered at all.

Following activation of the profile, incoming connections can be established. In the event of a successful incoming connection, a CSR_BT_AVRCP_CONNECT_IND message will be sent to the control application. The contents of the message are listed in Table 8:

Type	Parameter	Description
CsrBtAvrcpPrim	type	Signal identity – always CSR_BT_AVRCP_CONNECT_IND
CsrUint8	connectionId	Unique number for identifying the specific connection
deviceAddr_t	deviceAddr	The address of the remote device
CsrBtConnId	btConnId	Identifier used when moving the connection to another AMP controller, i.e. when calling the `CsrBtAmpmMoveReqSend`-function.

Table 8: Parameters in a CSR_BT_AVRCP_CONNECT_IND message

The control application will be notified about features and roles supported by a remote device with the CSR_BT_AVRCP_REMOTE_FEATURES_IND message with the parameters found in Table 9:

Type	Parameter	Description
CsrBtAvrcpPrim	type	Signal identity – always CSR_BT_AVRCP_REMOTE_FEATURES_IND
CsrUint8	connectionId	Unique number for identifying the specific connection
deviceAddr_t	deviceAddr	The address of the remote device
CsrBtAvrcpRoleDetails	tgFeatures	Refer to Table 10
CsrBtAvrcpRoleDetails	ctFeatures	Refer to Table 10

Table 9: Parameters in a CSR_BT_AVRCP_REMOTE_FEATURES_IND message

The parameters for the CsrBtAvrcpRoleDetails structure is as defined in Table 10:

Type	Parameter	Description
CsrBool	roleSupported	Is `TRUE` if the role is supported and `FALSE` otherwise
CsrBtAvrcpConfigRoleMask	roleConfig	Refer to Table 4
CsrBtAvrcpConfigSrVersion	srAvrcpVersion	Refer to Table 4
CsrBtAvrcpConfigSrFeatureMask	srFeatures	Refer to Table 4
CsrCharString	*providerName	Refer to Table 4
CsrCharString	*serviceName	Refer to Table 4

Table 10: Parameters in the CsrBtAvrcpRoleDetails structure

3.3 Deactivation

To prevent the profile from accepting incoming connections it must be deactivated by sending a CSR_BT_DEACTIVATE_REQ message by means of the `CsrBtAvrcpDeactivateReqSend` function. This is only necessary if the profile has been activated in advance. Refer to Figure 4.

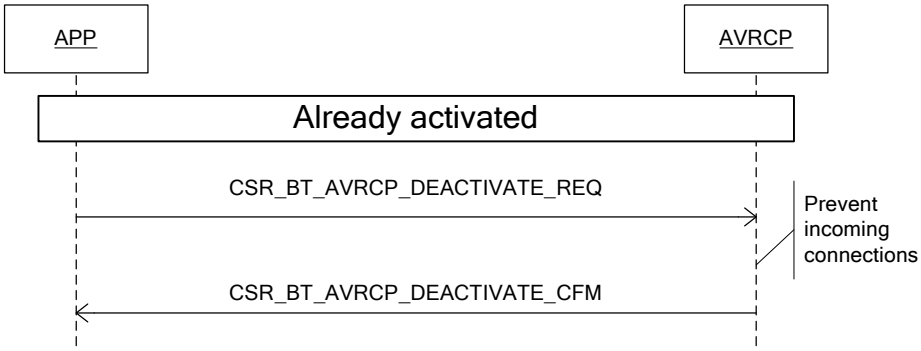


Figure 4: AVRCP deactivation

The prototype for the `CsrBtAvrcpDeactivateReqSend` function is:

```
CsrBtAvrcpDeactivateReqSend()
```

The function does not take any parameters.

When the AVRCP module has finished handling the CSR_BT_AVRCP_DEACTIVATE_REQ, it will confirm the request by sending a CSR_BT_AVRCP_DEACTIVATE_CFM message, which contains the parameters found in Table 11:

Type	Parameter	Description
CsrBtAvrcpPrim	Type	Signal identity – always CSR_BT_AVRCP_DEACTIVATE_CFM
CsrBtResultCode	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.
CsrBtSupplier	resultSupplier	This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h

Table 11: Parameters in a CSR_BT_AVRCP_DEACTIVATE_CFM message

NOTE: Deactivation of the profile does not imply release of existing active connections. Active connections must be released by the application prior to deactivation.

3.4 Outgoing Connection Establishment

In order to establish an outgoing connection to a remote device, the CSR_BT_AVRCP_CONNECT_REQ message can be utilised. An attempt to create an outgoing connection can be cancelled by sending a CSR_BT_AVRCP_CANCEL_CONNECT_REQ to the profile.

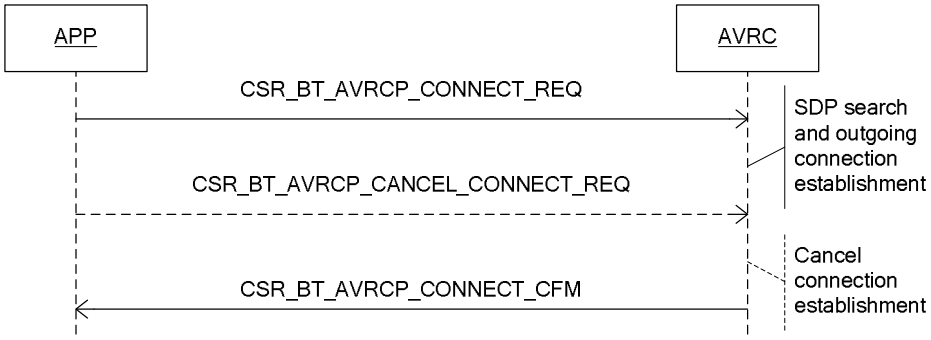


Figure 5: Outgoing connection establishment

The two messages can be sent with respectively the `CsrBtAvrcpConnectReqSend` function and the `CsrBtAvrcpCancelConnectReqSend` function.

The prototype for the `CsrBtAvrcpConnectReqSend` function is:

```
CsrBtAvrcpConnectReqSend (deviceAddr_t deviceAddr)
```

The prototype for the `CsrBtAvrcpCancelConnectReqSend` function is:

```
CsrBtAvrcpCancelConnectReqSend (deviceAddr_t deviceAddr)
```

The single argument to the functions is described in Table 12.

Type	Argument	Description
deviceAddr_t	deviceAddr	The address of the remote device

Table 12: Arguments for the `CsrBtAvrcpConnectReqSend` and the `CsrBtAvrcpCancelConnectReqSend` functions

When the AVRCP module has finished handling the CSR_BT_AVRCP_CONNECT_REQ, it will confirm the request by sending a CSR_BT_AVRCP_CONNECT_CFM message, which contains the parameters found in Table 13:

Type	Parameter	Description
CsrBtAvrcpPrim	Type	Signal identity – always CSR_BT_AVRCP_CONNECT_CFM
CsrUint8	connectionId	Unique number for identifying the specific connection – will be CSR_BT_`AVRCP_CONNECTION_ID_INVALID` if connection failed
deviceAddr_t	deviceAddr	The address of the remote device
CsrBtAvrcpRoleDetails	tgFeatures	Features supported by a remote target role – refer to Table 10
CsrBtAvrcpRoleDetails	ctFeatures	Features supported by a remote controller role – refer to Table 10
CsrBtResultCode	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.
CsrBtSupplier	resultSupplier	This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h
CsrBtConnId	btConnId	Identifier used when moving the connection to another AMP controller, i.e. when calling the `CsrBtAmpmMoveReqSend`-function.

Table 13: Parameters in a CSR_BT_AVRCP_CONNECT_CFM message

3.5 Disconnection

A successfully established connection can be disconnected by sending a CSR_BT_AVRCP_DISCONNECT_REQ message by using the `CsrBtAvrcpDisconnectReqSend` function. Local, remote and abnormal disconnections will result in a CSR_BT_AVRCP_DISCONNECT_IND message to be sent to the control application. This is illustrated Figure 6.

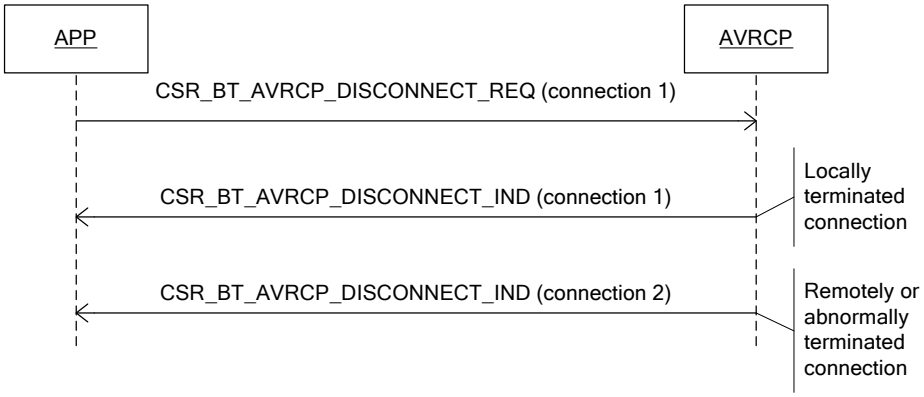


Figure 6: Disconnection

The prototype for the `CsrBtAvrcpDisconnectReqSend` function is:

```
CsrBtAvrcpDisconnectReqSend (CsrUint8 connectionId)
```

The single argument to the function is described in Table 14.

Type	Argument	Description
CsrUint8	connectionId	The unique connection ID previously received in a CSR_BT_AVRCP_CONNECT_IND or CSR_BT_AVRCP_CONNECT_CFM

Table 14: Arguments for the `CsrBtAvrcpDisconnectReqSend` function

If the CSR_BT_AVRCP_DISCONNECT_REQ message results in a connection being disconnected a CSR_BT_AVRCP_DISCONNECT_IND message is sent to the control application. This will also occur if the connection is terminated for any other reason. Refer to Table 15 for a description of the parameters in the CSR_BT_AVRCP_DISCONNECT_IND message.

Type	Parameter	Description
CsrBtAvrcpPrim	type	Signal identity – always CSR_BT_AVRCP_DISCONNECT_IND
CsrUint8	connectionId	The unique connection ID previously received in a CSR_BT_AVRCP_CONNECT_IND or CSR_BT_AVRCP_CONNECT_CFM
CsrBtReasonCode	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.
CsrBtSupplier	reasonSupplier	This parameter specifies the supplier of the reason given in reasonCode. Possible values can be found in csr_bt_result.h
CsrBool	localTerminated	TRUE if termination of connection happened on request from the local host; FALSE otherwise.

Table 15: Parameters in a CSR_BT_AVRCP_DISCONNECT_IND message

3.6 Security

Applications that wish to change the enforcement to a specific profile security level, i.e. authentication, encryption and/or authorisation, can use this API to set up 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 afterwards having to authorise as they both 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 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.

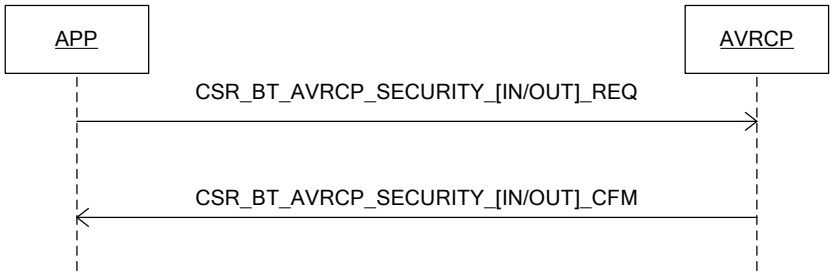


Figure 7: Security interface

The security for the two L2CAP channels used by AVRCP can be set by sending either the CSR_BT_AVRCP_SECURITY_IN_REQ or the CSR_BT_AVRCP_SECURITY_OUT_REQ message to the module by using respectively the `CsrBtAvrcpSecurityInReqSend` or `CsrBtAvrcpSecurityOutReqSend` functions.

The prototype for the `CsrBtAvrcpSecurityInReqSend` function is:

```
CsrBtAvrcpSecurityInReqSend(CsrSchedQid appHandle, CsrUint16 secLevel,
CsrBtAvrcpSecurityConfig config)
```

The prototype for the `CsrBtAvrcpSecurityOutReqSend` function is:

```
CsrBtAvrcpSecurityOutReqSend (CsrSchedQid appHandle, CsrUint16 secLevel,
CsrBtAvrcpSecurityConfig config)
```

The arguments for the function are described in Table 16.

Type	Argument	Description
CsrSchedQid	phandle	ID of the task to where the confirmation should be sent
CsrUint16	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)

Type	Argument	Description
CsrBtAvrcpSecurityConfig	config	This parameter specifies which channel (control or browsing) to update the security for – use one of the following values: – CSR_BT_AVRCP_SECURITY_CONFIG_CONTROL – CSR_BT_AVRCP_SECURITY_CONFIG_BROWSING – CSR_BT_AVRCP_SECURITY_CONFIG_ALL

Table 16: Arguments for the `CsrBtAvrcpSecurityInReqSend` and `CsrBtAvrcpSecurityOutReqSend` functions

When the AVRCP module has finished handling the CSR_BT_AVRCP_SECURITY_[IN/OUT]_REQ, it will confirm the request by sending a CSR_BT_AVRCP_SECURITY_[IN/OUT]_CFM message, which contains the parameters found in Table 17.

Type	Parameter	Description
CsrBtAvrcpPrim	type	Signal identity – always CSR_BT_AVRCP_SECURITY_[IN/OUT]_CFM
CsrBtResultCode	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.
CsrBtSupplier	resultSupplier	This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h

Table 17: Parameters in a CSR_BT_AVRCP_SECURITY_[IN/OUT]_CFM message

3.7 Media Player Registration (TG only)

Before the target role can be used, at least one media player must be registered. If the device supports the TG role and no media players are registered, activation of the profile will fail. By registering a media player, the profile can independently handle some commands from a controller. This relates to retrieval of Player Application Settings, supported notifications and retrieval of media players list. Figure 8 gives an overview of the messages involved in media player registration.

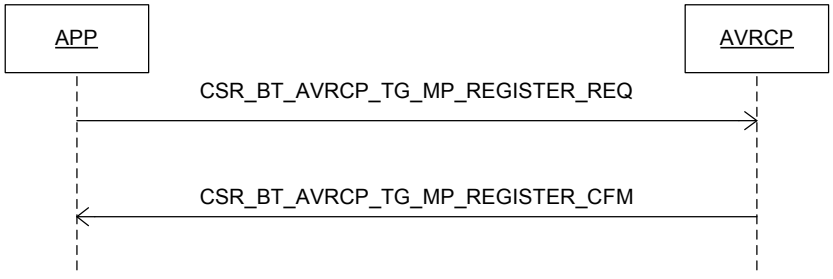


Figure 8: Media Player registration

In order to register a media player the CSR_BT_AVRCP_TG_MP_REGISTER_REQ message must be sent to the profile by means of the `CsrBtAvrcpTgMpRegisterReqSend` function.

The prototype for the `CsrBtAvrcpTgMpRegisterReqSend` function is:

```
   CsrBtAvrcpTgMpRegisterReqSend (CsrSchedQid playerHandle,
CsrBtAvrcpNotiMask   notificationMask, CsrBtAvrcpMpConfigMask configMask,
CsrUint16 pasLen,    CsrUint8 *pas, CsrBtAvrcpMpTypeMajor majorType,
CsrBtAvrcpMpTypeSub subType,  CsrBtAvrcpMpFeatureMask  featureMask,
CsrUtf8String *playerName);
```

The arguments for the function are described in Table 3.

Type	Argument	Description
CsrSchedQd	playerHandle	The task ID of the media player
CsrBtAvrcpNotiMask	notificationMask	Bitmask specifying which notifications the media player supports. Refer to the defines in csr_bt_avrcp_prim.h prefixed with CSR_BT_AVRCP_NOTI_FLAG_ – all the defines includes a cast to CsrBtAvrcpNotiMask
CsrBtAvrcpMpConfigMask	configMask	Bitmask for special configuration of the MP. Set to CSR_BT_AVRCP_TG_MP_REGISTER_CONFIG_NONE for no special configuration or to CSR_BT_AVRCP_TG_MP_REGISTER_CONFIG_SET_DEFAULT if the media player should be marked as the default media player.
CsrUint16	pasLen	Length of the data contained in *pas
CsrUint8	*pas	Tag-based byte stream containing all Player Application Settings for the target – refer to section 3.7.2 for information about how to create this
CsrBtAvrcpMpTypeMajor	majorType	Bitmask specifying the major type of the media player: – CSR_BT_AVRCP_MP_TYPE_MAJOR_AUDIO – CSR_BT_AVRCP_MP_TYPE_MAJOR_VIDEO – CSR_BT_AVRCP_MP_TYPE_MAJOR_BROAD_AUDIO – CSR_BT_AVRCP_MP_TYPE_MAJOR_BROAD_VIDEO
CsrBtAvrcpMpTypeSub	subType	Bitmask specifying the sub type of the media player: – CSR_BT_AVRCP_MP_TYPE_SUB_AUDIO_BOOK – CSR_BT_AVRCP_MP_TYPE_SUB_PODCAST
CsrBtAvrcpMpFeatureMask	featureMask	128 bit mask specifying the features supported by the media player. The mask is divided into four blocks of 32 bit. Refer to section 3.7.1 for information about the feature mask.
CsrUtf8String	*playerName	NUL-terminated string (UTF-8) containing the name of the Media Player

Table 18: Arguments for the `CsrBtAvrcpTgMpRegisterReqSend` function

3.7.1 Feature Mask

This is a 128 bit mask with the features supported by the media player. The mask is divided into four blocks each of them 32-bit long. The information that can be specified with this mask relates to features mentioned in the "feature bit mask" section in the AVRCP 1.4 spec and is the following:

- Pass through commands supported.
- Basic group navigation support
- Advanced control support
- Browsing support
- Searching support
- Add to now playing feature support
- Unique UIDs support
- Only browsable when addressed support
- Only searchable when addressed support
- Now playing folder support

- UID persistency support

All these features can be enabled or disabled using mask definition values defined in the csr_bt_avrcp_prim.h file. Only 67 bits of the 128 are used so far. The rest are reserved for future use if new features need to be specified.

3.7.2 Building *pas

In order to create the tag-based byte stream, two helper functions are provided: `CsrBtAvrcpTgLibPasAttribAdd` for adding attributes and `CsrBtAvrcpTgLibPasValueAdd` for adding values to an existing attribute. The prototypes for the functions are specified in the following:

```
void CsrBtAvrcpTgLibPasAttribAdd(CsrUint16 *pasLen, CsrUint8 **pas,
   CsrBtAvrcpPasAttId_t attId, CsrUtf8String *attTxt)

void CsrBtAvrcpTgLibPasValueAdd(CsrUint16 *pasLen, CsrUint8 **pas,
      CsrBtAvrcpPasAttId_t attId, CsrBtAvrcpPasValId valId,
CsrUtf8String *valTxt)
```

The arguments for the two functions can be found in Table 19:

Type	Argument	Description
CsrUint16	*pasLen	Length of the PAS data – will be maintained by the helper functions
CsrUint8	**pas	The PAS data itself – will be maintained by the helper functions
CsrBtAvrcpPasAttId_t	attId	ID of the attribute to add or update: – `CSR_BT_AVRCP_PAS_EQUALIZER_ATT_ID` – `CSR_BT_AVRCP_PAS_REPEAT_ATT_ID` – `CSR_BT_AVRCP_PAS_SHUFFLE_ATT_ID` – `CSR_BT_AVRCP_PAS_SCAN_ATT_ID` – `CSR_BT_AVRCP_PAS_EXT_ATT_ID_BEGIN`: Start value for target specified attributes
CsrUtf8String	*attTxt	NUL-terminated text (UTF-8) of the attribute – set to NULL if no text is available
CsrBtAvrcpPasValId	valId	ID of the value to add – refer to the defines csr_bt_avrcp_prim.h that are cast to CsrBtAvrcpPasValId
CsrUtf8String	*valTxt	NUL-terminated text (UTF-8) of the value – set to NULL if no text is available

Table 19: Arguments for the `CsrBtAvrcpTgLibPasAttribAdd` and `CsrBtAvrcpTgLibPasValueAdd` functions

Example of adding Player Application Settings

This example demonstrates how to add Player Application Settings for an equalizer:

```
CsrUint16 pasLen = 0;
CsrUint8 *pas = NULL;

CsrBtAvrcpTgLibPasAttribAdd(&pasLen, &pas,
   CSR_BT_AVRCP_PAS_EQUALIZER_ATT_ID, "Equalizer");

CsrBtAvrcpTgLibPasValueAdd(&pasLen, &pas,
CSR_BT_AVRCP_PAS_EQUALIZER_ATT_ID,
   CSR_BT_AVRCP_PAS_EQUALIZER_VAL_OFF, "Off");

CsrBtAvrcpTgLibPasValueAdd(&pasLen, &pas,
CSR_BT_AVRCP_PAS_EQUALIZER_ATT_ID,
   CSR_BT_AVRCP_PAS_EQUALIZER_VAL_ON, "On");
```

When the AVRCP module has finished handling the CSR_BT_AVRCP_TG_MP_REGISTER_REQ, it will confirm the request by sending a CSR_BT_AVRCP_TG_MP_REGISTER_CFM message, which contains the parameters found in Table 20:

Type	Parameter	Description
CsrBtAvrcpPrim	type	Signal identity – always CSR_BT_AVRCP_TG_MP_REGISTER_CFM
CsrUint32	playerId	A unique number for use in later references to the media player
CsrBtResultCode	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.
CsrBtSupplier	resultSupplier	This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h

Table 20: Parameters in a CSR_BT_AVRCP_TG_MP_REGISTER_CFM message

3.8 Media Player Unregistration

Registered media players can be unregistered by sending a CSR_BT_AVRCP_TG_MP_REGISTER_REQ to the profile by means of the `CsrBtAvrcpTgMpUnregisterReqSend` function. Addressed media players cannot be unregistered. Thus, this operation will fail if the media player being unregistered is an addressed media player.

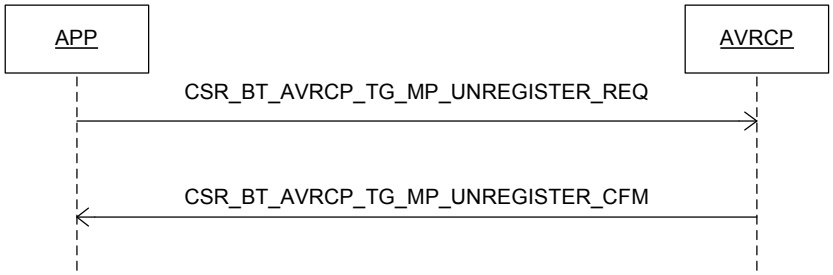


Figure 9: Media player unregistration

The prototype for the `CsrBtAvrcpTgMpUnregisterReqSend` function is:

```
CsrBtAvrcpTgMpUnregisterReqSend(CsrSchedQid pHandle, CsrUint32 playerId)
```

The arguments for the function are described in Table 21.

