#### dotstack API Reference

Generated by Doxygen 1.8.9.1

Tue Sep 8 2015 17:42:45

### **Contents**

1	Mod	ule Inde	ex		1
	1.1	Module	es		1
2	Data	Struct	ure Index		3
	2.1	Data S	tructures		3
3	Mod	ule Doo	umentati	on	13
	3.1	Systen	n Function	s	13
		3.1.1	Detailed	Description	13
		3.1.2	Typedef	Documentation	13
			3.1.2.1	bt_sys_callback_fp	13
		3.1.3	Function	Documentation	14
			3.1.3.1	bt_sys_get_l2cap_manager	14
			3.1.3.2	bt_sys_get_version	14
			3.1.3.3	bt_sys_init	14
			3.1.3.4	bt_sys_init_ex	14
			3.1.3.5	bt_sys_start	15
	3.2	HCI .			16
		3.2.1	Detailed	Description	23
		3.2.2	Typedef	Documentation	23
			3.2.2.1	bt_hci_connect_callback_fp	23
			3.2.2.2	bt_hci_disconnect_callback_fp	24
			3.2.2.3	bt_hci_start_callback_fp	24
			3.2.2.4	bt_hci_stop_callback_fp	24
		3.2.3	Function	Documentation	24
			3.2.3.1	bt_hci_connect	24
			3.2.3.2	bt_hci_init	25
			3.2.3.3	bt_hci_init_ex	25
			3.2.3.4	bt_hci_listen	25
			3.2.3.5	bt_hci_reset	26
			3.2.3.6	bt_hci_start	26
			3237	bt hei start no init	26

iv CONTENTS

		3.2.3.8	bt_hci_stop	26
		3.2.3.9	bt_hci_write_local_name	27
3.3	HCI UA	ART (H4) t	ransport protocol	28
	3.3.1	Detailed	Description	28
	3.3.2	Function	Documentation	28
		3.3.2.1	bt_hcitr_uart_init	28
		3.3.2.2	bt_hcitr_uart_reset	28
		3.3.2.3	bt_hcitr_uart_start	28
3.4	Serial I	Port Profile	e (SPP)	29
	3.4.1	Detailed	Description	30
	3.4.2	Typedef [	Documentation	31
		3.4.2.1	bt_spp_receive_callback_fp	31
		3.4.2.2	bt_spp_send_callback_fp	31
		3.4.2.3	bt_spp_state_callback_fp	31
	3.4.3	Enumera	tion Type Documentation	31
		3.4.3.1	bt_spp_port_event_e	31
		3.4.3.2	bt_spp_port_state_e	32
		3.4.3.3	bt_spp_send_status_e	32
	3.4.4	Function	Documentation	33
		3.4.4.1	bt_spp_allocate	33
		3.4.4.2	bt_spp_cancel_receive	33
		3.4.4.3	bt_spp_cancel_send	33
		3.4.4.4	bt_spp_connect	33
		3.4.4.5	bt_spp_deallocate	35
		3.4.4.6	bt_spp_disconnect	35
		3.4.4.7	bt_spp_get_frame_length	35
		3.4.4.8	bt_spp_get_local_modem_status	35
		3.4.4.9	bt_spp_get_remote_address	36
		3.4.4.10	bt_spp_get_remote_modem_status	36
		3.4.4.11	bt_spp_init	36
		3.4.4.12	bt_spp_listen	36
		3.4.4.13	bt_spp_receive	37
		3.4.4.14	bt_spp_send	37
		3.4.4.15	bt_spp_set_dtr	38
		3.4.4.16	bt_spp_set_local_modem_status	38
		3.4.4.17	bt_spp_set_rts	38
3.5	SPP C	onfiguratio	n	40
	3.5.1	Detailed	Description	40
	3.5.2	Macro De	efinition Documentation	40
		3.5.2.1	SPP_MAX_PORTS	40

CONTENTS

3.6	RFCO	MM Config	guration	41
	3.6.1	Detailed	Description	41
	3.6.2	Macro De	efinition Documentation	42
		3.6.2.1	RFCOMM_ENABLE_MULTIDEVICE_CHANNELS	42
		3.6.2.2	RFCOMM_INFO_LEN	42
		3.6.2.3	RFCOMM_LOCAL_CREDIT	42
		3.6.2.4	RFCOMM_MAX_CMD_BUFFERS	42
		3.6.2.5	RFCOMM_MAX_DLCS	42
		3.6.2.6	RFCOMM_MAX_SERVER_CHANNELS	42
		3.6.2.7	RFCOMM_MAX_SESSIONS	42
3.7	SDP .			44
	3.7.1	Detailed	Description	45
	3.7.2	Macro De	efinition Documentation	45
		3.7.2.1	BEGIN_DE_SEQUENCE	45
		3.7.2.2	DE_BOOL	46
		3.7.2.3	DE_INT	46
		3.7.2.4	DE_STRING	46
		3.7.2.5	DE_STRING2	46
		3.7.2.6	DE_UINT	46
		3.7.2.7	DE_UINT16	47
		3.7.2.8	DE_URL	47
		3.7.2.9	DE_UUID128	47
		3.7.2.10	DE_UUID16	47
		3.7.2.11	DE_UUID32	48
		3.7.2.12	END_DE_SEQUENCE	48
		3.7.2.13	INIT_DE_SEQUENCE	48
	3.7.3	Function	Documentation	48
		3.7.3.1	bt_sdp_request_service_attribute	48
		3.7.3.2	bt_sdp_request_service_search	49
		3.7.3.3	bt_sdp_start	49
3.8	Configu	uration .		50
	3.8.1	Detailed	Description	50
	3.8.2	Macro De	efinition Documentation	50
		3.8.2.1	SDP_MAX_ATTRIBUTE_RESULT_LEN	50
		3.8.2.2	SDP_MAX_PDU_BUFFERS	50
		3.8.2.3	SDP_MAX_SEARCH_RESULT_LEN	51
3.9	HID Pr	ofile (HIDF	۶)	52
	3.9.1	Detailed	Description	52
	3.9.2	Function	Documentation	52
		3.9.2.1	bt_hid_add_report	52

vi CONTENTS

	3.9.2.2	bt_hid_allocate_session	52
	3.9.2.3	bt_hid_connect	53
	3.9.2.4	bt_hid_disconnect	53
	3.9.2.5	bt_hid_free_session	53
	3.9.2.6	bt_hid_init	54
	3.9.2.7	bt_hid_listen	54
	3.9.2.8	bt_hid_send_report	54
	3.9.2.9	bt_hid_unplug	55
3.10 Hands	Free Profi	le (HFP)	56
3.10.1	Detailed	Description	59
3.10.2	Macro De	efinition Documentation	59
	3.10.2.1	bt_hfp_ag_set_mic_gain	59
	3.10.2.2	bt_hfp_ag_set_speaker_volume	59
	3.10.2.3	bt_hfp_allocate_session_ag	60
	3.10.2.4	bt_hfp_allocate_session_hf	60
	3.10.2.5	bt_hfp_get_remote_address	60
	3.10.2.6	bt_hfp_has_active_call	60
	3.10.2.7	bt_hfp_has_alerting_call	60
	3.10.2.8	bt_hfp_has_dialing_call	61
	3.10.2.9	bt_hfp_has_held_call	61
	3.10.2.10	bt_hfp_has_incoming_call	61
	3.10.2.11	bt_hfp_has_outgoing_call	62
	3.10.2.12	2 bt_hfp_has_waiting_call	62
	3.10.2.13	B bt_hfp_hf_terminate	62
3.10.3	Typedef I	Documentation	62
	3.10.3.1	bt_hfp_find_remote_callback_pf	62
	3.10.3.2	bt_hfp_send_audio_callback_pf	63
	3.10.3.3	bt_hfp_session_callback_pf	63
3.10.4	Function	Documentation	64
	3.10.4.1	bt_hfp_activate_voice_recognition	64
	3.10.4.2	bt_hfp_ag_call_connected	64
	3.10.4.3	bt_hfp_ag_call_disconnected	64
	3.10.4.4	bt_hfp_ag_incoming_call	64
	3.10.4.5	bt_hfp_ag_outgoing_call	65
	3.10.4.6	bt_hfp_ag_outgoing_call_alerting	65
	3.10.4.7	bt_hfp_ag_reject_outgoing_call	65
	3.10.4.8	bt_hfp_ag_set_battery_level	66
	3.10.4.9	bt_hfp_ag_set_inband_ring	66
	3.10.4.10	bt_hfp_ag_set_operator_name	66
	3.10.4.11	bt_hfp_ag_set_roam_state	67