Type	Argument	Description
CsrSchedQid	phandle	ID of the task to where the CSR_BT_AVRCP_TG_MP_UNREGISTER_CFM should be sent
CsrUint32	playerId	The same player ID that was previously received in a CSR_BT_AVRCP_TG_MP_REGISTER_CFM message.

Table 21: Arguments for `CsrBtAvrcpTgMpUnregisterReqSend` function

When the AVRCP module has finished handling the CSR_BT_AVRCP_TG_MP_UNREGISTER_REQ, it will confirm the request by sending a CSR_BT_AVRCP_TG_MP_UNREGISTER_CFM message, which contains the parameters found in Table 22.

Type	Parameter	Description
CsrBtAvrcpPrim	type	Signal identity – always CSR_BT_AVRCP_TG_MP_UNREGISTER_CFM
CsrUint32	playerId	The unique number of the media player that was unregistered

Type	Parameter	Description
CsrBtResultCode	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.
CsrBtSupplier	resultSupplier	This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h

Table 22: Parameters in a CSR_BT_AVRCP_TG_MP_UNREGISTER_CFM message

3.9 Notifications

In order for the controller to remain updated about the status of the target, it can register for a range of different notifications. Figure 10 depicts an overview of the notification interface.

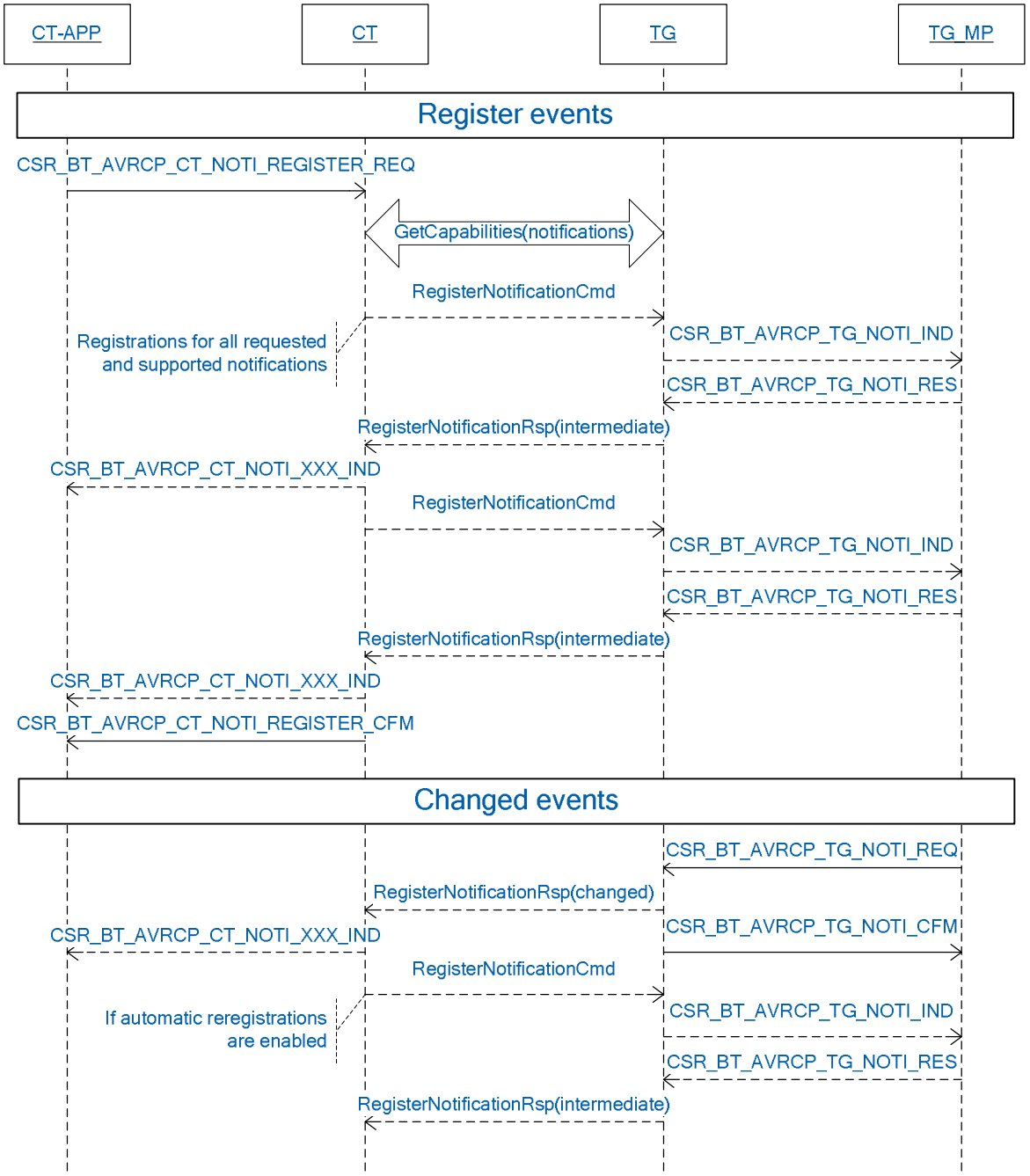


Figure 10: Notification overview

3.9.1 Controller Interface

A controller can choose to register for notifications by means of the CSR_BT_AVRCP_CT_NOTI_REGISTER_REQ message, which is sent by the function `CsrBtAvrcpCtNotiRegisterReqSend`. It is possible to specify whether the registrations should be persistent – i.e. whether the profile automatically should reregister for completed notifications.

Immediately following the registration of a notification, an indication with the current status will be sent to the controller application. A unique indication is available for each type of notification.

The prototype for the `CsrBtAvrcpCtNotiRegisterReqSend` function is:

```
CsrBtAvrcpCtNotiRegisterReqSend(CsrSchedQid phandle, CsrUint8 connId,
CsrBtAvrcpNotiMask notiMask, CsrUint32 playbackInterval,
CsrBtAvrcpNotiRegConfigMask configMask)
```

The arguments for the function are described in Table 23.

Type	Argument	Description
CsrSchedQid	phandle	ID of the task to where the confirmation message should be sent
CsrUint8	connId	The unique connection ID previously received in a CSR_BT_AVRCP_CONNECT_IND/CFM message
CsrBtAvrcpNotiMask	notiMask	Use a mask of the following defines to specify which notifications to register for: - CSR_BT_AVRCP_NOTI_FLAG_PLAYBACK_STATUS - CSR_BT_AVRCP_NOTI_FLAG_TRACK - CSR_BT_AVRCP_NOTI_FLAG_TRACK_END - CSR_BT_AVRCP_NOTI_FLAG_TRACK_START - CSR_BT_AVRCP_NOTI_FLAG_PLAYBACK_POS - CSR_BT_AVRCP_NOTI_FLAG_BATT_STATUS - CSR_BT_AVRCP_NOTI_FLAG_SYSTEM_STATUS - CSR_BT_AVRCP_NOTI_FLAG_PAS - CSR_BT_AVRCP_NOTI_FLAG_NOW_PLAYING_CONTENT - CSR_BT_AVRCP_NOTI_FLAG_AVAILABLE_PLAYERS - CSR_BT_AVRCP_NOTI_FLAG_ADDRESSED_PLAYER - CSR_BT_AVRCP_NOTI_FLAG_UIDS - CSR_BT_AVRCP_NOTI_FLAG_VOLUME The following to defines can be used instead of masking the ones above for certain configurations: - CSR_BT_AVRCP_NOTI_FLAG_ALL - CSR_BT_AVRCP_NOTI_FLAG_AVRCP13_ONLY
CsrUint32	playbackInterval	The interval in milliseconds at which the remote target should send a notification with the current playback position
CsrBtAvrcpNotiRegConfigMask	configMask	Defines used for configuring the notifications, which can either be persistent or not. If _STANDARD (persistent) is used the AVRCP profile handles re-newing the notification reqistration if the status is changed. - CSR_BT_AVRCP_NOTI_REG_STANDARD - CSR_BT_AVRCP_NOTI_REG_NON_PERSISTENT

Table 23: Arguments for CsrBtAvrcpCtNotiRegisterReqSend and function

When the AVRCP module has finished handling the CSR_BT_AVRCP_CT_NOTI_REGISTER_REQ, it will confirm the request by sending a CSR_BT_AVRCP_CT_NOTI_REGISTER_CFM message, which contains the parameters found in Table 24:

Type	Parameter	Description
CsrBtAvrcpPrim	type	Signal identity – always CSR_BT_AVRCP_CT_NOTI_REGISTER_CFM
CsrUint8	connectionId	The unique connection ID previously received in a CSR_BT_AVRCP_CONNECT_IND/CFM message
CsrBtAvrcpNotiMask	notiMask	See Table 23
CsrUint32	playbackInterval	The same value that was included in the request

Type	Parameter	Description
CsrBtResultCode	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.
CsrBtSupplier	resultSupplier	This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h

Table 24: Parameters in a CSR_BT_AVRCP_CT_NOTI_REGISTER_CFM message

Each type of notification response will result in a CSR_BT_AVRCP_CT_NOTI_XXX_IND message to be sent to the application, where the XXX part of the message depends on the notification type. The parameters common between the messages is listed in Table 25:

Type	Parameter	Description
CsrBtAvrcpPrim	type	Signal identity – always CSR_BT_AVRCP_CT_NOTI_XXX_IND
CsrUint8	connectionId	The unique connection ID previously received in a CSR_BT_AVRCP_CONNECT_IND/CFM message

Table 25: Common parameters in a CSR_BT_AVRCP_CT_NOTI_XXX_IND message

The parameters unique to each type of indication are listed in Table 26. For example, the message CSR_BT_AVRCP_CT_NOTI_UIDS_IND has the parameters `type`, `connectionId` and `uidCounter`.

Message (XXX-part)	Type	Parameter	Description
UIDS	CsrUint16	uidCounter	The current UID counter – refer to section 2.3.2 for details
VOLUME	CsrUint8	Volume	Indicates that the volume has changed. The volume is specified as a percentage of the maximum. The value 0x0 corresponds to 0%. The value 0x7F corresponds to 100%.
BATTERY_STATUS	CsrBtAvrcpBatteryStatus	batStatus	Indicates current battery status. Allowed values are: CSR_BT_AVRCP_BATTERY_STATUS_NORMAL CSR_BT_AVRCP_BATTERY_STATUS_WARNING CSR_BT_AVRCP_BATTERY_STATUS_CRITICAL CSR_BT_AVRCP_BATTERY_STATUS_EXTERNAL CSR_BT_AVRCP_BATTERY_STATUS_FULL_CHARGE
PLAYBACK_POS	CsrUint32	playbackPos	Indicates playback position of current track in milliseconds. If no track is currently selected, this parameter will have the following value: 0xFFFFFFFF
SYSTEM_STATUS	CsrBtAvrcpSystemStatus	systemStatus	Indicates current system status. Allowed values are: Allowed values are: CSR_BT_AVRCP_SYSTEM_STATUS_POWER_ON CSR_BT_AVRCP_SYSTEM_STATUS_POWER_OFF CSR_BT_AVRCP_SYSTEM_STATUS_UNPLUGGED
TRACK_CHANGED	CsrBtAvrcpUid	trackUid	The UID of the item – refer to section 2.3.2 for details
TRACK_END	-	-	Indicates that the track end is reached
TRACK_START	-	-	Indicates that the track start is reached
PLAYBACK_STATUS	CsrBtAvrcpPlaybackStatus	playbackStatus	Indicates the current status of playback. Allowed values are: CSR_BT_AVRCP_PLAYBACK_STATUS_STOPPED CSR_BT_AVRCP_PLAYBACK_STATUS_PLAYING CSR_BT_AVRCP_PLAYBACK_STATUS_PAUSED CSR_BT_AVRCP_PLAYBACK_STATUS_FWD_SEEK CSR_BT_AVRCP_PLAYBACK_STATUS_REV_SEEK CSR_BT_AVRCP_PLAYBACK_STATUS_ERROR
AVAILABLE_PLAYERS	-	-	Indicates that a new player has become available or that a previously available player is now not available anymore.
NOW_PLAYING	-	-	Indicates that the content of the Now Playing folder is changed for the Addressed Player.

Table 26: Unique parameters in the CSR_BT_AVRCP_CT_NOTI_XXX_IND messages

3.9.2 Target interface

When a controller registers for a notification, type that has previously been enabled when registering a media player, a CSR_BT_AVRCP_TG_NOTI_IND message, with the parameters in Table 27, will be sent to the addressed media player.

Type	Parameter	Description
CsrBtAvrcpPrim	type	Signal identity – always CSR_BT_AVRCP_CT_NOTI_XXX_IND

Type	Parameter	Description
CsrUint8	connectionId	The unique connection ID previously received in a CSR_BT_AVRCP_CONNECT_IND/CFM message
CsrUint32	playerId	Unique ID of the media player the message is intended for
CsrBtAvrcpNotiId	notiId	The ID of the notification: - CSR_BT_AVRCP_NOTI_ID_PLAYBACK_STATUS - CSR_BT_AVRCP_NOTI_ID_TRACK - CSR_BT_AVRCP_NOTI_ID_TRACK_END - CSR_BT_AVRCP_NOTI_ID_TRACK_START - CSR_BT_AVRCP_NOTI_ID_PLAYBACK_POS - CSR_BT_AVRCP_NOTI_ID_BATT_STATUS - CSR_BT_AVRCP_NOTI_ID_SYSTEM_STATUS - CSR_BT_AVRCP_NOTI_ID_PAS - CSR_BT_AVRCP_NOTI_ID_NOW_PLAYING_CONTENT - CSR_BT_AVRCP_NOTI_ID_AVAILABLE_PLAYERS - CSR_BT_AVRCP_NOTI_ID_ADDRESSED_PLAYER - CSR_BT_AVRCP_NOTI_ID_UIDS - CSR_BT_AVRCP_NOTI_ID_INVALID
CsrUint32	playbackInterval	The interval in milliseconds at which the target should send a notification with the current playback position
CsrUint32	msgId	Unique message ID for associating indications and responses – use the ID from an indication in the matching response

Table 27: Common parameters in a CSR_BT_AVRCP_TG_NOTI_XXX_IND message

The media player must reply to this message by sending a CSR_BT_AVRCP_TG_NOTI_RES message to the profile. A number of helper functions are provided for sending the message. When an event occurs on the media player for notifications it supports, it must send a CSR_BT_AVRCP_TG_NOTI_REQ message to the profile to complete any outstanding notifications. Several helper functions are provided for this purpose.

The arguments in common for the functions CsrBtAvrcpTgNotiXXXRes (for sending the CSR_BT_AVRCP_TG_NOTI_RES message) is listed in Table 28:

Type	Argument	Description
CsrUint8	connId	The unique connection ID previously received in a CSR_BT_AVRCP_CONNECT_IND/CFM message
CsrBtAvrcpStatus	status	Indicates whether the operation is accepted or rejected by means of the defines prefixed with CSR_BT_AVRCP_STATUS_ in csr_bt_avrcp_prim.h
CsrUint32	msgId	Unique message ID for associating indications and responses – use the ID from an indication in the matching response

Table 28: Unique arguments for the CsrBtAvrcpTgNotiXXXRes function

The arguments in common for the functions CsrBtAvrcpTgNotiXXXReq (for sending the CSR_BT_AVRCP_TG_NOTI_RES message) is listed in Table 29:

Type	Argument	Description
CsrUint32	playerId	Unique ID of the media player the message is intended for

Table 29: Unique arguments for the CsrBtAvrcpTgNotiXXXReq function

A list of the functions themselves and unique arguments can be found in Table 30:

Function (XXX-part)	Type	Argument	Description
PlaybackStatus	CsrBtAvrcpPlaybackStatus	playbackStatus	Indicates the current status of playback. Allowed values are: CSR_BT_AVRCP_PLAYBACK_STATUS_STOPPED CSR_BT_AVRCP_PLAYBACK_STATUS_PLAYING CSR_BT_AVRCP_PLAYBACK_STATUS_PAUSED CSR_BT_AVRCP_PLAYBACK_STATUS_FWD_SEEK CSR_BT_AVRCP_PLAYBACK_STATUS_REV_SEEK CSR_BT_AVRCP_PLAYBACK_STATUS_ERROR
Track	CsrBtAvrcpUid	Uid	The UID of the item – refer to section 2.3.2 for details
TrackStartEnd	CsrBool	Start	Track end or track start reached. If TRUE the start position is reached and if FALSE, the end position is reached
PlaybackPosition	CsrUint32	playbackPos	Indicates playback position of current track in milliseconds. If no track is currently selected, this parameter will have the following value: 0xFFFFFFFF
BatStatus	CsrBtAvrcpBatteryStatus	batStatus	Indicates current battery status. Allowed values are: CSR_BT_AVRCP_BATTERY_STATUS_NORMAL CSR_BT_AVRCP_BATTERY_STATUS_WARNING CSR_BT_AVRCP_BATTERY_STATUS_CRITICAL CSR_BT_AVRCP_BATTERY_STATUS_EXTERNAL CSR_BT_AVRCP_BATTERY_STATUS_FULL_CHARGE
SystemStatus	CsrBtAvrcpSystemStatus	sysStatus	Indicates current system status. Allowed values are: Allowed values are: CSR_BT_AVRCP_SYSTEM_STATUS_POWER_ON CSR_BT_AVRCP_SYSTEM_STATUS_POWER_OFF CSR_BT_AVRCP_SYSTEM_STATUS_UNPLUGGED
NowPlaying	-	-	Indicates that the content of the Now Playing folder is changed for the Addressed Player.
Uids	CsrUint16	uidCounter	The current UID counter – refer to section 2.3.2 for details
AddressedPlayer			Indicates that the Addressed Player is changed
	CsrUint16	playerId	Unique ID of the media player

Table 30: Unique arguments for the CsrBtAvrcpTgNotiXXXRes/Req functions

This implies for example that the prototype for the function CsrBtAvrcpTgNotiPlaybackStatusRes is defined as the following:

```
CsrBtAvrcpTgNotiPlaybackStatusRes(CsrUint8 connId, CsrBtAvrcpStatus
status, CsrUint32 msgId, CsrBtAvrcpPlaybackStatus playbackStatus)
```

3.10 Player Application Settings Retrieval

A controller can retrieve the Player Application Settings provided by a remote target through several interfaces. One group of interfaces covers the retrieval of attribute and value IDs (refer to section 3.10.1) and another group the retrieval of text for attributes and values (refer to section 3.10.2).

On the target side the media player itself will not be queried for this information as everything is handled internally by the profile manager.

3.10.1 Attribute and Value ID Retrieval

Figure 11 illustrates the concept of retrieving attribute and value IDs. A CSR_BT_AVRCP_CT_PAS_ATT_ID_REQ must be sent to get a list of attributes supported by a target. For each attribute a CSR_BT_AVRCP_CT_PASS_VAL_ID_REQ can be sent to retrieve a list of value IDs supported by the specific attribute.

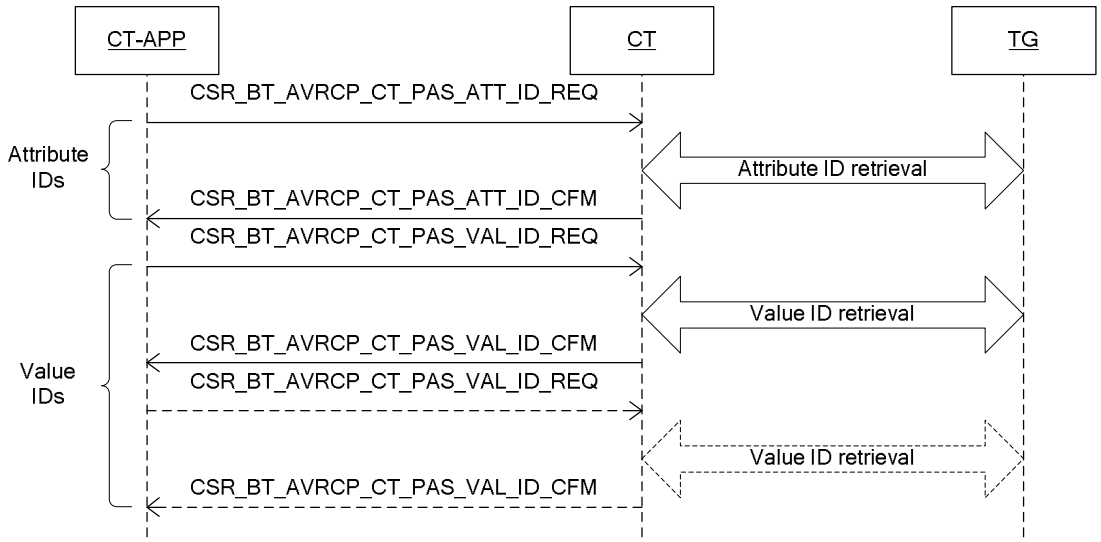


Figure 11: Retrieval of PAS attribute and value IDs from a remote target

The two functions `CsrBtAvrcpCtPasAttIdReqSend` and `CsrBtAvrcpCtPasValIdReqSend` are provided for sending the messages and the prototypes for the functions are listed below:

```
CsrBtAvrcpCtPasAttIdReqSend (CsrSchedQid phandle, CsrUint8 connId)

CsrBtAvrcpCtPasValIdReqSend (CsrSchedQid phandle, CsrUint8 connId,
CsrBtAvrcpPasAttId_t attribId)
```

The arguments for the functions are described in Table 31:

Type	Argument	Description
`CsrSchedQid`	`phandle`	ID of the task to where the confirmation message should be sent
`CsrUint8`	`connId`	The unique connection ID previously received in a CSR_BT_AVRCP_CONNECT_IND/CFM message
`CsrBtAvrcpPasAttId_t`	`attribId`	ID of the attribute to retrieve value IDs for – the following IDs are predefined: – `CSR_BT_AVRCP_PAS_EQUALIZER_ATT_ID` – `CSR_BT_AVRCP_PAS_REPEAT_ATT_ID` – `CSR_BT_AVRCP_PAS_SHUFFLE_ATT_ID` – `CSR_BT_AVRCP_PAS_SCAN_ATT_ID` – `CSR_BT_AVRCP_PAS_EXT_ATT_ID_BEGIN`: Start value for target specified attributes

Table 31: Arguments for the `CsrBtAvrcpCtPasAttIdReqSend` and `CsrBtAvrcpCtPasValIdReqSend` functions

At the completion of the attribute ID retrieval, a CSR_BT_AVRCP_CT_ATT_ID_CFM message will be sent to the application. The parameters in the message are listed in Table 32:

Type	Parameter	Description
CsrBtAvrcpPrim	type	Signal identity – always CSR_BT_AVRCP_CT_PAS_ATT_ID_CFM
CsrUint8	connectionId	The unique connection ID previously received in a CSR_BT_AVRCP_CONNECT_IND/CFM message
CsrUint8	attIdCount	Number of returned attributes
CsrBtAvrcpPasAttId_t	*attId	Pointer to the attribute IDs
CsrBtResultCode	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.
CsrBtSupplier	resultSupplier	This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h

Table 32: Parameters in a CSR_BT_AVRCP_CT_PAS_ATT_ID_CFM message

Following the completion of each value ID retrieval, a CSR_BT_AVRCP_CT_VAL_ID_CFM message will be sent to the application. The parameters in the message are listed in Table 33:

Type	Parameter	Description
CsrBtAvrcpPrim	type	Signal identity – always CSR_BT_AVRCP_CT_PAS_VAL_ID_CFM
CsrUint8	connectionId	The unique connection ID previously received in a CSR_BT_AVRCP_CONNECT_IND/CFM message
CsrBtAvrcpPasAttId_t	attId	The same attribute ID that was included in the request
CsrUint8	valIdCount	Number of returned values
CsrBtAvrcpPasValId	*valId	Pointer to the value IDs
CsrBtResultCode	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.
CsrBtSupplier	resultSupplier	This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h

Table 33: Parameters in a CSR_BT_AVRCP_CT_PAS_VAL_ID_CFM message

3.10.2 Text Retrieval

If a target provides any extended player application settings (attribute IDs >= CSR_BT_AVRCP_PAS_EXT_ATT_ID_BEGIN) the controller can retrieve the text for the attributes and associated values using the interfaces illustrated in Figure 12. Text for non-extended attributes should be provided by the controller itself.