CONTENTS vii

		3.10.4.12	bt_hfp_ag_set_service_state	68
		3.10.4.13	bt_hfp_ag_set_signal_strength	68
		3.10.4.14	bt_hfp_ag_set_subscriber_number	68
		3.10.4.15	bt_hfp_allocate_session	69
		3.10.4.16	bt_hfp_connect	69
		3.10.4.17	bt_hfp_connect_audio	70
		3.10.4.18	bt_hfp_disconnect_audio	70
		3.10.4.19	bt_hfp_free_session	70
		3.10.4.20	bt_hfp_get_indicator_value	70
		3.10.4.21	bt_hfp_has_call_with_status	71
		3.10.4.22	bt_hfp_hf_answer	71
		3.10.4.23	bt_hfp_hf_dial_number	71
		3.10.4.24	bt_hfp_hf_disable_nrec	72
		3.10.4.25	bt_hfp_hf_disconnect	72
		3.10.4.26	bt_hfp_hf_enable_call_waiting_notification	72
		3.10.4.27	bt_hfp_hf_enable_calling_line_identification	72
		3.10.4.28	bt_hfp_hf_enable_voice_recognition	73
		3.10.4.29	bt_hfp_hf_find_ag	73
		3.10.4.30	bt_hfp_hf_get_subscriber_number	73
		3.10.4.31	bt_hfp_hf_hold	74
		3.10.4.32	bt_hfp_hf_query_operator	74
		3.10.4.33	bt_hfp_hf_redial	74
		3.10.4.34	bt_hfp_hf_refresh_call_list	75
		3.10.4.35	bt_hfp_hf_reject	75
		3.10.4.36	bt_hfp_init	75
		3.10.4.37	bt_hfp_listen	75
		3.10.4.38	bt_hfp_set_mic_gain	76
		3.10.4.39	bt_hfp_set_speaker_volume	76
3.11	Audio/\	/ideo Distr	ibution Protocol (AVDP)	77
	3.11.1	Detailed I	Description	86
	3.11.2	Macro De	efinition Documentation	86
		3.11.2.1	AVDTP_EVT_STREAM_ABORTED	86
		3.11.2.2	AVDTP_EVT_STREAM_CLOSED	86
		3.11.2.3	AVDTP_EVT_STREAM_CONFIGURED	86
		3.11.2.4	AVDTP_EVT_STREAM_OPENED	86
		3.11.2.5	AVDTP_EVT_STREAM_RECONFIGURED	87
		3.11.2.6	AVDTP_EVT_STREAM_STARTED	87
		3.11.2.7	AVDTP_EVT_STREAM_SUSPENDED	87
		3.11.2.8	AVDTP_STREAM_STATE_ABORTING	87
		3.11.2.9	AVDTP_STREAM_STATE_CLOSING	87

viii CONTENTS

	3.11.2.10 AVD	TP_STREAM_	STATE_IDL	E		 	 	 			87
	3.11.2.11 AVD	TP_STREAM_	STATE_ST	REAMI	NG .	 	 	 			87
	3.11.2.12 bt_a	vdtp_connect				 	 	 			87
	3.11.2.13 bt_a	vdtp_connect_e	ex			 	 	 			88
3.11.3	Typedef Docum	nentation				 	 	 			88
	3.11.3.1 bt_a	vdtp_codec_ha	ndler_fp .			 	 	 			88
	3.11.3.2 bt_a	vdtp_mgr_callb	ack_fp			 	 	 			89
3.11.4	Function Docu	mentation				 	 	 			91
	3.11.4.1 bt_a	vdtp_abort_stre	eam			 	 	 			91
	3.11.4.2 bt_a	vdtp_add_medi	a_rx_buffer			 	 	 			91
	3.11.4.3 bt_a	vdtp_add_medi	a_tx_buffer			 	 	 			91
	3.11.4.4 bt_a	vdtp_cancel_lis	ten			 	 	 			92
	3.11.4.5 bt_a	vdtp_close_stre	eam			 	 	 			92
	3.11.4.6 bt_a	vdtp_create_str	eam			 	 	 			93
	3.11.4.7 bt_a	vdtp_destroy_s	tream			 	 	 			93
	3.11.4.8 bt_a	vdtp_disconnec	t			 	 	 			93
	3.11.4.9 bt_a	vdtp_discover				 	 	 			94
	3.11.4.10 bt_a	vdtp_find_code	c			 	 	 			94
	3.11.4.11 bt_a	vdtp_get_all_ca	pabilities			 	 	 			95
	3.11.4.12 bt_a	vdtp_get_capab	oilities			 	 	 			95
	3.11.4.13 bt_a	vdtp_get_config	guration .			 	 	 			96
	3.11.4.14 bt_a	vdtp_get_hci_c	onnection			 	 	 			96
	3.11.4.15 bt_a	vdtp_get_mgr				 	 	 			97
	3.11.4.16 bt_a	vdtp_get_sep				 	 	 			97
	3.11.4.17 bt_a	vdtp_get_strear	m_codec_c	onfig .		 	 	 			97
	3.11.4.18 bt_a	vdtp_get_strear	m_codec_ty	/pe		 	 	 			98
	3.11.4.19 bt_a	vdtp_get_strear	m_config .			 	 	 			98
	3.11.4.20 bt_a	vdtp_get_strear	m_local_se	p_id .		 	 	 			99
	3.11.4.21 bt_a	vdtp_get_strear	m_remote_	address		 	 	 			99
	3.11.4.22 bt_a	vdtp_get_strear	m_remote_	sep_id		 	 	 			99
	3.11.4.23 bt_a	vdtp_get_strear	m_state .			 	 	 			99
	3.11.4.24 bt_a	vdtp_init				 	 	 			100
	3.11.4.25 bt_a	vdtp_listen				 	 	 			100
	3.11.4.26 bt_a	vdtp_open_stre	am			 	 	 			100
	3.11.4.27 bt_a	vdtp_reconfigur	e_stream			 	 	 			101
	3.11.4.28 bt_a	vdtp_register_c	allback .			 	 	 			101
	3.11.4.29 bt_a	vdtp_register_c	odec			 	 	 			103
	3.11.4.30 bt_a	vdtp_register_s	ep			 	 	 			103
	3.11.4.31 bt_a	vdtp_remove_n	nedia_rx_bu	uffer .		 	 	 			104
	3.11.4.32 bt_a	vdtp_remove_n	nedia_tx_bu	uffer .		 	 	 			104

CONTENTS

	3.11.4.33 bt_avdtp_report_delay
	3.11.4.34 bt_avdtp_security_control
	3.11.4.35 bt_avdtp_set_configuration
	3.11.4.36 bt_avdtp_start
	3.11.4.37 bt_avdtp_start_stream
	3.11.4.38 bt_avdtp_suspend_stream
	3.11.4.39 bt_avdtp_unregister_codec
3.12 Config	uration
3.12.1	Detailed Description
3.12.2	Macro Definition Documentation
	3.12.2.1 AVDTP_CODEC_CONFIG_BUFFER_LEN
	3.12.2.2 AVDTP_MAX_CMD_BUFFERS
	3.12.2.3 AVDTP_MAX_CMD_PARAM_LEN
	3.12.2.4 AVDTP_MAX_REMOTE_DEVICES
	3.12.2.5 AVDTP_MAX_SEP
	3.12.2.6 AVDTP_MAX_STREAMS
	3.12.2.7 AVDTP_MAX_TRANSPORT_CHANNELS
	3.12.2.8 AVDTP_MAX_TX_BUFFER_LEN
3.13 Advan	ced Audio Distribution Profile (A2DP)
3.13.1	Detailed Description
3.13.2	Macro Definition Documentation
	3.13.2.1 A2DP_EVT_STREAM_ABORTED
	3.13.2.2 A2DP_EVT_STREAM_CLOSED
	3.13.2.3 A2DP_EVT_STREAM_CONFIGURED
	3.13.2.4 A2DP_EVT_STREAM_OPENED
	3.13.2.5 A2DP_EVT_STREAM_RECONFIGURED
	3.13.2.6 A2DP_EVT_STREAM_STARTED
	3.13.2.7 A2DP_EVT_STREAM_SUSPENDED
	3.13.2.8 bt_a2dp_abort_stream
	3.13.2.9 bt_a2dp_add_media_rx_buffer
	3.13.2.10 bt_a2dp_add_media_tx_buffer
	3.13.2.11 bt_a2dp_cancel_listen
	3.13.2.12 bt_a2dp_close_stream
	3.13.2.13 bt_a2dp_connect
	3.13.2.14 bt_a2dp_connect_ex
	3.13.2.15 bt_a2dp_create_stream
	3.13.2.16 bt_a2dp_destroy_stream
	3.13.2.17 bt_a2dp_disconnect
	3.13.2.18 bt_a2dp_discover
	3.13.2.19 bt_a2dp_find_codec