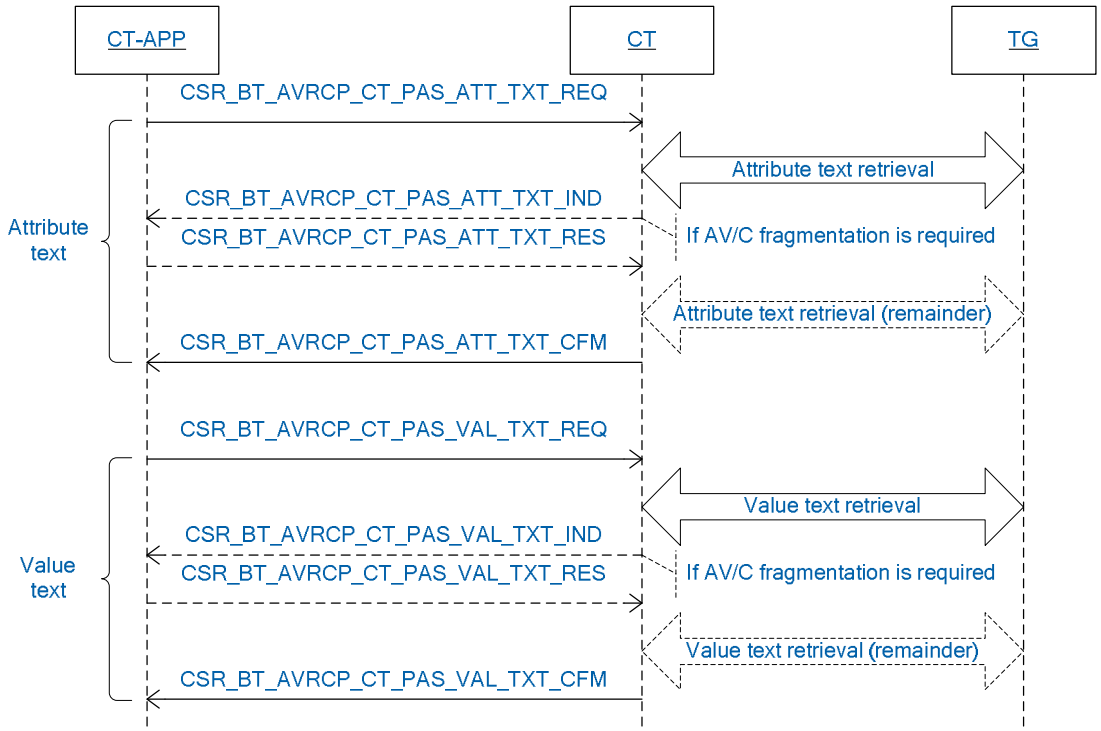


Figure 12: Retrieval of PAS attribute and value text from a remote target

To initiate retrieval of either attribute or value text the `CsrBtAvrcpCtPasAttTxtReqSend` and `CsrBtAvrcpCtPasValTxtReqSend` functions can be used respectively.

The prototype for the `CsrBtAvrcpCtPasAttTxtReqSend` function is:

```
CsrBtAvrcpCtPasAttTxtReqSend (CsrSchedQid phandle, CsrUint8 connId, CsrUint8
attribIdCount, CsrBtAvrcpPasAttId_t *attribId)
```

The arguments for the function are described in Table 34.

Type	Argument	Description
`CsrSchedQid`	`phandle`	ID of the task to where the confirmation message should be sent
`CsrUint8`	`connId`	The unique connection ID previously received in a CSR_BT_AVRCP_CONNECT_IND/CFM message
`CsrUint8`	`attribIdCount`	Number of attributes for which to retrieve the text
`CsrBtAvrcpPasAttId_t`	`*attribId`	Pointer to the specified number of attribute IDs

Table 34: Arguments for the `CsrBtAvrcpCtPasAttTxtReqSend` function

The prototype for the `CsrBtAvrcpCtPasValTxtReqSend` function is:

```
CsrBtAvrcpCtPasValTxtReqSend (CsrSchedQid phandle, CsrUint8 connId,
CsrBtAvrcpPasAttId_t attribId, CsrUint8 valIdCount, CsrBtAvrcpPasValId *valId)
```

The arguments for the function are described in Table 35.

Type	Argument	Description
`CsrSchedQid`	`phandle`	ID of the task to where the confirmation message should be sent

Type	Argument	Description
CsrUint8	connId	The unique connection ID previously received in a CSR_BT_AVRCP_CONNECT_IND/CFM message
CsrBtAvrcpPasAttId_t	attribId	ID of the attribute for which the value IDs should be retrieved
CsrUint8	valIdCount	Number of values for which to retrieve the text
CsrBtAvrcpPasValId	*valId	Pointer to the specified number of value IDs

Table 35: Arguments for the `CsrBtAvrcpCtPasValTxtReqSend` function

Fragmentation

PAS information is transferred using AV/C packets and since the maximum size of these is limited to 512 bytes, fragmentation can occur. In case of fragmentation a CSR_BT_AVRCP_CT_PAS_[ATT/VAL]_TXT_IND message will be sent to the application, which must choose either to request the next remaining fragment or to abort the procedure by sending a CSR_BT_AVRCP_CT_PAS_[ATT/VAL]_TXT_RES message to the profile.

Indications will continually be sent to the application until all of the PAS data is transferred or until the controller aborts the procedure.

In order to send the CSR_BT_AVRCP_CT_PAS_[ATT/VAL]_TXT_RES messages the following two functions can be used:

```
CsrBtAvrcpCtPasAttTxtResSend (CsrUint8 connId, CsrBool proceed)

CsrBtAvrcpCtPasValTxtResSend (CsrUint8 connId, CsrBool proceed)
```

The arguments for the functions are described in Table 36.

Type	Argument	Description
CsrUint8	connId	The unique connection ID previously received in a CSR_BT_AVRCP_CONNECT_IND/CFM message
CsrBool	proceed	Set to TRUE to get next fragment or to FALSE to abort the procedure

Table 36: Arguments for `CsrBtAvrcpCtPas[Att/Val]TxtResSend` function

If the text for the requested attributes is fragmented a CSR_BT_AVRCP_CT_PAS_ATT_TXT_IND will be sent to the application. The parameters included in the message are listed in Table 37.

Type	Parameter	Description
CsrBtAvrcpPrim	type	Signal identity – always CSR_BT_AVRCP_CT_PS_ATT_TXT_IND
CsrUint8	connectionId	The unique connection ID previously received in a CSR_BT_AVRCP_CONNECT_IND/CFM message
CsrUint16	pasDataLen	Refer to the end of the section
CsrUint8	*pasData	Refer to the end of the section

Table 37: Parameters in a CSR_BT_AVRCP_CT_PAS_ATT_TXT_IND message

When the text for the requested attributes is fully retrieved or the request has been aborted, a CSR_BT_AVRCP_CT_PAS_ATT_TXT_CFM will be sent to the application. The parameters included in the message are listed in Table 38.

Type	Parameter	Description
CsrBtAvrcpPrim	type	Signal identity – always CSR_BT_AVRCP_CT_PAS_ATT_TXT_CFM
CsrUint8	connectionId	The unique connection ID previously received in a CSR_BT_AVRCP_CONNECT_IND/CFM message
CsrUint16	pasDataLen	Refer to the end of the section

Type	Parameter	Description
CsrUint8	*pasData	Refer to the end of the section
CsrBtResultCode	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.
CsrBtSupplier	resultSupplier	This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h

Table 38: Parameters in the CSR_BT_AVRCP_CT_PAS_ATT_TXT_CFM message

If the text for the requested attributes is fragmented a CSR_BT_AVRCP_CT_PAS_ATT_TXT_IND will be sent to the application. The parameters included in the message are listed in Table 39.

Type	Parameter	Description
CsrBtAvrcpPrim	type	Signal identity – always CSR_BT_AVRCP_CT_PAS_VAL_TXT_IND
CsrUint8	connectionId	The unique connection ID previously received in a CSR_BT_AVRCP_CONNECT_IND/CFM message
CsrBtAvrcpPasAttId_t	attId	The same attribute ID that was specified in the request
CsrUint16	pasDataLen	Refer to the end of the section
CsrUint8	*pasData	Refer to the end of the section

Table 39: Parameters in the CSR_BT_AVRCP_CT_PAS_VAL_TXT_IND message

When the text for the requested attributes is fully retrieved or the request has been aborted, a CSR_BT_AVRCP_CT_PAS_VAL_TXT_CFM will be sent to the application. The parameters included in the message are listed in Table 40.

Type	Parameter	Description
CsrBtAvrcpPrim	type	Signal identity – always CSR_BT_AVRCP_CT_PAS_VAL_TXT_CFM
CsrUint8	connectionId	The unique connection ID previously received in a CSR_BT_AVRCP_CONNECT_IND/CFM message
CsrBtAvrcpPasAttId_t	attId	The same attribute ID that was specified in the request
CsrUint16	pasDataLen	Refer to the end of the section
CsrUint8	*pasData	Refer to the end of the section
CsrBtResultCode	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.
CsrBtSupplier	resultSupplier	This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h

Table 40: Parameters in the CSR_BT_AVRCP_CT_PAS_VAL_TXT_CFM message

Handling of pasDataLen and *pasData

The contents of the data referred by *pasData is a direct copy of the data included in the response from the target – i.e. no processing has been performed before the data is sent to the application. For this reason the following two helper functions has been provided for convenient parsing of the data:

```
CsrBool CsrBtAvrcpCtLibPasAttribTxtGet(CsrUint16 pasLen, CsrUint8
*pas,    CsrUint16 *index, CsrBtAvrcpPasAttId_t *attId,
CsrBtAvrcpCharSet *charset,   CsrUint8 *attTxtLen, CsrUint8 *attTxt);
```

```
CsrBool CsrBtAvrcpCtLibPasValueTxtGet(CsrUint16 pasLen, CsrUint8
*pas,    CsrUint16 *index, CsrBtAvrcpPasValId *valId,
CsrBtAvrcpCharSet *charset,   CsrUint8 *valTxtLen, CsrUint8 *valTxt);
```

3.11 Player Application Settings (getting Current Values)

In order to enable the controller to retrieve the currently set Player Application Setting values, the interface depicted in Figure 13 shall be used.

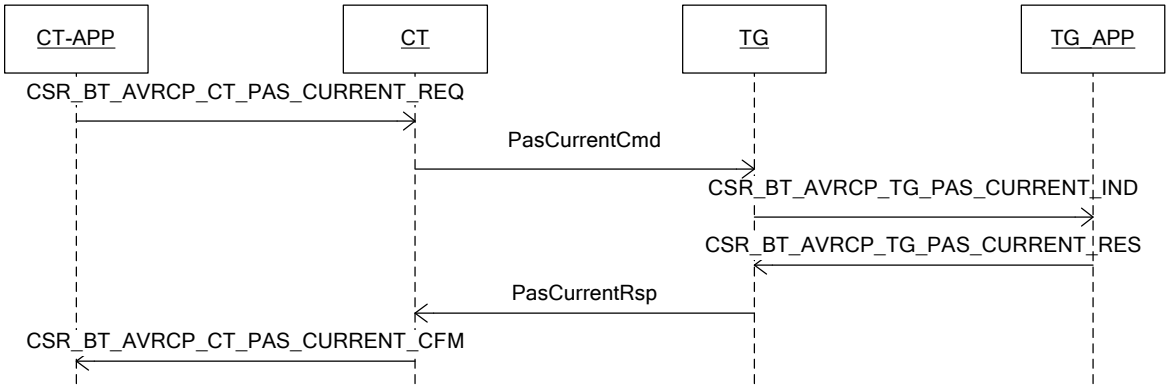


Figure 13: Interface for getting the currently set Player Application Settings

The functions `CsrBtAvrcpCtPasCurrentReqSend` and `CsrBtAvrcpTgPasCurrentResSend` should be used for sending the CSR_BT_AVRCP_CT_PAS_CURRENT_REQ and the CSR_BT_AVRCP_TG_PAS_CURRENT_RES messages respectively.

The prototype for the `CsrBtAvrcpCtPasCurrentReqSend` function is:

```
CsrBtAvrcpCtPasCurrentReqSend (CsrSchedQid phandle, CsrUint8 connId, CsrUint8
attribIdCount, CsrBtAvrcpPasAttId_t *attribId)
```

The arguments for the function are described in Table 41.

Type	Argument	Description
CsrSchedQid	phandle	ID of the task to where the confirmation message should be sent
CsrUint8	connId	The unique connection ID previously received in a CSR_BT_AVRCP_CONNECT_IND/CFM message
CsrUint8	attribIdCount	Number of attributes to retrieve the current value for
CsrBtAvrcpPasAttId_t	*attribId	Pointer to the specified number of attributes

Table 41: Arguments for `CsrBtAvrcpCtPasCurrentReqSend` function

The prototype for the `CsrBtAvrcpTgPasCurrentResSend` function is:

```
CsrBtAvrcpTgPasCurrentResSend (CsrUint8 connId, CsrUint32 msgId, CsrUint8
pasCount, CsrBtAvrcpPasAttValPair *pas, CsrBtAvrcpStatus status)
```

The arguments for the function are described in Table 42.

Type	Argument	Description
`CsrUint8`	`connId`	The unique connection ID previously received in a CSR_BT_AVRCP_CONNECT_IND/CFM message
`CsrUint32`	`playerId`	Unique ID of the media player the message is intended for

Type	Argument	Description
CsrUint32	msgId	Unique message ID for associating indications and responses – use the ID from an indication in the matching response
CsrUint8	pasCount	Number of attribute/value pairs
CsrBtAvrcpPasAttValPair	*pas	Pointer to number of attribute/value pairs of the following type: typedef struct { CsrBtAvrcpPasAttId_t attribId; CsrBtAvrcpPasValId valueId; } CsrBtAvrcpPasAttValPair;
CsrBtAvrcpStatus	status	Indicates whether the operation is accepted or rejected by means of the defines prefixed with AVRCP_STATUS_ in csr_bt_avrcp_prim.h

Table 42: Arguments for CsrBtAvrcpTgPasCurrentResSend function

When a remote controller request the current PAS values from a target, a CSR_BT_AVRCP_TG_PAS_CURRENT_IND message will be sent to the currently addressed media player. The contents of the message are listed in Table 43.

Type	Parameter	Description
CsrBtAvrcpPrim	type	Signal identity – always CSR_BT_AVRCP_TG_PAS_CURRENT_IND
CsrUint8	connectionId	The unique connection ID previously received in a CSR_BT_AVRCP_CONNECT_IND/CFM message
CsrUint32	playerId	Unique ID of the media player the message is intended for
CsrUint32	msgId	Unique message ID for associating indications and responses – use the ID from an indication in the matching response
CsrUint8	attIdCount	Number of attributes to return the current value for
CsrBtAvrcpPasAttId_t	*attId	Pointer to the specified number of attribute IDs

Table 43: Parameters in a CSR_BT_AVRCP_TG_PAS_CURRENT_IND message

When the procedure of retrieving the current PAS values is complete, a CSR_BT_AVRCP_CT_PAS_CURRENT_CFM message will be sent to the controller application. The contents of the message are listed in Table 44.

Type	Parameter	Description
CsrBtAvrcpPrim	type	Signal identity – always CSR_BT_AVRCP_CT_PAS_CURRENT_CFM
CsrUint8	connectionId	The unique connection ID previously received in a CSR_BT_AVRCP_CONNECT_IND/CFM message
CsrUint8	attValPairCount	Number of attribute/value pairs
CsrBtAvrcpPasAttValPair	*attValPair	Pointer to number of attribute/value pairs
CsrBtResultCode	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.
CsrBtSupplier	resultSupplier	This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h

Table 44: Parameters in a CSR_BT_AVRCP_CT_PAS_CURRENT_CFM message

3.12 Player Application Settings (setting current values)

A controller can request a target to change the values of specified attributes. Similarly, a locally initiated change on a target can result in a notification to be sent to a controller. The interfaces for performing these actions are illustrated in Figure 14.

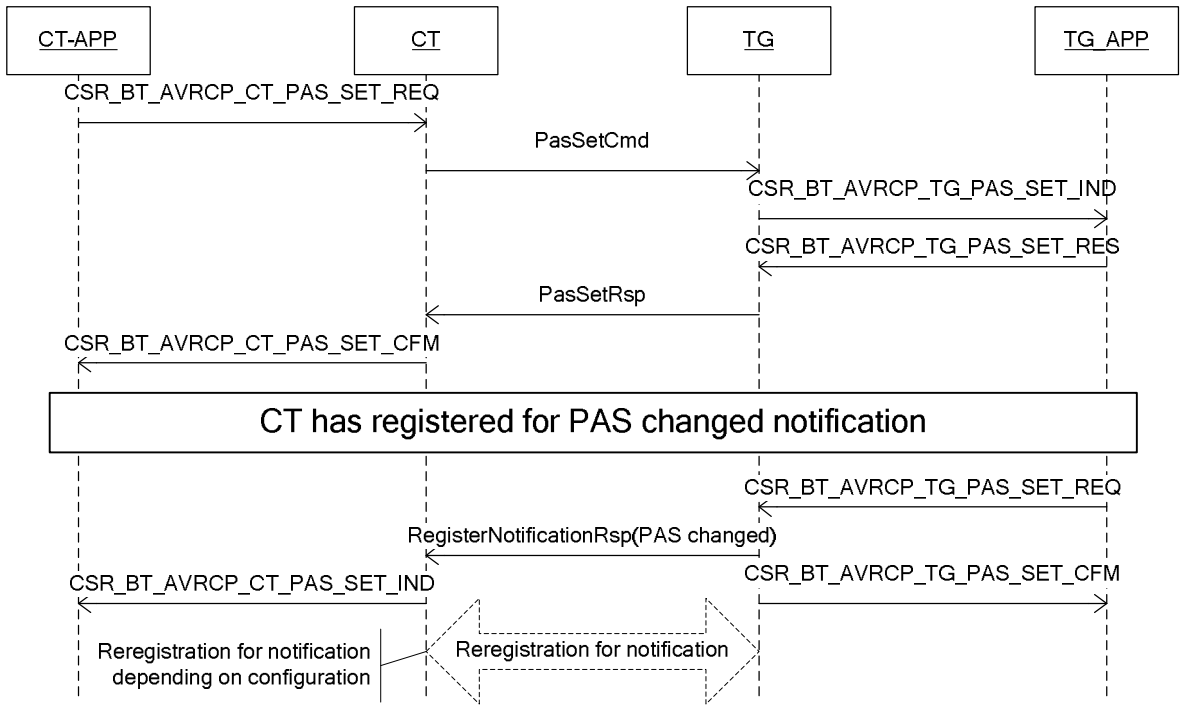


Figure 14: Interface for setting the Player Application Settings

Prototypes for the two functions for sending the CSR_BT_AVRCP_CT_PAS_SET_REQ and CSR_BT_AVRCP_TG_PAS_SET_REQ messages are:

```
CsrBtAvrcpCtPasSetReqSend(CsrSchedQid phandle, CsrUint8 connId,
CsrUint8 attValPairCount, CsrBtAvrcpPasAttValPair *attValPair)

CsrBtAvrcpTgPasSetReqSend(CsrSchedQid phandle, CsrUint32 playerId,
CsrUint8 changedPasCount, CsrBtAvrcpPasAttValPair *changedPas)
```

The arguments for the functions are described in Table 45.

Type	Argument	Description
CsrSchedQid	phandle	ID of the task to where the confirmation message should be sent
CsrUint8	connectionId	The unique connection ID previously received in a CSR_BT_AVRCP_CONNECT_IND/CFM message
CsrUint32	playerId	Unique ID of the media player the message is intended for
CsrUint8	attValPairCount	Number of attribute/value pairs
CsrBtAvrcpPasAttValPair	*attValPair	Pointer to number of attribute/value pairs

Table 45: Arguments for the `CsrBtAvrcpCtPasSetReqSend` and `CsrBtAvrcpTgPasSetReqSend` functions

For the target to send a CSR_BT_AVRCP_TG_PAS_SET_RES in order to reply to a CSR_BT_AVRCP_TG_PAS_SET_IND, the function `CsrBtAvrcpTgPasSetResSend` can be used:

```
CsrBtAvrcpTgPasSetResSend(CsrUint8 connectionId, CsrUint32 playerId,
CsrUint32  msgId, CsrBtAvrcpStatus status)
```

The arguments for the function are described in Table 46.

Type	Argument	Description
CsrUint8	connectionId	The unique connection ID previously received in a CSR_BT_AVRCP_CONNECT_IND/CFM message
CsrUint32	msgId	Unique message ID for associating indications and responses – use the ID from an indication in the matching response
CsrBtAvrcpStatus	status	Indicates whether the operation is accepted or rejected by means of the defines prefixed with CSR_BT_AVRCP_STATUS_ in csr_bt_avrcp_prim.h

Table 46: Arguments for `CsrBtAvrcpTgPasSetResSend` function

The contents of the CSR_BT_AVRCP_TG_PAS_SET_IND, CSR_BT_AVRCP_CT_PAS_SET_IND and CSR_BT_AVRCP_CT_PAS_SET_CFM messages are listed in Table 47, Table 48 and Table 49 respectively.

Type	Parameter	Description
CsrBtAvrcpPrim	type	Signal identity – always CSR_BT_AVRCP_TG_PAS_SET_IND
CsrUint8	connectionId	The unique connection ID previously received in a CSR_BT_AVRCP_CONNECT_IND/CFM message
CsrUint32	playerId	Unique ID of the media player the message is intended for
CsrUint32	msgId	Indicates whether the operation is accepted or rejected by means of the defines prefixed with CSR_BT_AVRCP_STATUS_ in csr_bt_avrcp_prim.h
CsrUint8	attValPairCount	Number of attribute/value pairs
CsrBtAvrcpPasAttValPair	*attValPair	Pointer to number of attribute/value pairs

Table 47: Parameters in a CSR_BT_AVRCP_TG_PAS_SET_IND message

Type	Parameter	Description
CsrBtAvrcpPrim	type	Signal identity – always CSR_BT_AVRCP_CT_PAS_SET_IND
CsrUint8	connectionId	The unique connection ID previously received in a CSR_BT_AVRCP_CONNECT_IND/CFM message
CsrUint8	attValPairCount	Number of attribute/value pairs
CsrBtAvrcpPasAttValPair	*attValPair	Pointer to number of attribute/value pairs

Table 48: Parameters in a CSR_BT_AVRCP_CT_PAS_SET_IND message

Type	Parameter	Description
CsrBtAvrcpPrim	type	Signal identity – always CSR_BT_AVRCP_CT_PAS_SET_CFM
CsrUint8	connectionId	The unique connection ID previously received in a CSR_BT_AVRCP_CONNECT_IND/CFM message
CsrBtResultCode	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.
CsrBtSupplier	resultSupplier	This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h

Table 49: Parameters in a CSR_BT_AVRCP_CT_PAS_SET_CFM message

3.13 Addressed Player

The term “Addressed Player” specifies which media player on a target certain commands should be sent to.

Figure 15 gives an overview of the interfaces provided for changing the addressed media player.

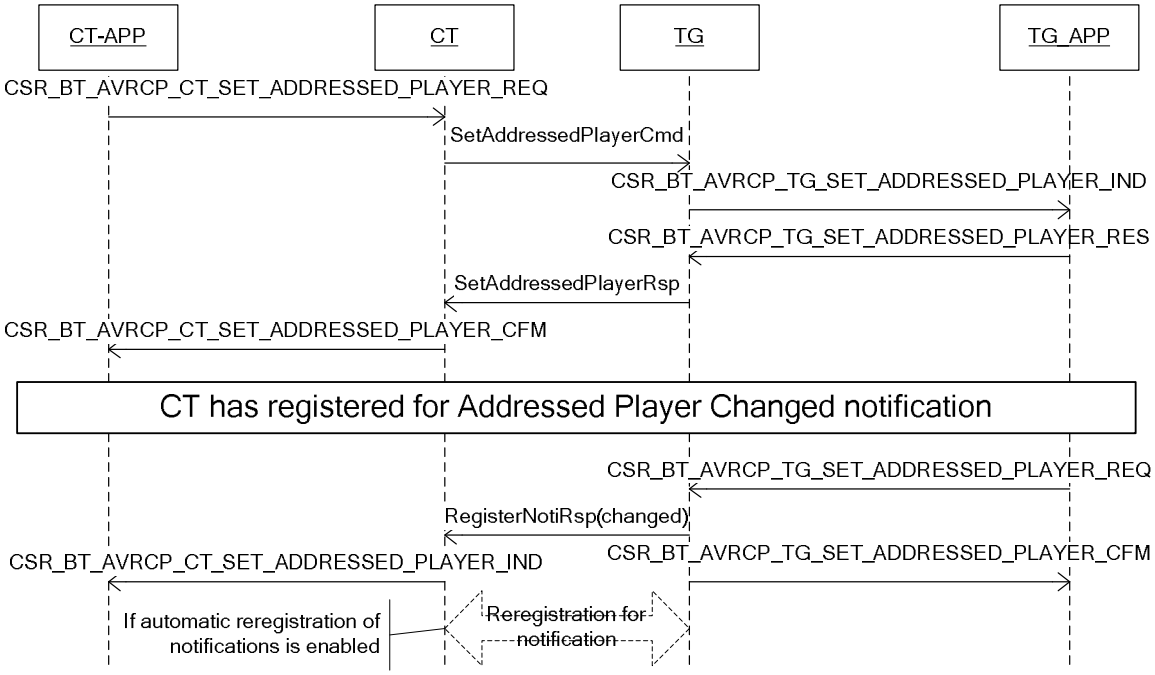


Figure 15: Changing the addressed media player

The addressed player can be changed from both the controller and the target side.It is highly recommended for a controller to register for the Addressed Player Changed notification in order to be notified when the addressed player is changed by the target.

The three downstream messages can be sent with the functions specified below

```
CsrBtAvrcpCtSetAddressedPlayerReqSend(CsrSchedQid phandle, CsrUint8 connId,
CsrUint32 playerId)

CsrBtAvrcpTgSetAddressedPlayerReqSend(CsrSchedQid phandle, CsrUint32 playerId,
CsrUint16 uidCounter)

CsrBtAvrcpTgSetAddressedPlayerResSend (CsrUint8 connId, CsrUint32 playerId,
CsrUint16 uidCounter, CsrUint32 msgId, CsrBtAvrcpStatus status)
```

Note `CsrBtAvrcpTgSetAddressedPlayerReqSend` will have no effect if called while no connection is established. I.e. when a new connection is established, the default media player at that time will become the addressed media player for that connection.

The arguments for the function are described in Table 50.

Type	Argument	Description
CsrSchedQid	phandle	ID of the task to where the confirmation message should be sent
CsrUint8	connId	The unique connection ID previously received in a CSR_BT_AVRCP_CONNECT_IND/CFM message
CsrUint32	playerId	Unique ID of the media player the controller requests to set as the addressed player
CsrUint16	uidCounter	The current UID counter – refer to section 2.3.2 for details
CsrUint32	msgId	Unique message ID for associating indications and responses – use the ID from an indication in the matching response

Type	Argument	Description
CsrBtAvrcpStatus	status	Indicates whether the operation is accepted or rejected by means of the defines prefixed with CSR_BT_AVRCP_STATUS_ in csr_bt_avrcp_prim.h

Table 50: Arguments for the CsrBtAvrcpCtSetAddressedPlayerReqSend, CsrBtAvrcpTgSetAddressedPlayerReqSend and CsrBtAvrcpTgSetAddressedPlayerResSend functions

When a request to change the addressed player is received by a target the CSR_BT_AVRCP_TG_SET_ADDRESSED_PLAYER_IND message, with the parameters in Table 51, is sent to the media player.

Type	Parameter	Description
CsrBtAvrcpPrim	Type	Signal identity – always CSR_BT_AVRCP_TG_SET_ADDRESSED_PLAYER_IND
CsrUint8	connectionId	The unique connection ID previously received in a CSR_BT_AVRCP_CONNECT_IND/CFM message
CsrUint32	playerId	Unique ID of the media player the controller requests to set as the addressed player
CsrUint32	msgId	Unique message ID for associating indications and responses – use the ID from an indication in the matching response

Table 51: Parameters in CSR_BT_AVRCP_TG_SET_ADDRESSED_PLAYER_IND message

When CSR_BT_AVRCP_CT_SET_ADDRESSED_PLAYER_REQ and CSR_BT_AVRCP_TG_SET_ADDRESSED_PLAYER_REQ have been processed, confirmation messages with the parameters listed in respectively Table 52 and Table 53 will be sent to the application.

Type	Parameter	Description
CsrBtAvrcpPrim	type	Signal identity – always CSR_BT_AVRCP_CT_SET_ADDRESSED_PLAYER_CFM
CsrUint8	connectionId	The unique connection ID previously received in a CSR_BT_AVRCP_CONNECT_IND/CFM message
CsrUint32	playerId	Unique ID of the media player the controller requests to set as the addressed player
CsrBtResultCode	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.
CsrBtSupplier	resultSupplier	This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h

Table 52: Parameters in CSR_BT_AVRCP_CT_SET_ADDRESSED_PLAYER_CFM message

Type	Parameter	Description
CsrBtAvrcpPrim	type	Signal identity – always CSR_BT_AVRCP_TG_SET_ADDRESSED_PLAYER_CFM
CsrUint32	playerId	Unique ID of the media player the controller requests to set as the addressed player
CsrBtResultCode	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.
CsrBtSupplier	resultSupplier	This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h

Table 53: Parameters in a CSR_BT_AVRCP_TG_SET_ADDRESSED_PLAYER_CFM message

If a controller had registered for the Addressed Player Changed notification, a CSR_BT_AVRCP_CT_SET_ADDRESSED_PLAYER_IND message will be sent to the application if the addressed player is changed locally on the remote target. The contents of the message are listed in Table 54.

Type	Parameter	Description
CsrBtAvrcpPrim	type	Signal identity – always CSR_BT_AVRCP_CT_SET_ADDRESSED_PLAYER_IND
CsrUint8	connectionId	The unique connection ID previously received in a CSR_BT_AVRCP_CONNECT_IND/CFM message
CsrUint32	playerId	Unique ID of the media player which has been set as the addressed player locally on the target
CsrUint16	uidCounter	The current UID counter – refer to section 2.3.2 for details

Table 54: Parameters in CSR_BT_AVRCP_CT_SET_ADDRESSED_PLAYER_IND message

3.14 Browsed Player

The term “Browsed Player” specifies which media player on a target shall receive browsing commands issued from the controller.

Figure 15 gives an overview of the interfaces provided for changing the browsed media player.

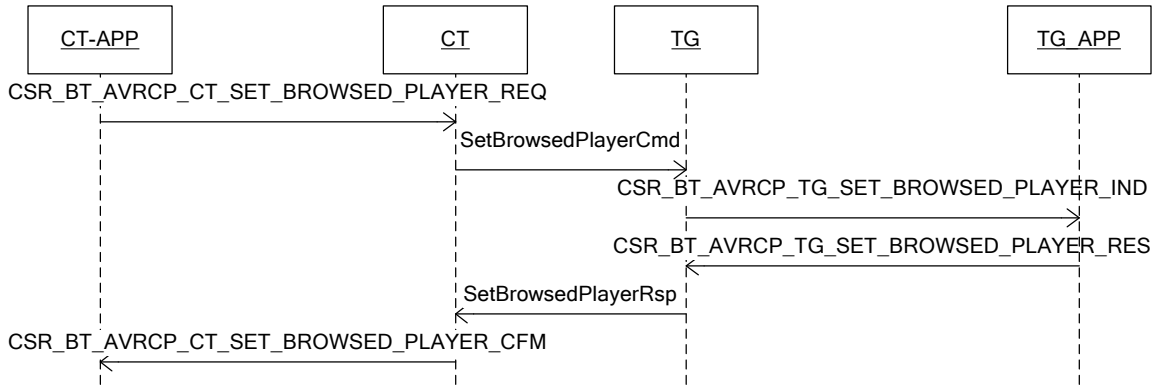


Figure 16: Setting the Browsed Player

The controller will not be notified if a different player is browsed locally on the target, since this occurs out of context to AVRCP.

To send the CSR_BT_AVRCP_CT_SET_BROWSED_PLAYER_REQ message to the profile, the application shall use the function `CsrBtAvrcpCtSetBrowsedPlayerReqSend`:

```
CsrBtAvrcpCtSetBrowsedPlayerReqSend (CsrSchedQid phandle, CsrUint8 connId,
CsrUint32 playerId)
```

The arguments for the function are described in Table 55.

Type	Argument	Description
`CsrSchedQid`	`phandle`	ID of the task to where the confirmation message should be sent
`CsrUint8`	`connId`	The unique connection ID previously received in a CSR_BT_AVRCP_CONNECT_IND/CFM message
`CsrUint32`	`playerId`	Unique ID of the media player the controller requests to set as the browsed player

Table 55: Arguments for `CsrBtAvrcpCtSetBrowsedPlayerReqSend` function

To notify the target that a controller requests to set the browsed player, a CSR_BT_AVRCP_TG_SET_BROWSED_PLAYER_IND message is sent to the media player. The contents of the message are listed in Table 56:

Type	Parameter	Description
CsrBtAvrcpPrim	type	Signal identity – always CSR_BT_AVRCP_TG_SET_BROWSED_PLAYER_IND
CsrUint8	connectionId	The unique connection ID previously received in a CSR_BT_AVRCP_CONNECT_IND/CFM message
CsrUint32	playerId	Unique ID of the media player the controller requests to be set as the browsed player
CsrUint32	msgId	Unique message ID for associating indications and responses – use the ID from an indication in the matching response

Table 56: Parameters in a CSR_BT_AVRCP_TG_SET_BROWSED_PLAYER_IND message

The target must reply to the CSR_BT_AVRCP_TG_SET_BROWSED_PLAYER_IND by sending a CSR_BT_AVRCP_TG_SET_BROWSED_PLAYER_RES to the profile. This is done with the function `CsrBtAvrcpTgSetBrowsedPlayerResSend`:

```
CsrBtAvrcpTgSetBrowsedPlayerResSend (CsrUint8 connId, CsrUint32 playerId,
CsrUint16 uidCounter, CsrUint32 itemsCount, CsrUint8 folderDepth, CsrUint16
folderNamesLen, CsrUint8 *folderNames, CsrUint32 msgId, CsrBtAvrcpStatus status)
```

The arguments for the function are described in Table 57.

Type	Argument	Description
CsrUint8	connId	The unique connection ID previously received in a CSR_BT_AVRCP_CONNECT_IND/CFM message
CsrUint32	playerId	Unique ID of the media player the controller requests to set as the browsed player
CsrUint16	uidCounter	The current UID counter – refer to section 2.3.2 for details
CsrUint32	itemsCount	Number of items in the current folder of the browsed player
CsrUint8	folderDepth	Depth of the current folder
CsrUint16	folderNamesLen	Number of bytes allocated in the “folderNames” field below
CsrUint8	*folderNames	Directory path from the root directory to the current directory, separated by either the ‘\’ or the ‘/’ symbol.
CsrUint32	msgId	Unique message ID for associating indications and responses – use the ID from an indication in the matching response
CsrBtAvrcpStatus	status	Indicates whether the operation is accepted or rejected by means of the defines prefixed with `AVRCP_STATUS_` in csr_bt_avrcp_prim.h

Table 57: Arguments for `CsrBtAvrcpTgSetBrowsedPlayerResSend` function

When the CSR_BT_AVRCP_CT_SET_BROWSED_PLAYER_REQ request has been fully processed on the controller a CSR_BT_AVRCP_CT_SET_BROWSED_PLAYER_CFM is sent to the application. Table 58 lists the contents of the message.

Type	Parameter	Description
CsrBtAvrcpPrim	type	Signal identity – always CSR_BT_AVRCP_CT_SET_BROWSED_PLAYER_CFM
CsrUint8	connectionId	The unique connection ID previously received in a CSR_BT_AVRCP_CONNECT_IND/CFM message
CsrUint32	playerId	The same media player ID that was specified in the request
CsrUint16	uidCounter	The current UID counter – refer to section 2.3.2 for details
CsrUint32	itemsCount	Number of items in the current folder of the browsed player

Type	Parameter	Description
CsrUint8	folderDepth	Depth of the current folder
CsrUint16	folderNamesLen	Number of bytes allocated in the “folderNames” field below
CsrUint8	*folderNames	Pointer to a NULL terminated string containing the full directory path received from the remote device.
CsrBtResultCode	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.
CsrBtSupplier	resultSupplier	This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h

Table 58: Parameters in a CSR_BT_AVRCP_CT_SET_BROWSED_PLAYER_CFM message

3.15 Play Item

The controller can request to play a specific media item on a remote controller by using the interfaces illustrated in Figure 17.

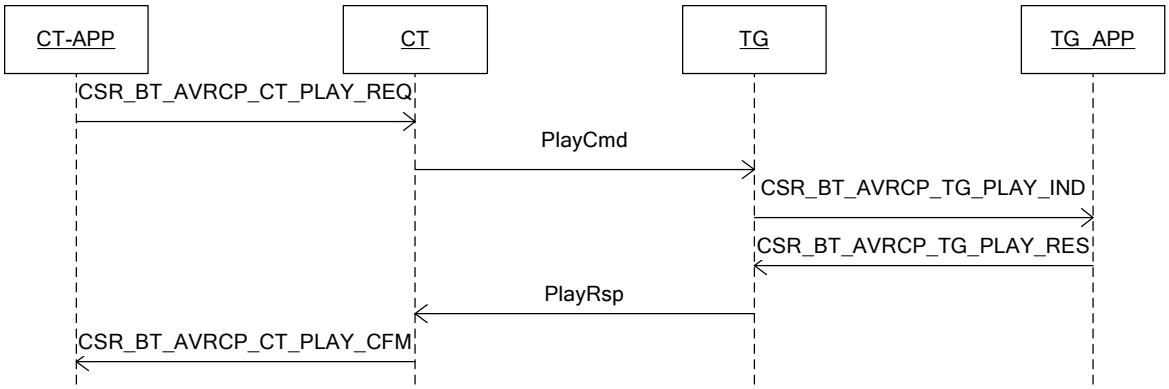


Figure 17: Play Item interfaces

To send the CSR_BT_AVRCP_CT_PLAY_REQ message the following function must be used:

```
CsrBtAvrcpCtPlayReqSend(CsrSchedQid phandle, CsrUint8 connId,
CsrBtAvrcpScope scope, CsrUint16 uidCounter, CsrBtAvrcpUid uid)
```

The arguments for the function are described in Table 59.

Type	Argument	Description
CsrSchedQid	phandle	ID of the task to where the confirmation message should be sent
CsrUint8	connId	The unique connection ID previously received in a CSR_BT_AVRCP_CONNECT_IND/CFM message
CsrBtAvrcpScope	scope	Scope as defined in section 2.3.1
CsrUint16	uidCounter	The current UID counter – refer to section 2.3.2 for details
CsrBtAvrcpUid	uid	The UID of the item – refer to section 2.3.2 for details

Table 59: Arguments for `CsrBtAvrcpCtPlayReqSend` function

On the target, a CSR_BT_AVRCP_TG_PLAY_IND message will be sent to the addressed media player. The contents of the message are listed in Table 60.

Type	Parameter	Description
CsrBtAvrcpPrim	type	Signal identity – always CSR_BT_AVRCP_TG_PLAY_IND
CsrUint8	connectionId	The unique connection ID previously received in a CSR_BT_AVRCP_CONNECT_IND/CFM message
CsrUint32	playerId	Unique ID of the media player the message is intended for
CsrBtAvrcpScope	scope	Scope as defined in section 2.3.1
CsrUint16	uidCounter	The current UID counter – refer to section 2.3.2 for details
CsrBtAvrcpUid	uid	The UID of the item – refer to section 2.3.2 for details
CsrUint32	msgId	Unique message ID for associating indications and responses – use the ID from an indication in the matching response

Table 60: Parameters in a CSR_BT_AVRCP_TG_PLAY_IND message

The target must reply to the CSR_BT_AVRCP_TG_PLAY_IND message by sending a CSR_BT_AVRCP_TG_PLAY_RES by means of the following function:

```
CsrBtAvrcpTgPlayResSend(CsrUint8 connectionId, CsrBtAvrcpUid uid,
CsrBtAvrcpScope scope, CsrUint32 msgId, CsrBtAvrcpStatus status)
```

The arguments for the function are described in Table 61.

Type	Argument	Description
CsrUint8	connectionId	The unique connection ID previously received in a CSR_BT_AVRCP_CONNECT_IND/CFM message
CsrBtAvrcpUid	uid	The UID of the item – refer to section 2.3.2 for details
CsrBtAvrcpScope	scope	Scope as defined in section 2.3.1
CsrUint32	msgId	Unique message ID for associating indications and responses – use the ID from an indication in the matching response
CsrBtAvrcpStatus	status	Indicates whether the operation is accepted or rejected by means of the defines prefixed with `AVRCP_STATUS_` in csr_bt_avrcp_prim.h

Table 61: Arguments for `CsrBtAvrcpTgPlayResSend` function

At the completion of the CSR_BT_AVRCP_CT_PLAY_REQ request on the controller, the CSR_BT_AVRCP_CT_PLAY_CFM message, with the contents listed in Table 62, will be sent to the application.

Type	Parameter	Description
CsrBtAvrcpPrim	type	Signal identity – always CSR_BT_AVRCP_CT_PLAY_CFM
CsrUint8	connectionId	The unique connection ID previously received in a CSR_BT_AVRCP_CONNECT_IND/CFM message
CsrBtAvrcpUid	uid	The UID of the item – refer to section 2.3.2 for details
CsrBtAvrcpScope	scope	Scope as defined in section 2.3.1
CsrBtResultCode	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.
CsrBtSupplier	resultSupplier	This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h

Table 62: Parameters in a CSR_BT_AVRCP_CT_PLAY_CFM message

3.16 Search

In order for a controller to generate a list of search results from a target, the interfaces in Figure 18 can be used.

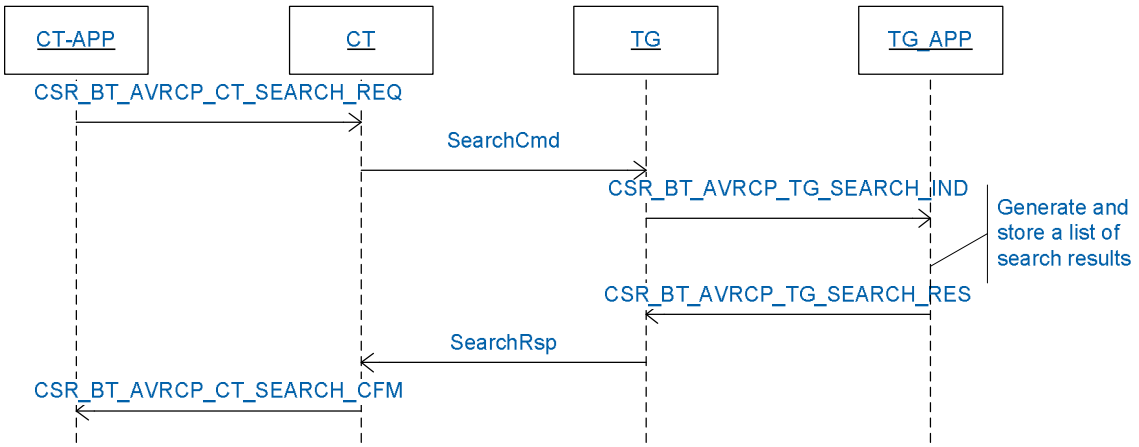


Figure 18: Search interface

To send the CSR_BT_AVRCP_CT_SEARCH_REQ from the controller application, the following function should be used:

```
CsrBtAvrcpCtSearchReqSend(CsrSchedQid phandle, CsrUint8 connId,
CsrUtf8String *text)
```

The arguments for the function are described in Table 63.

Type	Argument	Description
CsrSchedQid	phandle	ID of the task to where the confirmation message should be sent
CsrUint8	connId	The unique connection ID previously received in a CSR_BT_AVRCP_CONNECT_IND/CFM message
CsrUtf8String	*text	NUL-terminated string to search for

Table 63: Arguments for CsrBtAvrcpCtSearchReqSend function

When a remote controller requests a search, the CSR_BT_AVRCP_TG_SEARCH_IND message is sent to the browsed media player. The parameters in the message are listed in Table 64:

Type	Parameter	Description
CsrBtAvrcpPrim	type	Signal identity – always CSR_BT_AVRCP_TG_SEARCH_IND
CsrUint8	connectionId	The unique connection ID previously received in a CSR_BT_AVRCP_CONNECT_IND/CFM message
CsrUint32	playerId	Unique ID of the media player the message is intended for
CsrUtf8String	*text	NUL-terminated string to search for
CsrUint32	msgId	Unique message ID for associating indications and responses – use the ID from an indication in the matching response

Table 64: Parameters in a CSR_BT_AVRCP_TG_SEARCH_IND message

The browsed player must reply to the CSR_BT_AVRCP_TG_SEARCH_IND message by sending a CSR_BT_AVRCP_TG_SEARCH_RES message using the following function:

```
CsrBtAvrcpTgSearchResSend (CsrUint8 connId, CsrUint16 uidCounter,
CsrUint32 numberOfItems, CsrUint32 msgId, CsrBtAvrcpStatus status)
```

The arguments for the function are described in Table 65.