X CONTENTS

		3.13.2.20 bt_a2dp_get_all_capabilities	123
		3.13.2.21 bt_a2dp_get_capabilities	124
		3.13.2.22 bt_a2dp_get_hci_connection	124
		3.13.2.23 bt_a2dp_get_stream_codec_config	125
		3.13.2.24 bt_a2dp_get_stream_codec_type	125
		3.13.2.25 bt_a2dp_get_stream_local_sep_id	126
		3.13.2.26 bt_a2dp_get_stream_remote_address	126
		3.13.2.27 bt_a2dp_get_stream_remote_sep_id	126
		3.13.2.28 bt_a2dp_get_stream_state	127
		3.13.2.29 bt_a2dp_listen	127
		3.13.2.30 bt_a2dp_reconfigure_stream	128
		3.13.2.31 bt_a2dp_register_sink	128
		3.13.2.32 bt_a2dp_register_source	128
		3.13.2.33 bt_a2dp_remove_media_rx_buffer	129
		3.13.2.34 bt_a2dp_remove_media_tx_buffer	129
		3.13.2.35 bt_a2dp_start_stream	130
		3.13.2.36 bt_a2dp_suspend_stream	130
		3.13.2.37 bt_avdtp_get_stream_config	131
	3.13.3	Typedef Documentation	131
		3.13.3.1 bt_a2dp_find_server_callback_fp	131
		3.13.3.2 bt_a2dp_mgr_callback_fp	131
	3.13.4	Function Documentation	133
		3.13.4.1 bt_a2dp_find_sink	133
		3.13.4.2 bt_a2dp_find_source	133
		3.13.4.3 bt_a2dp_get_mgr	134
		3.13.4.4 bt_a2dp_init	134
		3.13.4.5 bt_a2dp_open_and_start_stream	134
		3.13.4.6 bt_a2dp_register_aac_codec	135
		3.13.4.7 bt_a2dp_register_callback	135
		3.13.4.8 bt_a2dp_register_mpeg_codec	137
		3.13.4.9 bt_a2dp_start	
3.14		/ideo Control Transport Protocol (AVCTP)	
		Detailed Description	
	3.14.2	Typedef Documentation	
		3.14.2.1 bt_avctp_mgr_callback_fp	140
	3.14.3	Function Documentation	
		3.14.3.1 bt_avctp_cancel_command	
		3.14.3.2 bt_avctp_cancel_listen	
		3.14.3.3 bt_avctp_cancel_response	
		3.14.3.4 bt_avctp_connect	141

CONTENTS xi

		3.14.3.5 bt_avctp_create_channel
		3.14.3.6 bt_avctp_create_outgoing_channel
		3.14.3.7 bt_avctp_destroy_channel
		3.14.3.8 bt_avctp_disconnect
		3.14.3.9 bt_avctp_get_channel_remote_address
		3.14.3.10 bt_avctp_get_channel_state
		3.14.3.11 bt_avctp_get_hci_connection
		3.14.3.12 bt_avctp_get_mgr
		3.14.3.13 bt_avctp_init
		3.14.3.14 bt_avctp_listen
		3.14.3.15 bt_avctp_send_command
		3.14.3.16 bt_avctp_send_response
		3.14.3.17 bt_avctp_set_callback
3.15	Configu	ration
3	3.15.1	Detailed Description
3	3.15.2	Macro Definition Documentation
		3.15.2.1 AVCTP_ALLOCATE_BUFFERS_VARS
		3.15.2.2 AVCTP_MAX_CHANNELS
		3.15.2.3 AVCTP_MAX_TRANSPORT_CHANNELS
3.16 A	Audio/V	/ideo Remote Control Profile (AVRCP)
3	3.16.1	Detailed Description
3	3.16.2	Typedef Documentation
		$3.16.2.1  bt\_avrcp\_find\_callback\_fp  .  .  .  .  .  .  .  .  .  $
		$3.16.2.2  bt\_avrcp\_mgr\_callback\_fp \ . \ . \ . \ . \ . \ . \ . \ . \ . \ $
3	3.16.3	Function Documentation
		$3.16.3.1  bt\_avrcp\_add\_to\_now\_playing \ . \ . \ . \ . \ . \ . \ . \ . \ . \ $
		3.16.3.2 bt_avrcp_cancel_find
		3.16.3.3 bt_avrcp_cancel_listen
		3.16.3.4 bt_avrcp_connect
		3.16.3.5 bt_avrcp_create_channel
		3.16.3.6 bt_avrcp_create_outgoing_channel
		3.16.3.7 bt_avrcp_destroy_channel
		3.16.3.8 bt_avrcp_disconnect
		3.16.3.9 bt_avrcp_find_controller
		3.16.3.10 bt_avrcp_find_target
		3.16.3.11 bt_avrcp_find_targets
		3.16.3.12 bt_avrcp_get_browsing_channel_state
		3.16.3.13 bt_avrcp_get_channel_remote_address
		3.16.3.14 bt_avrcp_get_company_id_list
		3.16.3.15 bt_avrcp_get_control_channel_state