Type	Argument	Description
CsrUint8	connId	The unique connection ID previously received in a CSR_BT_AVRCP_CONNECT_IND/CFM message
CsrUint16	uidCounter	The current UID counter – refer to section 2.3.2 for details
CsrUint32	numberOfItems	Number of items in the search result
CsrUint32	msgId	Unique message ID for associating indications and responses – use the ID from an indication in the matching response
CsrBtAvrcpStatus	status	Indicates whether the operation is accepted or rejected by means of the defines prefixed with `AVRCP_STATUS_` in csr_bt_avrcp_prim.h

Table 65: Arguments for `CsrBtAvrcpTgSearchResSend` function

At the completion of the search procedure the CSR_BT_AVRCP_CT_SEARCH_CFM message will be sent to the controller application. The parameters in the message are listed in Table 66:

Type	Parameter	Description
CsrBtAvrcpPrim	type	Signal identity – always CSR_BT_AVRCP_CT_SEARCH_CFM
CsrUint8	connectionId	The unique connection ID previously received in a CSR_BT_AVRCP_CONNECT_IND/CFM message
CsrUint16	uidCounter	The current UID counter – refer to section 2.3.2 for details
CsrUint32	numberOfItems	Number of items in the search result
CsrBtResultCode	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.
CsrBtSupplier	resultSupplier	This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h

Table 66: Parameters in a CSR_BT_AVRCP_CT_SEARCH_CFM message

3.17 Change Path

The interfaces in Figure 19 can be used for changing the current path used for other browsing commands.

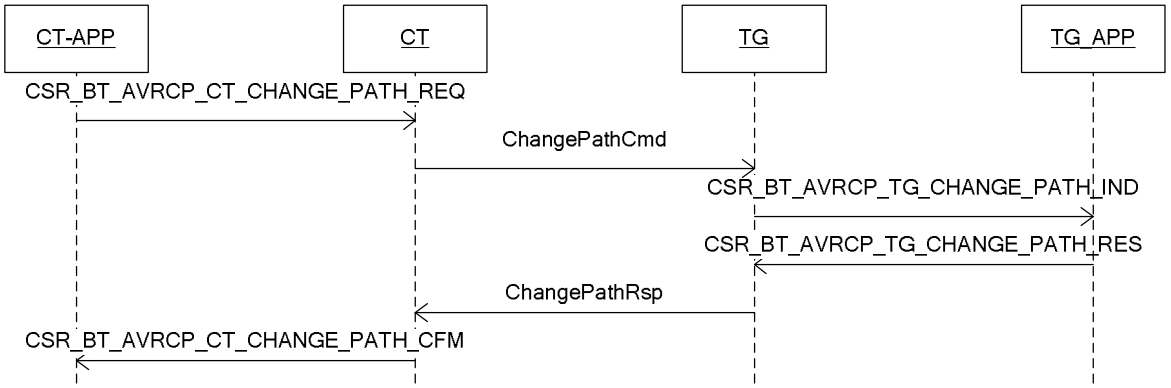


Figure 19: Change Path interface

For a controller to initiate a change of path the CSR_BT_AVRCP_CT_CHANGE_PATH_REQ message must be sent to the profile by means of the following function:

```
CsrBtAvrcpCtChangePathReqSend (CsrSchedQid phandle, CsrUint8 connId, CsrUint16
uidCounter, CsrBtAvrcpFolderDirection folderDir, CsrBtAvrcpUid folderUid)
```

The arguments for the function are described in Table 67.

Type	Argument	Description
CsrSchedQid	phandle	ID of the task to where the confirmation message should be sent
CsrUint8	connId	The unique connection ID previously received in a CSR_BT_AVRCP_CONNECT_IND/CFM message
CsrUint16	uidCounter	The current UID counter – refer to section 2.3.2 for details
CsrBtAvrcpFolderDirection	folderDir	Use one of the following values: – CSR_BT_AVRCP_CHANGE_PATH_UP – CSR_BT_AVRCP_CHANGE_PATH_DOWN
CsrBtAvrcpUid	folderUid	The UID of the item – refer to section 2.3.2 for details

Table 67: Arguments for `CsrBtAvrcpCtChangePathReqSend` function

When a target receives a command to change the current path, the CSR_BT_AVRCP_TG_CHANGE_PATH_IND message is sent to the browsed player. The contents of the message are listed in Table 68.

Type	Parameter	Description
CsrBtAvrcpPrim	type	Signal identity – always CSR_BT_AVRCP_TG_CHANGE_PATH_IND
CsrUint8	connectionId	The unique connection ID previously received in a CSR_BT_AVRCP_CONNECT_IND/CFM message
CsrUint32	playerId	Unique ID of the media player the message is intended for
CsrBtAvrcpFolderDirection	folderDir	Will be one of the following values: – CSR_BT_AVRCP_CHANGE_PATH_UP – CSR_BT_AVRCP_CHANGE_PATH_DOWN
CsrBtAvrcpUid	folderUid	The UID of the item – refer to section 2.3.2 for details
CsrUint32	msgId	Unique message ID for associating indications and responses – use the ID from an indication in the matching response

Table 68: Parameters in a CSR_BT_AVRCP_TG_CHANGE_PATH_IND message

The browsed player must reply to the CSR_BT_AVRCP_TG_CHANGE_PATH_IND message by sending a CSR_BT_AVRCP_TG_CHANGE_PATH_RES to the profile. This is done using the following function:

```
CsrBtAvrcpTgChangePathResSend (CsrUint8 connId, CsrUint32 itemsCount, CsrUint32
msgId, CsrBtAvrcpStatus status)
```

The arguments for the function are described in Table 69.

Type	Argument	Description
CsrUint8	connId	The unique connection ID previously received in a CSR_BT_AVRCP_CONNECT_IND/CFM message
CsrUint32	itemsCount	Number of items in the new folder
CsrUint32	msgId	Unique message ID for associating indications and responses – use the ID from an indication in the matching response
CsrBtAvrcpStatus	status	Indicates whether the operation is accepted or rejected by means of the defines prefixed with AVRCP_STATUS_ in csr_bt_avrcp_prim.h

Table 69: Arguments for `CsrBtAvrcpTgChangePathResSend` function

When the change path procedure is complete on the controller, the CSR_BT_AVRCP_TG_CHANGE_PATH_CFM message will be sent to the application. The contents of the message are listed in Table 70:

Type	Parameter	Description
CsrBtAvrcpPrim	type	Signal identity – always CSR_BT_AVRCP_CT_CHANGE_PATH_CFM
CsrUint8	connectionId	The unique connection ID previously received in a CSR_BT_AVRCP_CONNECT_IND/CFM message
CsrUint32	itemsCount	Number of items in the new folder
CsrBtResultCode	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.
CsrBtSupplier	resultSupplier	This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h

Table 70: Parameters in a CSR_BT_AVRCP_CT_CHANGE_PATH_CFM message

3.18 Pass-through and Group Navigation Interface

The interfaces in Figure **20** can be used for sending pass-through commands from a controller and notifying a target media player about incoming commands.

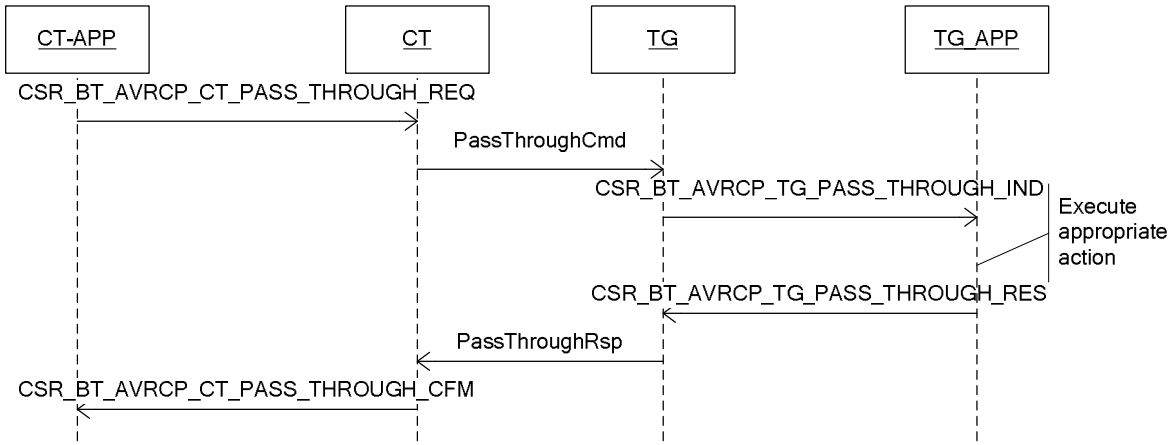


Figure 20: Pass-through and Group Navigation interface

For a controller to send a CSR_BT_AVRCP_PASS_THROUGH message to the profile the following function must be used:

```
CsrBtAvrcpCtPassThroughReqSend (CsrSchedQid phandle, CsrUint8 connId, CsrUint8
opId, CsrUint8 state)
```

The arguments for the function are described in Table 71.

Type	Argument	Description
CsrSchedQid	phandle	ID of the task to where the confirmation message should be sent
CsrUint8	connId	The unique connection ID previously received in a CSR_BT_AVRCP_CONNECT_IND/CFM message
CsrUint8	opId	Refer to the defines prefixed with `AVRCP_PT_OP_ID_` in csr_bt_avrcp_prim.h

Type	Argument	Description
CsrUint8	state	Use one of the following values: – CSR_BT_AVRCP_PT_STATE_PRESS – CSR_BT_AVRCP_PT_STATE_RELEASE – CSR_BT_AVRCP_PT_STATE_PRESS_RELEASE: Send both a press and release command – CSR_BT_AVRCP_PT_STATE_NEXT_GROUP: Send a Next Group command – opId is ignored – CSR_BT_AVRCP_PT_STATE_PREV_GROUP

Table 71: Arguments for `CsrBtAvrcpCtPassThroughReqSend` function

When an incoming pass-through command is received by a target, a CSR_BT_AVRCP_TG_PASS_THROUGH_IND message will be sent to the addressed media player. The contents of the message are listed in Table 72:

Type	Parameter	Description
CsrBtAvrcpPrim	Type	Signal identity – always CSR_BT_AVRCP_TG_PASS_THROUGH_IND
CsrUint8	connectionId	The unique connection ID previously received in a CSR_BT_AVRCP_CONNECT_IND/CFM message
CsrUint32	playerId	Unique ID of the media player the message is intended for
CsrBtAvrcpPTOpId	operationId	Refer to the defines prefixed with CSR_BT_AVRCP_PT_OP_ID_ in csr_bt_avrcp_prim.h
CsrBtAvrcpPTState	state	See Table 71
CsrUint32	msgId	Unique message ID for associating indications and responses – use the ID from an indication in the matching response

Table 72: Parameters in a CSR_BT_AVRCP_TG_PASS_THROUGH_IND message

To reply to the CSR_BT_AVRCP_TG_PASS_THROUGH_IND message the media player must send a CSR_BT_AVRCP_TG_PASS_THROUGH_RES by means of the following function:

```
CsrBtAvrcpTgPassThroughResSend (CsrUint8 connId, CsrUint32 msgId, CsrUint8
status)
```

The arguments for the function are described in Table 73.

Type	Argument	Description
CsrUint8	connId	The unique connection ID previously received in a CSR_BT_AVRCP_CONNECT_IND/CFM message
CsrUint32	msgId	Unique message ID for associating indications and responses – use the ID from an indication in the matching response
CsrUint8	status	Indicates whether the operation is accepted or rejected by means of the defines prefixed with CSR_BT_AVRCP_STATUS_ in csr_bt_avrcp_prim.h

Table 73: Arguments for `CsrBtAvrcpTgPassThroughResSend` function

At the completion of the procedure a CSR_BT_AVRCP_CT_PASS_THROUGH_CFM message will be sent to the controller application. The contents of the message are listed in Table 74:

Type	Parameter	Description
CsrBtAvrcpPrim	type	Signal identity – always CSR_BT_AVRCP_CT_PASS_THROUGH_CFM
CsrUint8	connectionId	The unique connection ID previously received in a CSR_BT_AVRCP_CONNECT_IND/CFM message

Type	Parameter	Description
CsrBtAvrcpPTOpId	operationId	Same operation ID that was included in the request
CsrBtAvrcpPTState	state	Same state that was included in the request
CsrBtResultCode	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.
CsrBtSupplier	resultSupplier	This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h

Table 74: Parameters in a CSR_BT_AVRCP_CT_PASS_THROUGH_CFM message

3.19 Volume Interface

The controller can change the volume of a remote target by using the interfaces in Figure 21.

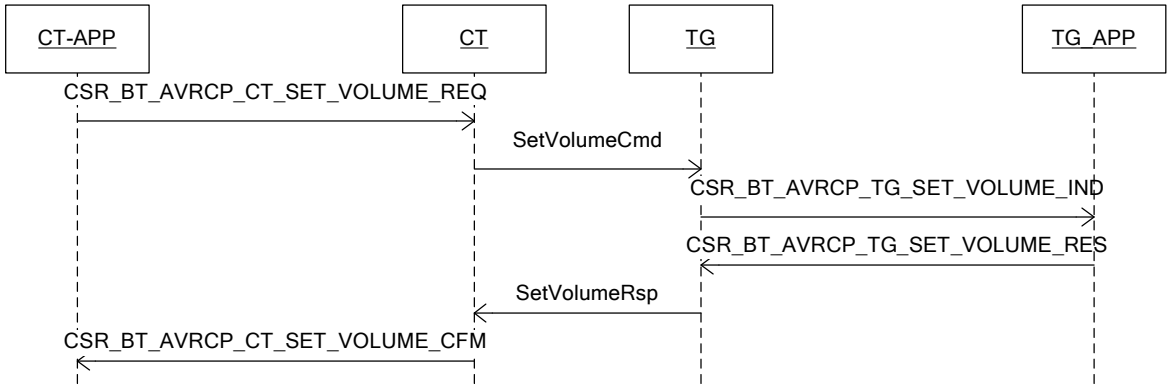


Figure 21: Volume interfaces

For a controller to send the CSR_BT_AVRCP_CT_SET_VOLUME_REQ message, the following function must be used:

```
CsrBtAvrcpCtSetVolumeReqSend (CsrSchedQid phandle, CsrUint8 connId,
CsrUint8 volume)
```

The arguments for the function are described in Table 75.

Type	Argument	Description
CsrSchedQid	phandle	ID of the task to where the confirmation message should be sent
CsrUint8	connId	The unique connection ID previously received in a CSR_BT_AVRCP_CONNECT_IND/CFM message
CsrUint8	volume	Volume level (scale accordingly between the two limits): - 0x00 → 0% - 0x7F → 100%

Table 75: Arguments for `CsrBtAvrcpTgSetVolumeResSend` function

Arguments for `CsrBtAvrcpCtSetVolumeReqSend` function

When a remote controller requests the volume to be changed on a target, a CSR_BT_AVRCP_TG_SET_VOLUME_IND message will be sent to the addressed media player. The contents of the message are listed in Table 76:

Type	Parameter	Description
CsrBtAvrcpPrim	type	Signal identity – always CSR_BT_AVRCP_TG_SET_VOLUME_IND
CsrUint8	connectionId	The unique connection ID previously received in a CSR_BT_AVRCP_CONNECT_IND/CFM message
CsrUint32	playerId	Unique ID of the media player the message is intended for
CsrUint8	volume	Volume level requested by controller
CsrUint32	msgId	Unique message ID for associating indications and responses – use the ID from an indication in the matching response

Table 76: Parameters in a CSR_BT_AVRCP_TG_SET_VOLUME_IND message

The media player must reply to the CSR_BT_AVRCP_TG_SET_VOLUME_IND message by sending a CSR_BT_AVRCP_TG_SET_VOLUME_RES message, which is done using the following function:

```
CsrBtAvrcpTgSetVolumeResSend (CsrUint8 connId, CsrUint8 volume,
CsrUint32 msgId, CsrBtAvrcpStatus status)
```

The arguments for the function are described in Table 77.

Type	Argument	Description
CsrUint8	connId	The unique connection ID previously received in a CSR_BT_AVRCP_CONNECT_IND/CFM message
CsrUint8	volume	Actual volume set
CsrUint32	msgId	Unique message ID for associating indications and responses – use the ID from an indication in the matching response
CsrBtAvrcpStatus	status	Indicates whether the operation is accepted or rejected by means of the defines prefixed with `AVRCP_STATUS_` in csr_bt_avrcp_prim.h

Table 77: Arguments for the `CsrBtAvrcpTgSetVolumeResSend` function

At the completion of the procedure a CSR_BT_AVRCP_CT_SET_VOLUME_CFM message, with the parameters listed in Table 78 is sent to the controller application:

Type	Parameter	Description
CsrBtAvrcpPrim	type	Signal identity – always CSR_BT_AVRCP_CT_SET_VOLUME_CFM
CsrUint8	connectionId	The unique connection ID previously received in a CSR_BT_AVRCP_CONNECT_IND/CFM message
CsrUint8	volume	Actual volume set
CsrBtResultCode	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.
CsrBtSupplier	resultSupplier	This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h

Table 78: Parameters in a CSR_BT_ AVRCP_CT_SET_VOLUME_CFM message

3.20 Get Folder Items Interface

The Get Folder Items functionality can be used by a controller for retrieving two types of information from a target:

- List of media and folder items from the browsed player's:

- Virtual file system
- Now Playing List
- List of search results

- List of media players supported by the target

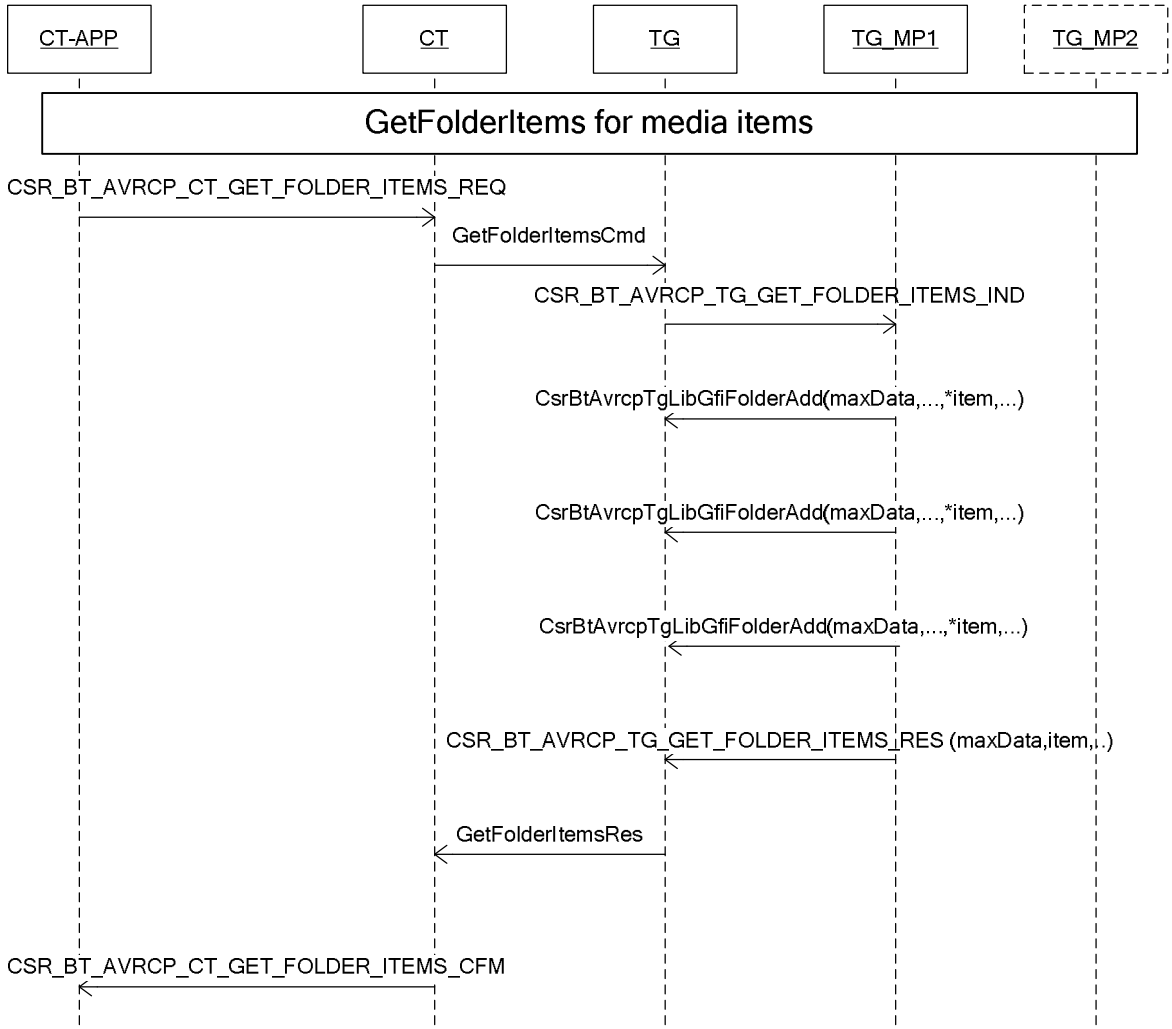


Figure 22: Get Folder Items interfaces

Requests relating to media items will be passed directly from the controller to the browsed media player, since the profile does not hold any information related to media items.

Regarding Get Folder Items for media players, all of the required information is available internally in the profile with the exception of the status of a media player. For this reason, a CSR_BT_AVRCP_TG_GET_PLAY_STATUS_IND is sent to each media player, which must reply according to the description in section 3.22.

In order for a controller application to send the CSR_BT_AVRCP_CT_GET_FOLDER_ITEMS_REQ message to the profile, the following function must be used:

```
CsrBtAvrcpCtGetFolderItemsReqSend (CsrSchedQid phandle, CsrUint8
connId, CsrBtAvrcpScope scope, CsrUint32 startItem, CsrUint32
endItem, CsrUint32 attributeMask)
```

The arguments for the function are described in Table 79.

Type	Argument	Description
CsrSchedQid	phandle	ID of the task to where the confirmation message should be sent
CsrUint8	connId	The unique connection ID previously received in a CSR_BT_AVRCP_CONNECT_IND/CFM message
CsrBtAvrcpScope	scope	Scope as defined in section 2.3.1
CsrUint32	startItem	Index of the first item to get starting from 0
CsrUint32	endItem	Index of the last item to get starting from 0
CsrBtAvrcpItemAttMask	attributeMask	Bitmask combined of the following flags: – CSR_BT_AVRCP_ITEM_ATT_MASK_TITLE – CSR_BT_AVRCP_ITEM_ATT_MASK_ARTIST – CSR_BT_AVRCP_ITEM_ATT_MASK_ALBUM – CSR_BT_AVRCP_ITEM_ATT_MASK_MEDIA_NUMBER – CSR_BT_AVRCP_ITEM_ATT_MASK_TOTAL_NUMBER – CSR_BT_AVRCP_ITEM_ATT_MASK_GENRE – CSR_BT_AVRCP_ITEM_ATT_MASK_TIME To retrieve all available attributes use the following value: – CSR_BT_AVRCP_ITEM_ATT_MASK_ALL If no attributes are requested: – CSR_BT_AVRCP_ITEM_ATT_MASK_NONE

Table 79: Arguments for the `CsrBtAvrcpCtGetFolderItemsReqSend` function

When a remote controller requests the folder list for the Virtual File System, Now Playing List or search results, the CSR_BT_AVRCP_TG_GET_FOLDER_ITEMS_IND message will be sent to the browsed player. The contents of the message are listed in Table 80:

Type	Parameter	Description
CsrBtAvrcpPrim	type	Signal identity – always CSR_BT_AVRCP_TG_GET_FOLDER_ITEMS_IND
CsrUint8	connectionId	The unique connection ID previously received in a CSR_BT_AVRCP_CONNECT_IND/CFM message
CsrUint32	playerId	Unique ID of the media player the message is intended for
CsrBtAvrcpScope	scope	Scope as defined in section 2.3.1
CsrUint32	startItem	Index of the first item to get starting from 0
CsrUint32	endItem	Index of the last item to get starting from 0
CsrBtAvrcpItemAttMask	attributeMask	Refer to Table 79
CsrUint32	maxData	Maximum amount of data in bytes, that the controller can handle when returning the response.
CsrUint32	msgId	Unique message ID for associating indications and responses – use the ID from an indication in the matching response

Table 80: Parameters in a CSR_BT_AVRCP_TG_GET_FOLDER_ITEMS_IND message

The controller must reply to the CSR_BT_AVRCP_TG_GET_FOLDER_ITEMS_IND message by sending a CSR_BT_AVRCP_TG_GET_FOLDER_ITEMS_RES message with the following function:

```
CsrBtAvrcpTgGetFolderItemsResSend (CsrUint8 connId, CsrUint16
itemCount, CsrUint16 uidCounter, CsrUint16 itemsLen, CsrUint8 *items,
CsrUint32 msgId, CsrBtAvrcpStatus status)
```

The arguments for the function are described in Table 81.

Type	Argument	Description
CsrUint8	connId	The unique connection ID previously received in a CSR_BT_AVRCP_CONNECT_IND/CFM message
CsrUint16	itemCount	Number of media items in the response message
CsrUint16	uidCounter	The current UID counter – refer to section 2.3.2 for details
CsrUint16	itemsLen	Length of the media items data in bytes. The length must not exceed the value given by maxData, see Table 80. If all media items can not fit into one response, as many complete media items as possible should be included in the response.
CsrUint8	*items	Pointer to the media items data
CsrUint32	msgId	Unique message ID for associating indications and responses – use the ID from an indication in the matching response
CsrBtAvrcpStatus	status	Indicates whether the operation is accepted or rejected by means of the defines prefixed with CSR_BT_AVRCP_STATUS_ in csr_bt_avrcp_prim.h

Table 81: Arguments for `CsrBtAvrcpTgGetFolderItemsResSend` function

At the completion of the Get Folder Items procedure, a CSR_BT_AVRCP_CT_GET_FOLDER_ITEMS_CFM message will be sent to the controller application that initiated the procedure. The contents of the message are listed in Table 82:

Type	Parameter	Description
CsrBtAvrcpPrim	Type	Signal identity – always CSR_BT_AVRCP_CT_GET_FOLDER_ITEMS_CFM
CsrUint8	connectionId	The unique connection ID previously received in a CSR_BT_AVRCP_CONNECT_IND/CFM message
CsrBtAvrcpScope	Scope	Scope as defined in section 2.3.1
CsrUint16	uidCounter	The current UID counter – refer to section 2.3.2 for details
CsrUint32	startItem	Index of the first item to get starting from 0
CsrUint32	endItem	Index of the last item to get starting from 0
CsrUint16	itemsCount	Number of media items in the confirm message
CsrUint16	itemsDataLen	Length of the media items data in bytes
CsrUint8	*itemsData	Pointer to the media items data
CsrBtResultCode	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.
CsrBtSupplier	resultSupplier	This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h

Table 82: Parameters in a CSR_BT_AVRCP_CT_GET_FOLDER_ITEMS_CFM message

3.21 Get Attributes Interface

In order for a controller to retrieve metadata for a specific media item, the interfaces in Figure 23 can be used.

Figure 23: Get Attributes interface

If the AVRCP versions of both the target and controller roles are 1.4 or newer, the GetItemAttributes AVRCP command/response will be used. If the version of one or both roles is 1.3, the GetElementAttributes AVRCP command/response will be used instead.

The main differences between the two commands are the following:

- The GetElementAttributes response is sent over the control channel and can be fragmented using AV/C
- The GetElementAttributes functionality can only be used for the currently playing media item

Due to the possibility of AV/C fragmentation when using the GetElementAttributes, the CSR_BT_AVRCP_CT_GET_ATTRIBUTES_IND message will be sent to the controller if multiple fragments are available. On the target, AV/C fragmentation is handled internally in the profile.

A controller can initiate the procedure by sending the CSR_BT_AVRCP_CT_GET_ATTRIBUTES_REQ message to the profile by using the following function:

```
CsrBtAvrcpCtGetAttributesReqSend (CsrSchedQid phandle, CsrUint8 connId,
CsrBtAvrcpScope scope, CsrBtAvrcpUid uid, CsrUint16 uidCounter,
CsrBtAvrcpItemAttMask attributeMask)
```

The arguments for the function are described in Table 83:

Type	Argument	Description
CsrSchedQid	phandle	ID of the task to where the confirmation message should be sent
CsrUint8	connId	The unique connection ID previously received in a CSR_BT_AVRCP_CONNECT_IND/CFM message
CsrBtAvrcpScope	scope	Scope as defined in section 2.3.1
CsrBtAvrcpUid	uid	The UID of the item – refer to section 2.3.2 for details
CsrUint16	uidCounter	The current UID counter – refer to section 2.3.2 for details
CsrBtAvrcpItemAttMask	attributeMask	Refer to Table 79

Table 83: Arguments for `CsrBtAvrcpCtGetAttributesReqSend` function

When a remote controller requests the target for attributes for a media item, the CSR_BT_AVRCP_TG_GET_ATTRIBUTES_IND message will be sent to the addressed player if using GetElementAttributes or the browsed player if using GetItemAttributes. The contents of the message are listed in Table 84:

Type	Parameter	Description
CsrBtAvrcpPrim	Type	Signal identity – always CSR_BT_AVRCP_TG_GET_ATTRIBUTES_IND
CsrUint8	connectionId	The unique connection ID previously received in a CSR_BT_AVRCP_CONNECT_IND/CFM message
CsrUint32	playerId	Unique ID of the media player the message is intended for
CsrBtAvrcpUid	Uid	The UID of the item – refer to section 2.3.2 for details
CsrBtAvrcpScope	Scope	Scope as defined in section 2.3.1
CsrBtAvrcpItemAttMask	attributeMask	Refer to Table 79
CsrUint32	maxData	Maximum number of data bytes allowed to return in the corresponding response message, i.e. when calling the CsrBtAvrcpTgGetAttributesResSend function (refer to Table 85).
CsrUint32	msgId	Unique message ID for associating indications and responses – use the ID from an indication in the matching response
CsrUint8	uidCounter	The current UID counter – refer to section 2.3.2 for details

Table 84: Parameters in a CSR_BT_AVRCP_TG_GET_ATTRIBUTES_IND message

The media player must reply to the CSR_BT_AVRCP_TG_GET_ATTRIBUTES _IND message by sending a CSR_BT_AVRCP_TG_GET_ATTRIBUTES_RES message to the profile with the following function:

```
CsrBtAvrcpTgGetAttributesResSend (CsrUint8 connId, CsrUint8 attribCount,
CsrUint16 attribDataLen, CsrUint8 *attribData, CsrUint32 msgId, CsrBtAvrcpStatus
status)
```

The arguments for the function are described in Table 85:

Type	Argument	Description
CsrUint8	connId	The unique connection ID previously received in a CSR_BT_AVRCP_CONNECT_IND/CFM message
CsrUint8	attribCount	Number of attributes returned in the response message
CsrUint16	attribDataLen	Length of the attribute data in bytes including two bytes extra that will be filled in by the AVRCP profile.
CsrUint8	*attribData	Pointer to the attribute data. The first two bytes allocated must be left unused.
CsrUint32	msgId	Unique message ID for associating indications and responses – use the ID from an indication in the matching response

Type	Argument	Description
CsrBtAvrcpStatus	Status	Indicates whether the operation is accepted or rejected by means of the defines prefixed with AVRCP_STATUS_ in csr_bt_avrcp_prim.h

Table 85: Arguments for CsrBtAvrcpTgGetAttributesResSend function

In case the GetElementAttribute response is fragmented, a CSR_BT_AVRCP_CT_GET_ATTRIBUTES _IND message will be sent to the controller application with the parameters listed in Table 86:

Type	Parameter	Description
CsrBtAvrcpPrim	Type	Signal identity – always CSR_BT_AVRCP_CT_GET_ATTRIBUTES_IND
CsrUint8	connectionId	The unique connection ID previously received in a CSR_BT_AVRCP_CONNECT_IND/CFM message
CsrBtAvrcpScope	scope	Scope as defined in section 2.3.1
CsrBtAvrcpUid	uid	The UID of the item – refer to section 2.3.2 for details
CsrUint8	attributeCount	Number of attributes in the indication message
CsrUint16	attribDataLen	Length of the attribute data in bytes
CsrUint8	*attribData	Pointer to the attribute data. For AVRCP 1.3, the data may be decoded using the helper function CsrBtAvrcpCtLibElementsAttributeGet, which is described in section 4.2.8. For AVRCP 1.4, the helper function CsrBtAvrcpCtLibItemsAttributeGet described in section 4.2.9 may be used. The AVRCP version may be derived reading the tgFeatures parameter of the CSR_BT_AVRCP_CONNECT_CFM primitive (refer to section 3.4 for details.
CsrUint16	attribDataPayloadOffset	Offset to the first attribute data byte; that is the header data length, that shall be skipped to access the attribute data.

Table 86: Parameters in a CSR_BT_AVRCP_CT_GET_ATTRIBUTES_IND message

The controller application must reply to the CSR_BT_AVRCP_CT_GET_ATTRIBUTES _IND message by sending a CSR_BT_AVRCP_CT_GET_ATTRIBUTES _RES message. This is done by the following function:

```
CsrBtAvrcpCtGetAttributesResSend (CsrUint8 connId, CsrBool proceed)
```

The proceed parameter should be set to TRUE to request the next fragment or FALSE to abort the procedure.

When the operation is complete, a CSR_BT_AVRCP_CT_GET_ATTRIBUTES _CFM message will be sent to the controller application. Then contents of the message are listed in Table 87:

Type	Parameter	Description
CsrBtAvrcpPrim	type	Signal identity – always CSR_BT_AVRCP_CT_GET_ATTRIBUTES_CFM
CsrUint8	connectionId	The unique connection ID previously received in a CSR_BT_AVRCP_CONNECT_IND/CFM message
CsrBtAvrcpScope	scope	Scope as defined in section 2.3.1
CsrBtAvrcpUid	uid	The UID of the item – refer to section 2.3.2 for details
CsrUint8	attributeCount	Number of attributes in the confirm message
CsrUint16	attribDataLen	Length of the attribute data in bytes

Type	Parameter	Description
CsrUint8	*attribData	Pointer to the attribute data. For AVRCP 1.3, the data may be decoded using the helper function `CsrBtAvrcpCtLibElementsAttributeGet`, which is described in section 4.2.8. For AVRCP 1.4, the helper function `CsrBtAvrcpCtLibItemsAttributeGet` described in section 4.2.9 may be used. The AVRCP version may be derived reading the tgFeatures parameter of the CSR_BT_AVRCP_CONNECT_CFM primitive (refer to section 3.4 for details.
CsrBtResultCode	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.
CsrBtSupplier	resultSupplier	This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h
CsrUint16	attribDataPayloadOffset	Offset to the first attribute data byte; that is the header data length, that shall be skipped to access the attribute data.

Table 87: Parameters in a CSR_BT_AVRCP_CT_GET_ATTRIBUTES_CFM message

3.22 Get Play Status

The controller can query the addressed media player on a remote target by using the interface in Figure 24.

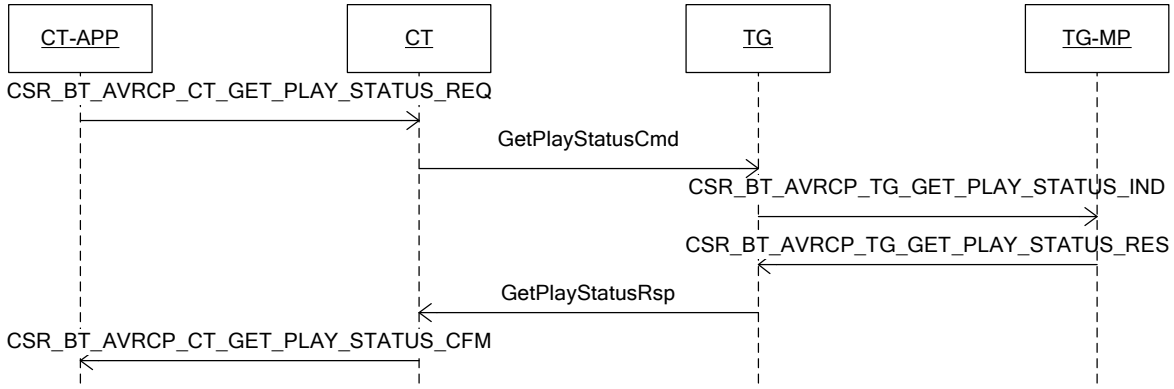


Figure 24: Get Play Status interface

The procedure can be initiated by the controller by sending a CSR_BT_AVRCP_CT_GET_PLAY_STATUS_REQ to the profile. This is achieved by using the function specified in the following:

```
CsrBtAvrcpCtGetPlayStatusReqSend (CsrSchedQid phandle, CsrUint8 connId)
```

The arguments for the function are described in Table 88.

Type	Argument	Description
CsrSchedQid	phandle	ID of the task to where the confirmation message should be sent
CsrUint8	connId	The unique connection ID previously received in a CSR_BT_AVRCP_CONNECT_IND/CFM message

Table 88: Arguments for the `CsrBtAvrcpCtGetPlayStatusReqSend` function

The addressed media player on the target will be notified of the controller command by a CSR_BT_AVRCP_TG_GET_PLAY_STATUS_IND message, which has the parameters listed in Table 89:

Type	Parameter	Description
CsrBtAvrcpPrim	type	Signal identity – always CSR_BT_AVRCP_TG_SET_PLAY_STATUS_IND
CsrUint8	connectionId	The unique connection ID previously received in a CSR_BT_AVRCP_CONNECT_IND/CFM message
CsrUint32	playerId	Unique ID of the media player the message is intended for
CsrUint32	msgId	Unique message ID for associating indications and responses – use the ID from an indication in the matching response

Table 89: Parameters in a CSR_BT_AVRCP_TG_GET_PLAY_STATUS_IND message

The CSR_BT_AVRCP_TG_GET_PLAY_STATUS_IND message must be replied with a CSR_BT_AVRCP_TG_GET_PLAY_STATUS_RES message using the following function:

```
CsrBtAvrcpTgGetPlayStatusResSend (CsrUint8 connId, CsrUint32
songLength, CsrUint32 songPosition, CsrBtAvrcpPlaybackStatus
playStatus, CsrUint32 msgId, CsrBtAvrcpStatus status)
```

The arguments for the function are described in Table 90:

Type	Argument	Description
CsrUint8	connId	The unique connection ID previously received in a CSR_BT_AVRCP_CONNECT_IND/CFM message
CsrUint32	songLength	The total length of the playing song in milliseconds
CsrUint32	songPosition	The current position of the playing song in milliseconds elapsed
CsrBtAvrcpPlaybackStatus	playStatus	Use one of the following values: – CSR_BT_AVRCP_PLAYBACK_STATUS_STOPPED – CSR_BT_AVRCP_PLAYBACK_STATUS_PLAYING – CSR_BT_AVRCP_PLAYBACK_STATUS_PAUSED – CSR_BT_AVRCP_PLAYBACK_STATUS_FWD_SEEK – CSR_BT_AVRCP_PLAYBACK_STATUS_REV_SEEK – CSR_BT_AVRCP_PLAYBACK_STATUS_ERROR
CsrUint32	msgId	Unique message ID for associating indications and responses – use the ID from an indication in the matching response
CsrBtAvrcpStatus	status	Indicates whether the operation is accepted or rejected by means of the defines prefixed with AVRCP_STATUS_ in csr_bt_avrcp_prim.h

Table 90: Arguments for the `CsrBtAvrcpTgGetPlayStatusResSend` function

When the operation is complete, the controller application will be notified with a CSR_BT_AVRCP_CT_GET_PLAY_STATUS message with the parameters in Table 91:

Type	Parameter	Description
CsrBtAvrcpPrim	Type	Signal identity – always CSR_BT_AVRCP_CT_GET_PLAY_STATUS_CFM
CsrUint8	connectionId	The unique connection ID previously received in a CSR_BT_AVRCP_CONNECT_IND/CFM message
CsrUint32	songLength	The total length of the playing song in milliseconds
CsrUint32	songPosition	The current position of the playing song in milliseconds elapsed
CsrBtAvrcpPlaybackStatus	playStatus	Refer to Table 90

Type	Parameter	Description
CsrBtResultCode	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.
CsrBtSupplier	resultSupplier	This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h

Table 91: Parameters in a CSR_BT_AVRCP_CT_GET_PLAY_STATUS_CFM message

3.23 Inform Displayable Character Set

The controller can notify a remote target of which character sets the controller supports. This is done using the interfaces in Figure 25.

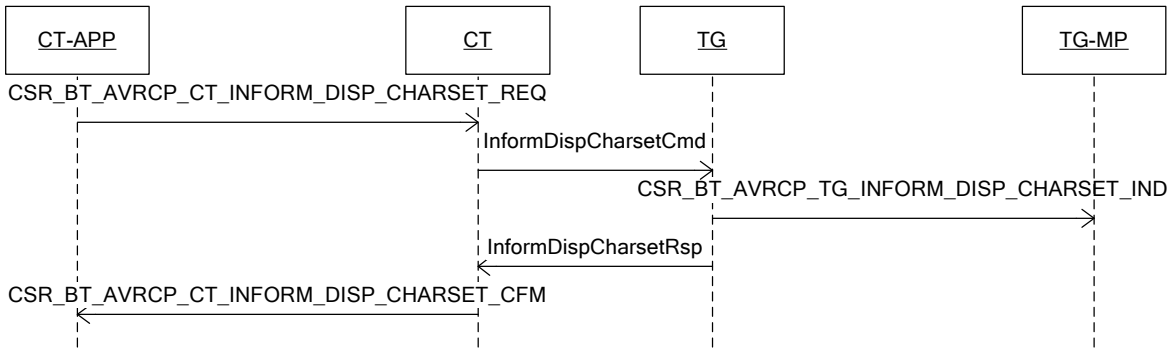


Figure 25: Inform Displayable Character Set interface

To initiate the procedure, the controller application must send a CSR_BT_AVRCP_CT_INFORM_DISP_CHARSET_REQ message to the profile – this is done using the following function:

```
CsrBtAvrcpCtInformDispCharSetReqSend (CsrSchedQid phandle, CsrUint8
connId, CsrUint8 charsetCount, CsrBtAvrcpCharSet *charset)
```

The arguments for the function are described in Table 92.

Type	Argument	Description
CsrSchedQid	phandle	ID of the task to where the confirmation message should be sent
CsrUint8	connId	The unique connection ID previously received in a CSR_BT_AVRCP_CONNECT_IND/CFM message
CsrUint8	charsetCount	Number of different character sets supported by the controller - set to 0 to use defaults (only UTF-8)
CsrBtAvrcpCharSet	*charset	Pointer to supported character sets - set to NULL to use defaults (only UTF-8)

Table 92: Arguments for `CsrBtAvrcpCtInformDispCharSetReqSend` function

The addressed media player on the target will be notified of the supported character sets of a controller by a CSR_BT_AVRCP_TG_INFORM_DISP_CHARSET_IND message with the parameters in Table 93:

Type	Parameter	Description
CsrBtAvrcpPrim	type	Signal identity – always CSR_BT_AVRCP_TG_INFORM_DISP_CHARSET_IND

Type	Parameter	Description
CsrUint8	connectionId	The unique connection ID previously received in a CSR_BT_AVRCP_CONNECT_IND/CFM message
CsrUint32	playerId	Unique ID of the media player the message is intended for
CsrUint8	charsetCount	Number of different character sets supported by the controller. If set to 0, use default character set (UTF-8).
CsrBtAvrcpCharSet	*charset	Pointer to supported character sets – if set to NULL, use default character set (UTF-8)

Table 93: Parameters in a CSR_BT_AVRCP_TG_INFORM_DISP_CHARSET_IND message

When the procedure is complete, a CSR_BT_AVRCP_CT_INFORM_DISP_CHARSET_CFM message will be sent to the controller application. The parameters in the message are listed in Table 94:

Type	Parameter	Description
CsrBtAvrcpPrim	type	Signal identity – always CSR_BT_AVRCP_CT_INFORM_DISP_CHARSET_CFM
CsrUint8	connectionId	The unique connection ID previously received in a CSR_BT_AVRCP_CONNECT_IND/CFM message
CsrBtResultCode	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.
CsrBtSupplier	resultSupplier	This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h

Table 94: Parameters in a CSR_BT_AVRCP_CT_INFORM_DISP_CHARSET_CFM message

3.24 Add To Now Playing List

The controller can add a media element to the Now Playing List by using the interface in Figure 26.

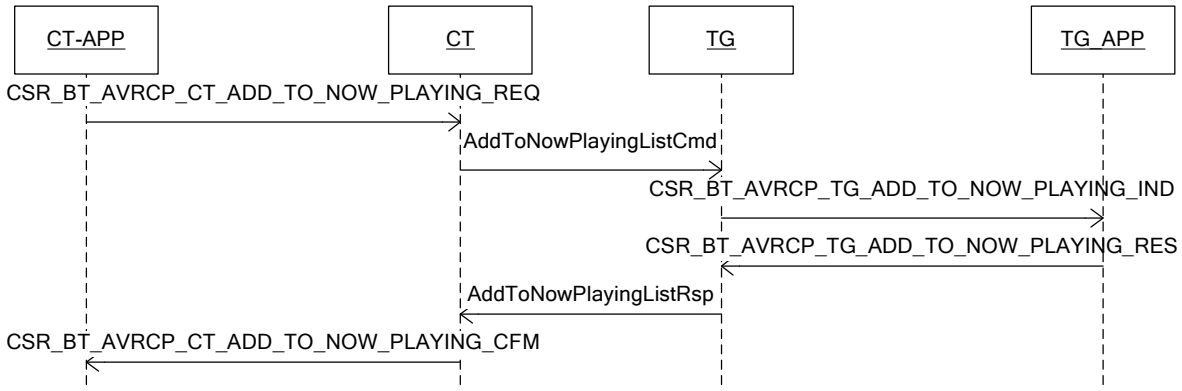


Figure 26: Add To Now Playing List interface

To add a media item to the Now Playing List, the controller must send a CSR_BT_AVRCP_CT_ADD_TO_NOW_PLAYING_REQ message to the profile using the following function:

```
CsrBtAvrcpCtAddToNowPlayingReqSend (CsrSchedQid phandle, CsrUint8
connId, CsrBtAvrcpScope scope, CsrUint16 uidCounter, CsrBtAvrcpUid
uid)
```

The arguments for the function are described in Table 95.

Type	Argument	Description
Phandle_t	phandle	ID of the task to where the confirmation message should be sent
CsrUint8	connId	The unique connection ID previously received in a CSR_BT_AVRCP_CONNECT_IND/CFM message
CsrBtAvrcpScope	scope	Scope as defined in section 2.3.1
CsrUint16	uidCounter	The current UID counter – refer to section 2.3.2 for details
CsrBtAvrcpUid	uid	The UID of the item – refer to section 2.3.2 for details

Table 95: Arguments for `CsrBtAvrcpCtAddToNowPlayingReqSend` function

The addressed media player will be notified of the request by a CSR_BT_AVRCP_TG_ADD_TO_NOW_PLAYING_IND message with the parameters in Table 96:

Type	Parameter	Description
CsrBtAvrcpPrim	type	Signal identity – always CSR_BT_AVRCP_TG_ADD_TO_NOW_PLAYING_IND
CsrUint8	connectionId	The unique connection ID previously received in a CSR_BT_AVRCP_CONNECT_IND/CFM message
CsrUint32	playerId	Unique ID of the media player the message is intended for
CsrBtAvrcpScope	scope	Scope as defined in section 2.3.1
CsrUint16	uidCounter	The current UID counter – refer to section 2.3.2 for details
CsrBtAvrcpUid	uid	The UID of the item – refer to section 2.3.2 for details
CsrUint32	msgId	Unique message ID for associating indications and responses – use the ID from an indication in the matching response

Table 96: Parameters in a CSR_BT_AVRCP_TG_ADD_TO_NOW_PLAYING_IND message

The media player must reply to the CSR_BT_AVRCP_TG_ADD_TO_NOW_PLAYING_IND by sending a CSR_BT_AVRCP_TG_ADD_TO_NOW_PLAYING_RES message with the following function:

```
CsrBtAvrcpTgAddToNowPlayingResSend (CsrUint8 connId, CsrUint32 msgId,
CsrBtAvrcpStatus status)
```

The arguments for the function are described in Table 97.

Type	Argument	Description
CsrUint8	connId	The unique connection ID previously received in a CSR_BT_AVRCP_CONNECT_IND/CFM message
CsrUint32	msgId	Unique message ID for associating indications and responses – use the ID from an indication in the matching response
CsrBtAvrcpStatus	status	Indicates whether the operation is accepted or rejected by means of the defines prefixed with `AVRCP_STATUS_` in csr_bt_avrcp_prim.h

Table 97: Arguments for `CsrBtAvrcpTgAddToNowPlayingResSend` function

At the completion of the procedure, a CSR_BT_AVRCP_CT_ADD_TO_NOW_PLAYING_CFM message will be sent to the controller application. The contents of the message are listed in Table 98:

Type	Parameter	Description
CsrBtAvrcpPrim	type	Signal identity – always CSR_BT_AVRCP_CT_ADD_TO_NOW_PLAYING_CFM
CsrUint8	connectionId	The unique connection ID previously received in a CSR_BT_AVRCP_CONNECT_IND/CFM message
CsrBtAvrcpScope	scope	Scope as defined in section 2.3.1
CsrBtAvrcpUid	uid	The UID of the item – refer to section 2.3.2 for details

Type	Parameter	Description
CsrBtResultCode	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.
CsrBtSupplier	resultSupplier	This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h

Table 98: Parameters in a CSR_BT_AVRCP_CT_ADD_TO_NOW_PLAYING_CFM message

3.25 Inform Battery Status of CT

The controller can notify a remote target about the actual status of its battery, through the interfaces in the figure below.

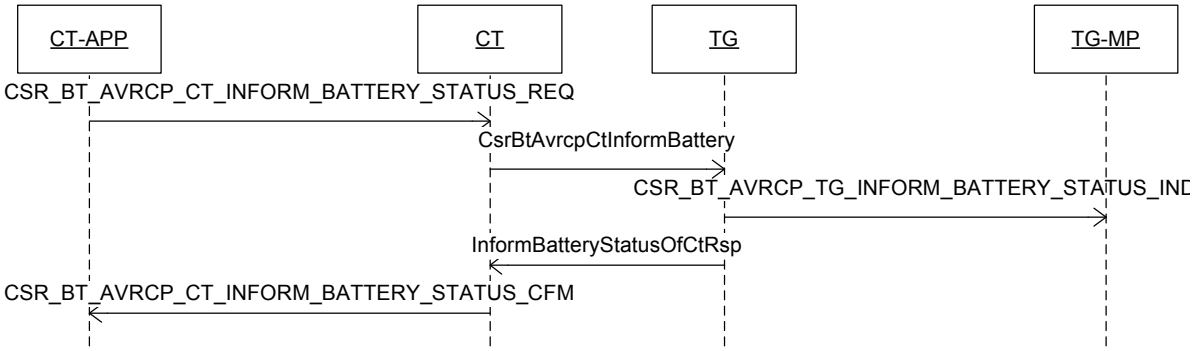


Figure 27: Inform Battery status of CT interface

To initiate the procedure, the controller application must send a CSR_BT_AVRCP_CT_INFORM_BATTERY_STATUS_REQ message to the profile – this is done using the following function:

```
CsrBtAvrcpCtInformBatteryStatusReqSend (CsrSchedQid phandle, CsrUint8
connId, CsrBtAvrcpBatteryStatus batStatus)
```

The arguments for the function are described in the table below.

Type	Argument	Description
CsrSchedQid	phandle	ID of the task to where the confirmation message should be sent
CsrUint8	connId	The unique connection ID previously received in a CSR_BT_AVRCP_CONNECT_IND/CFM message
CsrBtAvrcpBatteryStatus	batStatus	See list of possible values in csr_bt_avrcp_prim.h

Table 99: Arguments for CsrBtAvrcpCtInformBatteryStatusReqSend function

The addressed media player on the target will be notified of the supported character sets of a controller by a CSR_BT_AVRCP_TG_INFORM_BATTERY_STATUS_IND message with the parameters in the table below:

Type	Parameter	Description
CsrBtAvrcpPrim	type	Signal identity – always CSR_BT_AVRCP_TG_INFORM_BATTERY_STATUS_IND
CsrUint8	connectionId	The unique connection ID previously received in a CSR_BT_AVRCP_CONNECT_IND/CFM message
CsrUint32	playerId	Unique ID of the media player the message is intended for

Type	Parameter	Description
CsrBtAvrcpBatteryStatus	batStatus	See list of possible values in csr_bt_avrcp_prim.h

Table 100: Parameters in a CSR_BT_AVRCP_TG_INFORM_BATTERY_STATUS_IND message

When the procedure is complete, a CSR_BT_AVRCP_CT_INFORM_BATTERY_STATUS_CFM message will be sent to the controller application. The parameters in the message are listed in the table below:

Type	Parameter	Description
CsrBtAvrcpPrim	type	Signal identity – always CSR_BT_AVRCP_CT_INFORM_BATTERY_STATUS_CFM
CsrUint8	connectionId	The unique connection ID previously received in a CSR_BT_AVRCP_CONNECT_IND/CFM message
CsrBtResultCode	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.
CsrBtSupplier	resultSupplier	This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h

Table 101: Parameters in a CSR_BT_AVRCP_CT_INFORM_BATTERY_STATUS_CFM message

3.26 UNIT INFO COMMAND and SUB-UNIT INFO COMMAND (only CT)

The CT device can choose to send a “unit info command” or a “sub-unit info command” to the TG device, with data payload gathered at the application level. To do that, the application shall issue either the CSR_BT_AVRCP_CT_UNIT_INFO_CMD_REQ or the CSR_BT_AVRCP_CT_SUB_UNIT_INFO_CMD_REQ. When the TG device has answered the request or no answer has been received after a timeout, the profile will issue the relevant confirm message (CSR_BT_AVRCP_CT_UNIT_INFO_CMD_CFM or CSR_BT_AVRCP_CT_SUB_UNIT_INFO_CFM) to the application including the result of the operation performed.

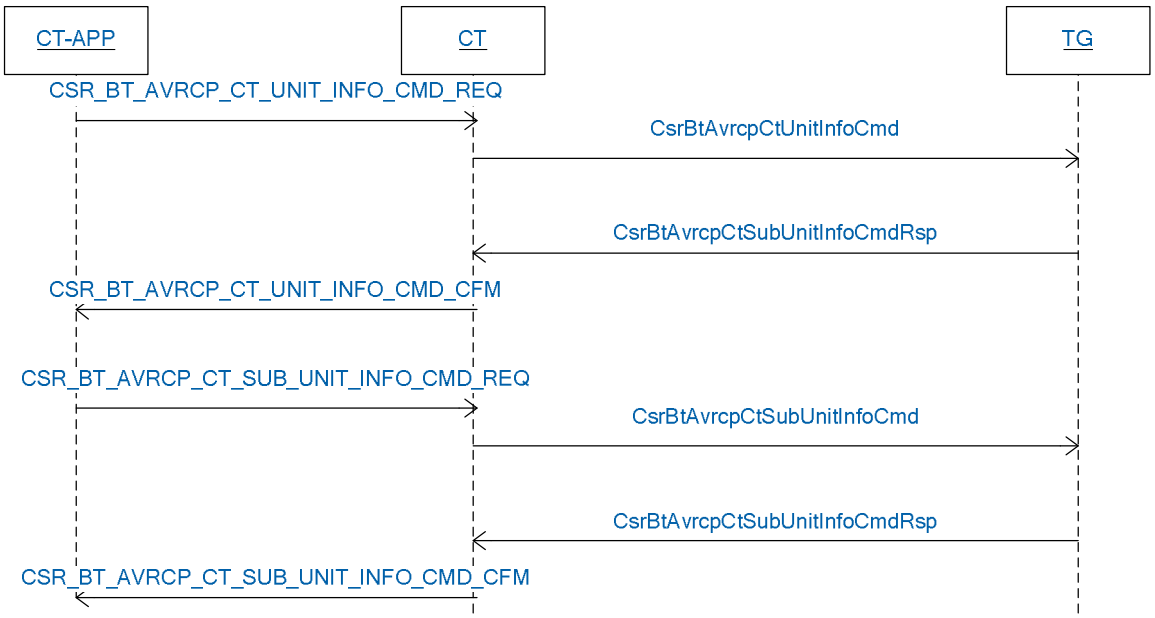


Figure 28: Unit Info command and Sub-Unit info command interface

The application shall use the interface functions:

CsrBtAvrcpCtUnitInfoCmdReqSend(CsrSchedQid phandle, CsrUint8 connId, CsrUint16 pDatalen, CsrUint8 *pData)

Or CsrBtAvrcpCtUnitInfoCmdReqSend(CsrSchedQid phandle, CsrUint8 connId, CsrUint16 pDatalen, CsrUint8 *pData)

In order to start the procedure.

The arguments for these functions are described in the table below.

Type	Argument	Description
CsrSchedQid	phandle	ID of the task to where the confirmation message should be sent
CsrUint8	connId	The unique connection ID previously received in a CSR_BT_AVRCP_CONNECT_IND/CFM message
CsrUint16	pDataLen	Length in bytes of the data payload to attach
CsrUint8	*pData	Pointer to the data stream to send in the message. The contents of the allocated area will be sent as they are and the pointer will be de-allocated by the CT.

Table 102: Arguments for CsrBtAvrcpCtUnitInfoCmdReqSend and CsrBtAvrcpCtSubUnitInfoCmdReqSend functions

When the procedure is complete, a CSR_BT_AVRCP_CT_UNIT_INFO_CMD_CFM or CSR_BT_AVRCP_CT_SUB_UNIT_INFO_CMD_CFM messages will be sent to the controller application. The parameters in the message are listed in the table below:

Type	Parameter	Description
CsrBtAvrcpPrim	type	Signal identity – always CSR_BT_AVRCP_CT_UNIT_INFO_CMD_CFM or CSR_BT_AVRCP_CT_SUB_UNIT_INFO_CMD_CFM
CsrUint8	connectionId	The unique connection ID previously received in a CSR_BT_AVRCP_CONNECT_IND/CFM message
CsrBtResultCode	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.
CsrBtSupplier	resultSupplier	This parameter specifies the supplier of the result given in resultCode. Possible values can be found in csr_bt_result.h
CsrUint16	pDataLen	Length in bytes of the data payload received
CsrUint8	*pData	Pointer to the data stream received in the message. The contents of the allocated area must be de-allocated by the receiving task.

Table 103: Parameters in a CSR_BT_AVRCP_CT_UNIT_INFO_CMD_CFM and CSR_BT_AVRCP_CT_SUB_UNIT_INFO_CMD_CFM messages

4 Helper Functions

Helper functions are used for getting AVRCP information e.g. folder name, attribute data etc. or to build a specific message e.g. one that contains the media name or media attributes.

4.1 Helper Functions to add

4.1.1 CsrBtAvrcpTgLibGfiFolderAdd

A function to build the message/data that is e.g. used in the `CsrBtAvrcpTgGetFolderItemsResSend()` function. As the figure show the TG must send a CSR_BT_AVRCP_TG_GET_FOLDER_ITEMS_RES when receiving a CSR_BT_AVRCP_TG_GET_FOLDER_ITEMS_IND, this helper function helps to generate the input data `*items` which is used in `CsrBtAvrcpTgGetFolderItemsResSend()`.

```
CsrBool CsrBtAvrcpTgLibGfiFolderAdd(CsrUint16 maxData, CsrUint16 *index,
CsrUint8 **data, CsrBtAvrcpUid *folderUid, CsrBtAvrcpFolderType folderType,
CsrBtAvrcpFolderPlayableType playableType, CsrBtAvrcpCharSet charset,
CsrCharString *folderName);
```

The arguments for this function are described in the table below.

Type	Argument	Description
`CsrUint16`	`maxData`	(Input) Maximum of data
`CsrUint16`	`*index`	(Output) The length on the data
`CsrUint8`	`**data`	(Output) The data
`CsrBtAvrcpUid`	`*folderUid`	(Input) The folder Uid
`CsrBtAvrcpFolderType`	`folderType`	(Input) The folder type
`CsrBtAvrcpFolderPlayableType`	`playableType`	(Input) The playable type
`CsrBtAvrcpCharSet`	`charset`	(Input) The char set used in folder name
`CsrCharString`	`*folderName`	(Input) The folder name

Table 104: Arguments for CsrBtAvrcpTgLibGfiFolderAdd

Figure 29: Get Folder Items example

4.1.2 CsrBtAvrcpTgLibGfiMediaAdd

A function to build the message/data that is used in e.g. `CsrBtAvrcpTgGetFolderItemsResSend`

```
CsrBool CsrBtAvrcpTgLibGfiMediaAdd(CsrUint16 maxData, CsrUint16 *index,
CsrUint8 **data, CsrUint16 *mediaIndex, CsrBtAvrcpUid *mediaUid,
CsrBtAvrcpMediaType mediaType, CsrBtAvrcpCharSet charset, CsrCharString
*mediaName);
```

The arguments for this function are described in the table below.

Type	Argument	Description
`CsrUint16`	`maxData`	(Input) Maximum of data
`CsrUint16`	`*index`	(Output) The length on the data
`CsrUint8`	`**data`	(Output) The data
`CsrUint16`	`*mediaIndex`	(Input) The media
`CsrBtAvrcpUid`	`*mediaUid`	(Input) The media Uid
`CsrBtAvrcpMediaType`	`mediaType`	(Input) The Media Type
`CsrBtAvrcpCharSet`	`charset`	(Input) The character set used in media Name
`CsrCharString`	`*mediaName`	(Input) The media name

Table 105: Arguments for CsrBtAvrcpTgLibGfiMediaAdd

4.1.3 CsrBtAvrcpTgLibGfiMediaAttributeAdd

A function to put in attributes in items.

```
CsrBool CsrBtAvrcpTgLibGfiMediaAttributeAdd(CsrUint16 maxData,
CsrUint16 *itemsLen, CsrUint8 **items, CsrUint16 mediaIndex,
CsrBtAvrcpItemMediaAttributeId attribId, CsrBtAvrcpCharSet charset,
CsrUint16 attLen, CsrUint8 *att);
```

The arguments for this function are described in the table below.

Type	Argument	Description
CsrUint16	maxData	(Input) Maximum of number of data bytes
CsrUint16	*itemsLen	(Input/output) Length of the data to be sent in the corresponding CSR_BT_AVRCP_TG_GET_FOLDER_ITEMS_RES
CsrUint8	**items	(Input/output) Address of the data to be sent in the corresponding CSR_BT_AVRCP_TG_GET_FOLDER_ITEMS_RES
CsrUint16	*mediaIndex	(Input) Index to the media item that needs to be parsed.
CsrBtAvrcpItemMediaAttributeId	attribId	(Input) Attribute to add. Valid values are: • CSR_BT_AVRCP_ITEM_ATT_MINMUM • CSR_BT_AVRCP_ITEM_ATT_TITLE • CSR_BT_AVRCP_ITEM_ATT_ARTIST • CSR_BT_AVRCP_ITEM_ATT_ALBUM • CSR_BT_AVRCP_ITEM_ATT_MEDIA_NUMBER • CSR_BT_AVRCP_ITEM_ATT_TOTAL_NUMBER • CSR_BT_AVRCP_ITEM_ATT_GENRE • CSR_BT_AVRCP_ITEM_ATT_TIME • CSR_BT_AVRCP_ITEM_ATT_COUNT • CSR_BT_AVRCP_ITEM_ATT_INVALID
CsrBtAvrcpCharSet	charset	(Input) The character set used for the media attributes. Valid value is: CSR_BT_AVRCP_CHARACTER_SET_UTF_8
CsrUint16	attLen	(Input) Length of the attributes data in bytes
CsrUint8	*att	(Input) Pointer to the attributes data

Table 106: Arguments for `CsrBtAvrcpTgLibGfiMediaAttributeAdd`

4.2 Helper Functions to parse the Response

4.2.1 CsrBtAvrcpCtLibGfiNextGet

The helper function `CsrBtAvrcpCtLibGfiNextGet` allows the application to get the type (`itemType`) of the next item pointed by the index. The type of item returned is of `CsrBtAvrcpItemType` and can carry one of the item types; `…ITEM_TYPE_MEDIA_PLAYER`, `…ITEM_TYPE_FOLDER` or `…ITEM_TYPE_MEDIA_ELEMENT` which is defined in csr_bt_avrcp_prim.h

```
CsrBool CsrBtAvrcpCtLibGfiNextGet(CsrUint16 *index, CsrUint16 itemsLen,
CsrUint8 *items, CsrBtAvrcpItemType *itemType);
```

The arguments for this function are described in the table below.

Type	Argument	Description
CsrUint16	*index	(Input/output) Index to the item that has been just read. If the input index is set to 0x0000, then it is considered to read the first item
CsrUint16	itemsLen	(Input) Total length of item, the length received by the application through CSR_BT_AVRCP_CT_GET_FOLDER_ITEMS_CFM
CsrUint8	*items	(Input) The item data, the pointer received by application through CSR_BT_AVRCP_CT_GET_FOLDER_ITEMS_CFM
CsrBtAvrcpItemType	*itemType	(Output) The item type of the next item referred by the index

Table 107: Arguments for `CsrBtAvrcpCtLibGfiNextGet`

Depending upon the type of the item returned by `CsrBtAvrcpCtLibGfiNextGet`, the application can use the other helper function to get the details of the item.

When there are no more items types to be read, then `CsrBtAvrcpCtLibGfiNextGet` () returns FALSE.

4.2.2 CsrBtAvrcpCtLibGfiMpGet

This helper function has been deprecated. Instead the `CsrBtAvrcpCtLibExtGfiMpGet` described in chapter 4.2.3 should be used.

The helper function `CsrBtAvrcpCtLibGfiMpGet` can be used for parsing the item of type `CSR_BT_AVRCP_ITEM_TYPE_MEDIA_PLAYER`. The application must call `CsrBtAvrcpCtLibGfiMpGet` with the items data and the index of the item to be parsed. The function will then read the media player item information on the output parameters.

```
CsrBool CsrBtAvrcpCtLibGfiMpGet(CsrUint16 *index, CsrUint16 itemsLen,
CsrUint8 *items, CsrBtAvrcpMpTypeMajor *majorType, CsrBtAvrcpMpTypeSub
*subType, CsrBtAvrcpPlaybackStatus *playbackStatus, CsrBtAvrcpMpFeatureMask
*featureMask, CsrBtAvrcpCharSet *charset, CsrUint16 *playerNameLen, CsrUint8
**playerName);
```

The arguments for this function are described in the table below.

Type	*Argument*	*Description*
CsrUint16	**index*	*(Input/output) Index to the item that has been just read. If the input index is set to 0x0000, then it is considered to read the first item*
CsrUint16	*itemsLen*	*(Input) Total length of item, the length received by the application through CSR_BT_AVRCP_CT_GET_FOLDER_ITEMS_CFM*
CsrUint8	**items*	*(Input) The item data, the pointer received by application through CSR_BT_AVRCP_CT_GET_FOLDER_ITEMS_CFM*
CsrBtAvrcpMpTypeMajor	**majorType*	*(Output) The media player major type*
CsrBtAvrcpMpTypeSub	**subtype*	*(Output) The media player sub type*
CsrBtAvrcpPlaybackStatus	**playbackStatus*	*(Output) Playback status*
CsrBtAvrcpMpFeatureMask	**featureMask*	*(Output) Feature mask*

Type	Argument	Description
CsrBtAvrcpCharSet	*charset	(Output) The character set used in the playerName
CsrUint16	*playerNameLen	(Output) The length of playerName
CsrUint8	**playerName	(Output) The player name

Table 108: Arguments for CsrBtAvrcpCtLibGfiMpGet

If there are no more items types to be read, then the CsrBtAvrcpCtLibGfiMpGet () returns FALSE. For detail explanation of each individual output parameter see csr_bt_avrcp_prim.h and csr_bt_avrcp_lib.h.

4.2.3 CsrBtAvrcpCtLibExtGfiMpGet

The helper function CsrBtAvrcpCtLibExtGfiMpGet can be used for parsing the item of type CSR_BT_AVRCP_ITEM_TYPE_MEDIA_PLAYER. The application must call CsrBtAvrcpCtLibExtGfiMpGet with the items data and the index of the item to be parsed. The function will then read the media player item information on the output parameters.

```
CsrBool CsrBtAvrcpCtLibExtGfiMpGet(CsrUint16 *index, CsrUint16 itemsLen,
CsrUint8 *items, CsrUint16 *playerId, CsrBtAvrcpMpTypeMajor *majorType,
CsrBtAvrcpMpTypeSub *subtype, CsrBtAvrcpPlaybackStatus *playbackStatus,
CsrBtAvrcpMpFeatureMask  *featureMask, CsrBtAvrcpCharSet *charset, CsrUint16
*playerNameLen, CsrUint8 **playerName);
```

The arguments for this function are described in the table below.

Type	Argument	Description
CsrUint16	*index	(Input/output) Index to the item that has been just read. If the input index is set to 0x0000, then it is considered to read the first item
CsrUint16	itemsLen	(Input) Total length of item, the length received by the application through CSR_BT_AVRCP_CT_GET_FOLDER_ITEMS_CFM
CsrUint8	*items	(Input) The item data, the pointer received by application through CSR_BT_AVRCP_CT_GET_FOLDER_ITEMS_CFM
CsrUint16	*playerId	(Output) Unique identifier for the media player
CsrBtAvrcpMpTypeMajor	*majorType	(Output) The media player major type
CsrBtAvrcpMpTypeSub	*subtype	(Output) The media player sub type
CsrBtAvrcpPlaybackStatus	*playbackStatus	(Output) Playback status
CsrBtAvrcpMpFeatureMask	*featureMask	(Output) Feature mask
CsrBtAvrcpCharSet	*charset	(Output) The character set used in the playerName
CsrUint16	*playerNameLen	(Output) The length of playerName
CsrUint8	**playerName	(Output) The player name

Table 109: Arguments for CsrBtAvrcpCtLibExtGfiMpGet

If there are no more items types to be read, then the CsrBtAvrcpCtLibExtGfiMpGet () returns FALSE. For detail explanation of each individual output parameter see csr_bt_avrcp_prim.h and csr_bt_avrcp_lib.h.

4.2.4 CsrBtAvrcpCtLibGfiFolderGet

The helper function CsrBtAvrcpCtLibGfiFolderGet can be used for parsing the item of type CSR_BT_AVRCP_ITEM_TYPE_FOLDER. The application must call CsrBtAvrcpCtLibGfiFolderGet with the items data and the index of the item to be parsed. The function will then read the folder item information on the output parameters.

```
CsrBool CsrBtAvrcpCtLibGfiFolderGet(CsrUint16 *index, CsrUint16 itemsLen,
CsrUint8 *items, CsrBtAvrcpUid *folderUid, CsrBtAvrcpFolderType *folderType,
CsrBtAvrcpFolderPlayableType *playableType, CsrBtAvrcpCharSet *charset,
CsrUint16 *folderNameLen,
CsrUint8 **folderName);
```

The arguments for this function are described in the table below.

Type	Argument	Description
CsrUint16	*index	(Input/output) Index to the item that has been just read. If the input index is set to 0x0000, then it is considered to read the first item
CsrUint16	itemsLen	(Input) Total length of item, the length received by the application through CSR_BT_AVRCP_CT_GET_FOLDER_ITEMS_CFM
CsrUint8	*items	(Input) The item data, the pointer received by application through CSR_BT_AVRCP_CT_GET_FOLDER_ITEMS_CFM
CsrBtAvrcpUid	*folderUid	(Output) Folder Uid
CsrBtAvrcpFolderType	*folderType	(Output) Folder type
CsrBtAvrcpFolderPlayableType	*playableType	(Output) Playable type
CsrBtAvrcpCharSet	*charset	(Output) The character set used in the folderName
CsrUint16	*folderNameLen	(Output) The length of folderName
CsrUint8	**folderName	(Output) The folder name

Table 110: Arguments for CsrBtAvrcpCtLibGfiFolderGet

If there are no more item types to be read, then CsrBtAvrcpCtLibGfiFolderGet () returns FALSE. For detail explanation of each individual output parameter please see csr_bt_avrcp_prim.h and csr_bt_avrcp_lib.h.

A pseudo code example showing the use of the CsrBtAvrcpCtLibGfiFolderGet–function is found in section 4.2.10.

4.2.5 CsrBtAvrcpCtLibGfiMediaGet

The helper function CsrBtAvrcpCtLibGfiMediaGet can be used for parsing the item of type CSR_BT_AVRCP_ITEM_TYPE_MEDIA_ELEMENT. The application must call CsrBtAvrcpCtLibGfiMediaGet with the items data and the index of the item to be parsed. The function will then read the media elements item information on the output parameters.

```
CsrBool CsrBtAvrcpCtLibGfiMediaGet(CsrUint16 *index, CsrUint16 itemsLen,
CsrUint8 *items, CsrBtAvrcpUid *mediaUid, CsrBtAvrcpMediaType *mediaType,
CsrBtAvrcpCharSet *charset, CsrUint16 *mediaNameLen, CsrUint8 **mediaName,
CsrUint8 *attributeCount);
```

The arguments for this function are described in the table below.

Type	Argument	Description
CsrUint16	*index	(Input/output) Index to the item that has been just read. If the input index is set to 0x0000, then it is considered to read the first item
CsrUint16	itemsLen	(Input) Total length of item, the length received by the application through CSR_BT_AVRCP_CT_GET_FOLDER_ITEMS_CFM
CsrUint8	*items	(Input) The item data, the pointer received by application through CSR_BT_AVRCP_CT_GET_FOLDER_ITEMS_CFM

Type	Argument	Description
CsrBtAvrcpUid	*mediaUid	(Output) The media Uid
CsrBtAvrcpMediaType	*media	(Output) The media type
CsrBtAvrcpCharSet	*charset	(Output) The character set used in the mediaName
CsrUint16	*mediaNameLen	(Output) Length of the mediaName
CsrUint8	**mediaName	(Output) The mediaName
CsrUint8	*attributeCount	(Output) The number of attributes

Table 111: Arguments for CsrBtAvrcpCtLibGfiMediaGet

If there are no more items types to be read, then CsrBtAvrcpCtLibGfiMediaGet () returns FALSE. For detail explanation of each individual output parameter, please see csr_bt_avrcp_prim.h and csr_bt_avrcp_lib.h.

A pseudo code example showing the use of the then CsrBtAvrcpCtLibGfiMediaGet–function is found in section 4.2.10.

4.2.6 CsrBtAvrcpCtLibGfiMediaAttributeNextGet

This helper function will be deprecated in a future version of CSR Synergy/BT. It is therefore not recommended to use this function. Instead, the CsrBtAvrcpCtLibGfiMediaAttributeGet described in chapter 4.2.7 should be used.

The application may call CsrBtAvrcpCtLibGfiMediaAttributeNextGet with the items data and a pointer to the index of the attribute of interest (*attIndex) to read the attribute id which is returned through *attribId. The different attribute Id's (…ITEM_ATT_MINIMUM,…ITEM_ATT_TITLE,…ITEM_ATT_ARTIST,…ITEM_ATT_ALBUM, …ITEM_ATT_MEDIA_NUMBER,…ITEM_ATT_TOTAL_NUMBER,…ITEM_ATT_GENRE,…ITEM_ATT_TIME) are found in csr_bt_avrcp_prim.h.

```
CsrBool CsrBtAvrcpCtLibGfiMediaAttributeNextGet(CsrUint16 *index, CsrUint16
*attIndex, CsrUint16 itemsLen, CsrUint8 *items, CsrBtAvrcpItemMediaAttributeId
*attribId);
```

The arguments for this function are described in the table below.

Type	Argument	Description
CsrUint16	* mediaIndex	(Input) Index to the media item that needs to be parsed.
CsrUint16	*attIndex	(Input/output) Pointer to the index of attribute to be parsed. The function will update the pointer, thus if the items data contain more then one attribute the function must be called with the new value
CsrUint16	itemsLen	(Input) Total length of item data. Typically it will be the attribDataLen received by the application from CSR_BT_AVRCP_CT_GET_ATTRIBUTES_CFM
CsrUint8	*items	(Input)The item data. Typically it will be the attribdata received by the application from CSR_BT_AVRCP_CT_GET_ATTRIBUTES_CFM
CsrBtAvrcpItemMediaAttributeId	*attribId	(Output) Attribute id

Table 112: Arguments for CsrBtAvrcpCtLibGfiMediaAttributeNextGet

Pseudo code explaining the usage of CsrBtAvrcpCtLibGfiMediaAttributeNextGet

Example 1: Use of both CsrBtAvrcpCtLibGfiMediaAttributeNextGet and CsrBtAvrcpCtLibGfiMediaAttributeGet

```
if (*prim == CSR_BT_AVRCP_CT_GET_ATTRIBUTES_CFM)
```

```
{
   CsrUint16   index=0;
   CsrUint16   attIndex=0;
   CsrBtAvrcpItemMediaAttributeId attribId;
   CsrUint16   i = 0;

   index = attIndex;
   CsrBtAvrcpCtLibGfiMediaAttributeNextGet(&index, &attIndex, ...);

    for (i = 0; i < prim->attributeCount; i++)
    {
        CsrUint16   indexTmp=0;

        switch (*attribId)
        {
           case CSR_BT_AVRCP_ITEM_ATT_ARTIST:
              CsrBtAvrcpCtLibGfiMediaAttributeGet(0xDEAD, &attIndex, ...);
              /* … … …*/
              break;

           case … … …
        }
    }
}
```

Example 2: Search of a specific attribute

```
if (*prim == CSR_BT_AVRCP_CT_GET_ATTRIBUTES_CFM)
{
    CsrUint16   index=0;
   CsrUint16   attIndex=0;
   CsrUint16   i = 0;

   for (i=0; i<ptr->attributeCount ; i++)
   {
       if(CsrBtAvrcpCtLibGfiMediaAttributeNextGet(&index, &attIndex, ...) ==
       CSR_BT_AVRCP_ITEM_ATT_ARTIST)
       {
          /* The application knows that there is a  */
          /* CSR_BT_AVRCP_ITEM_ATT_ARTIST attribute */
       }
   }
}
```

4.2.7 CsrBtAvrcpCtLibGfiMediaAttributeGet

A function to get media element attributes, the application must call CsrBtAvrcpCtLibGfiMediaAttributeGet with the items data and a pointer to the attribute that is of interest (*attIndex), the function will then read the attributes and return the data in **att.

```
CsrBool CsrBtAvrcpCtLibGfiMediaAttributeGet(CsrUint16 maxData, CsrUint16
*attIndex, CsrUint16 itemsLen, CsrUint8 *items,
CsrBtAvrcpItemMediaAttributeId *attribId, CsrBtAvrcpCharSet *charset,
CsrUint16 *attLen, CsrUint8 **att);
```

The arguments for this function are described in the table below.

Type	Argument	Description
CsrUint16	maxData	(Input) Not used currently.
CsrUint16	*attIndex	(Input/output) Pointer to the index of attribute to be parsed. The function will update the pointer, thus if the items data contain more then one attribute the function must be called with the new value
CsrUint16	itemsLen	(Input) Total length of item data. Typically it will be the itemsLen received by the application from CSR_BT_AVRCP_CT_GET_FOLDER_ITEMS_CFM
CsrUint8	*items	(Input)The item data. Typically it will be the items received by the application from CSR_BT_AVRCP_CT_GET_FOLDER_ITEMS_CFM
CsrBtAvrcpItemMediaAttributeId	*attribId	(Output) The attribute id
CsrBtAvrcpCharSet	*charset	(Output)The character set used in the attribute
CsrUint16	*attLen	(Output)The length of the attribute
CsrUint8	**att	(Output) The attribute data.

Table 113: Arguments for `CsrBtAvrcpCtLibGfiMediaAttributeGet`

If there are no more items types to be read, then the helper function returns FALSE. For detail explanation of each individual output parameter, please see csr_bt_avrcp_prim.h and csr_bt_avrcp_lib.h.

A pseudo code example showing the use of the then `CsrBtAvrcpCtLibGfiMediaAttributeGet`–function is found in section 4.2.10.

To parse all the attributes received in CSR_BT_AVRCP_CT_GET_FOLDER_ITEMS_CFM for type CSR_BT_AVRCP_ITEM_TYPE_MEDIA_ELEMENT, the application needs to call the function `CsrBtAvrcpCtLibGfiMediaAttributeGet()` repeatedly for `attributeCount` times. Each time when the `CsrBtAvrcpCtLibGfiMediaAttributeGet()` is called, the function will parse attribute ID, character set, attribute length and attribute data for attribute pointed by the `attIndex` from item. The `CsrBtAvrcpCtLibGfiMediaAttributeGet` () function will update the index each time its call thus pointing at the next attribute data when it is called again.

4.2.8 CsrBtAvrcpCtLibElementsAttributeGet

A function to get element attributes, the application must call CsrBtAvrcpCtLibElementsAttributeGet with the items data and a pointer to the attribute that is of interest(*attIndex), the function will then read the attributes and return the data in *att.

```
CsrBool CsrBtAvrcpCtLibElementsAttributeGet(CsrUint16 maxData, CsrUint16
*attIndex, CsrUint16 itemsLen, CsrUint8 *items,
CsrBtAvrcpItemMediaAttributeId *attribId, CsrBtAvrcpCharSet *charset,
CsrUint16 *attLen, CsrUint8 CsrUint8);
```

The arguments for this function are described in the table below.

Type	Argument	Description
CsrUint16	maxData	(Input) Defult value is 0x0000
CsrUint16	*attIndex	(Input/output) Pointer to the attribute of interest. The function will update the pointer, thus if the items data contain more then one attribute the function must be called with the new value
CsrUint16	itemsLen	(Input) Total length of item data
CsrUint8	*items	(Input)The item data
CsrBtAvrcpItemMediaAttributeId	*attribId	(Output) The attribute id

Type	Argument	Description
CsrBtAvrcpCharSet	*charset	(Output)The character set used in the attribute
CsrUint16	*attLen	(Output)The length of the attribute
CsrUint8	**att	(Output) The attribute data.

Table 114: Arguments for `CsrBtAvrcpCtLibElementsAttributeGet`

4.2.9 CsrBtAvrcpCtLibItemsAttributeGet

A function to get items attribute, the application must call CsrBtAvrcpCtLibItemsAttributeGet with the items data and a pointer to the attribute that is of interest (*attIndex), the function will then read the attributes and return the data in **att.

```
CsrBool CsrBtAvrcpCtLibItemsAttributeGet(CsrUint16 maxData, CsrUint16
*attIndex, CsrUint16 itemsLen, CsrUint8 *items,
CsrBtAvrcpItemMediaAttributeId *attribId, CsrBtAvrcpCharSet *charset,
CsrUint16 *attLen, CsrUint8 CsrUint8);
```

The arguments for this function are described in the table below.

Type	Argument	Description
CsrUint16	maxData	(Input) Defult value is 0x0000
CsrUint16	*attIndex	(Input/output) Pointer to the attribute of interest. The function will update the pointer, thus if the items data contain more then one attribute the function must be called with the new value
CsrUint16	itemsLen	(Input) Total length of item data
CsrUint8	*items	(Input)The item data
CsrBtAvrcpItemMediaAttributeId	*attribId	(Output) The attribute id
CsrBtAvrcpCharSet	*charset	(Output)The character set used in the attribute
CsrUint16	*attLen	(Output)The length of the attribute
CsrUint8	**att	(Output) The attribute data.

Table 115: Arguments for `CsrBtAvrcpCtLibItemsAttributeGet`

To parse all the attributes received in CSR_BT_AVRCP_CT_GET_ATTRIBUTES_CFM , the application needs to call the function `CsrBtAvrcpCtLibGfiMediaAttributeGet()` repeatedly for `attributeCount` times. Each time when the `CsrBtAvrcpCtLibGfiMediaAttributeGet()` is called, the function will parse attribute ID, character set, attribute length and attribute data for attribute pointed by the `attIndex` from item. The `CsrBtAvrcpCtLibGfiMediaAttributeGet` () function will update the index each time its call thus pointing at the next attribute data when it is called again.

Pseudo code explaining the usage of `CsrBtAvrcpCtLibItemsAttributeGet is found below:`

```
CsrBtAvrcpCtGetAttributesCfm     *prim = (CsrBtAvrcpCtGetAttributesCfm *)msg;

CsrUint16                        index = prim->attribDataPayloadOffset;
CsrBtAvrcpItemMediaAttributeId   attribId;
CsrBtAvrcpCharSet                charset;
CsrUint16                        attLen;
CsrUint8                         *att;
CsrUint8                         n=0;

    while (n < prim->attributeCount)
    {
        CsrUint16    i;

        CsrBtAvrcpCtLibItemsAttributeGet(0xDEAD,
```

```
                                        &index,
                                        prim->attribDataLen,
                                        prim->attribData,
                                        &attribId,
                                        &charset,
                                        &attLen,
                                        &att);

            /* The attribute data are now parsed and are found in the relevant */
            /* variables. It is now up the application to show the parameters  */
            /* on it's UI and save the parameters it needs for later use.      */

            n++;
        }
        CsrPfree(prim->attribData);
```

4.2.10 Example on using Helper Functions

Below, a pseudo code example on parsing the media items data of an incoming CSR_BT_AVRCP_CT_GET_FOLDER_ITEMS_CFM primitive is found:

```
CsrBtAvrcpCtGetFolderItemsCfm * prim = (CsrBtAvrcpCtGetFolderItemsCfm *)msg;
if (prim->resultCode == CSR_BT_RESULT_CODE_AVRCP_SUCCESS)
{
    switch(prim->scope)
    {
        case CSR_BT_AVRCP_SCOPE_SEARCH:
        case CSR_BT_AVRCP_SCOPE_NPL:
        case CSR_BT_AVRCP_SCOPE_MP_FS:
        {
            CsrBtAvrcpUid                   mediaUid;
            CsrBtAvrcpMediaType             mediaType;
            CsrBtAvrcpCharSet               charset;
            CsrBtAvrcpUid                   folderUid;
            CsrBtAvrcpFolderType            folderType;
            CsrBtAvrcpFolderPlayableType    playableType;
            CsrUint16                       mediaNameLen;
            CsrUint8                        *mediaName;
            CsrUint16                       folderNameLen;
            CsrUint8                        *folderName;
            CsrUint8                        attributeCount;
            CsrUint16                       index = CSR_BT_AVRCP_LIB_GFI_HEADER_OFFSET;
            CsrUint8                        idx = 0;
            CsrUint8                        i = 0, j = 0;
            CsrBtAvrcpItemType              itemType = CSR_BT_AVRCP_ITEM_TYPE_INVALID;

            /* Every folder can have a number of items in it. The number of attributes */
            /* are found in itemCount  */
            for (i = 0; i < prim->itemsCount; i++)
            {
                itemType = *(prim->itemsData + index);
                switch(itemType)
                {
                    case CSR_BT_AVRCP_ITEM_TYPE_FOLDER:
                    {
                        CsrUint8    *pName;

                        CsrBtAvrcpCtLibGfiFolderGet(&index,
                                                    prim->itemsDataLen,
                                                    prim->itemsData,
                                                    &folderUid,
                                                    &folderType,
                                                    &playableType,
                                                    &charset,
                                                    &folderNameLen,
                                                    &folderName);

                        /* The folder data are now parsed and are found in the relevant   */
                        /* variables. It is now up the application to show the parameters */
                        /* on it's UI and save the parameters it needs for later use.     */

                        CsrPfree(pName);
```

```
                break;
            }
            case CSR_BT_AVRCP_ITEM_TYPE_MEDIA_ELEMENT:
            {
               CsrUint16 count;
               CsrUint8 *pName;

               CsrBtAvrcpCtLibGfiMediaGet(&index,
                                          prim->itemsDataLen,
                                          prim->itemsData,
                                          &mediaUid,
                                          &mediaType,
                                          &charset,
                                          &mediaNameLen,
                                          &mediaName,
                                          &attributeCount);

                /* The media data are now parsed and are found in the relevant    */
                /* variables. It is now up the application to show the parameters */
                 /* on it's UI and save the parameters it needs for later use.     */

                CsrPfree(pName);

                /* Every media element can have a number of attributes associated */
                /* with it. The number of attributes are found in attributeCount  */
                for (count = 0; count < attributeCount; count++)
                {/* Get attributes */
                    CsrBtAvrcpItemMediaAttributeId  attribId;
                    CsrUint16                       attLen;
                    CsrUint8                        * att;
                    CsrUint8                        n=0;

                    if (CsrBtAvrcpCtLibGfiMediaAttributeGet(prim->itemsDataLen,&index,
                                                            prim->itemsDataLen,
                                                            prim->itemsData,
                                                            &attribId,
                                                            &charset,
                                                            &attLen,
                                                            &att))
                    {
                        /* The attribute data are now parsed and are found in the */
                        /* relevant variables. It is now up the application to    */
                        /* show the parameters on it's UI and save the parameters */
                        /* it needs for later use.                                */

                        CsrPfree(pName);
                    }
                }
                break;
            }
            case CSR_BT_AVRCP_ITEM_TYPE_INVALID: /* Intended fall-through */
            default:
            {
                printf("\nUnexpected  item type (0x%02X) received!\n", itemType);
                break;
            }
        }
    }
    break;
}
case CSR_BT_AVRCP_SCOPE_MP_LIST:
{
    CsrBtAvrcpMpTypeMajor      majorType;
    CsrBtAvrcpMpTypeSub        subType;
    CsrBtAvrcpPlaybackStatus   playbackStatus;
    CsrBtAvrcpMpFeatureMask    featureMask;
    CsrBtAvrcpCharSet          charset;
    CsrUint16                  playerId;
    CsrUint16                  playerNameLen;
    CsrUint8                   *playerName;
    CsrUint16                  index = CSR_BT_AVRCP_LIB_GFI_HEADER_OFFSET;
    CsrUint8                   mp_idx = 0;

    while (CsrBtAvrcpCtLibExtGfiMpGet(&index,
                                      prim->itemsDataLen,
```

```
                                        prim->itemsData,
                                        &playerId,
                                        &majorType,
                                        &subType,
                                        &playbackStatus,
                                        &featureMask,
                                        &charset,
                                        &playerNameLen,
                                        &playerName))
            {
                /* The playerName does not have a NULL terminator: It must be added */
                CsrUint8 *pName = CsrPmalloc(playerNameLen + 1);
                CsrMemSet(pName,0,playerNameLen + 1);
                CsrMemCpy(pName,playerName,playerNameLen);

                /* The media data are now parsed and are found in the relevant    */
                /* variables. It is now up the application to show the parameters */
                /* on it's UI and save the parameters it needs for later use.     */

                CsrPfree(pName);
            }
            break;
        }
        default:
        {
            /* This should not happen */
            break;
        }
    }

    CsrPfree(prim->itemsData);
```

5 Document References

Document	Reference
Audio/Video Remote Control Profile Version: 1.4 Date: 2008-06-26	[AVRCP]
Audio/Video Control Transport Protocol Specification Version: 1.2 Date: : 2007-04-16	[AVCTP]
AV/C Digital Interface Command Set General Specification Version 4.0. 1394 Trade Association	[AV/C Interface]
AV/C Panel Subunit Specification 1.1 1394 Trade Association	[AV/C Panel]
CSR Synergy Bluetooth, SC – Security Controller API Description	[SC]

Terms and Definitions

AV	Audio/Video
AVRCP	Audio Video Remote Control Profile
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
CT	Controller, AVRCP device s
TG	Target, AVRCP device role
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.