xii CONTENTS

		3.16.3.16 bt_avrcp_get_current_player_application_setting_value	7
		3.16.3.17 bt_avrcp_get_element_attributes	8
		3.16.3.18 bt_avrcp_get_hci_connection	9
		3.16.3.19 bt_avrcp_get_mgr	9
		3.16.3.20 bt_avrcp_get_play_status	9
		3.16.3.21 bt_avrcp_get_player_application_setting_attr_text	9
		3.16.3.22 bt_avrcp_get_player_application_setting_value_text	'1
		3.16.3.23 bt_avrcp_get_subuint_info	1
		3.16.3.24 bt_avrcp_get_supported_event_id_list	'1
		3.16.3.25 bt_avrcp_get_unit_info	2
		3.16.3.26 bt_avrcp_inform_battery_status	2
		3.16.3.27 bt_avrcp_inform_displayable_character_set	3
		3.16.3.28 bt_avrcp_init_controller	
		3.16.3.29 bt_avrcp_init_target	3
		3.16.3.30 bt_avrcp_list_player_application_setting_attributes	4
		3.16.3.31 bt_avrcp_list_player_application_setting_values	4
		3.16.3.32 bt_avrcp_listen	6
		3.16.3.33 bt_avrcp_play_item	
		3.16.3.34 bt_avrcp_register_notification	6
		3.16.3.35 bt_avrcp_register_notifications	8
		3.16.3.36 bt_avrcp_send_button_click	9
		3.16.3.37 bt_avrcp_send_cmd	9
		3.16.3.38 bt_avrcp_send_panel_control	9
		3.16.3.39 bt_avrcp_send_press_panel_control	0
		3.16.3.40 bt_avrcp_send_release_panel_control	0
		3.16.3.41 bt_avrcp_send_simple_panel_cmd	0
		3.16.3.42 bt_avrcp_set_absolute_volume	1
		3.16.3.43 bt_avrcp_set_addressed_player	1
		3.16.3.44 bt_avrcp_set_player_application_setting_value	2
		3.16.3.45 bt_avrcp_start	2
		3.16.3.46 bt_avrcp_tg_send_element_attributes	2
		3.16.3.47 bt_avrcp_tg_set_absolute_volume	2
		3.16.3.48 bt_avrcp_tg_set_battery_status	3
		3.16.3.49 bt_avrcp_tg_set_current_track	3
		3.16.3.50 bt_avrcp_tg_set_play_status	3
		3.16.3.51 bt_avrcp_tg_set_system_status	4
3.17	Configu	uration	5
	3.17.1	Detailed Description	5
	3.17.2	Macro Definition Documentation	6
		3.17.2.1 AVRCP_CMD_TIMEOUT	6

CONTENTS xiii

		3.17.2.2 AVRCP_MAX_CHANNELS
		3.17.2.3 AVRCP_MAX_CMD_BUFFERS
		3.17.2.4 AVRCP_MAX_CMD_PARAM_LEN
		3.17.2.5 AVRCP_MAX_DEVICE_NAME_LEN
		3.17.2.6 AVRCP_MAX_SEARCH_RESULTS
3.18	ATT .	
	3.18.1	Detailed Description
	3.18.2	Macro Definition Documentation
		3.18.2.1 ATT_ERROR_UNLIKELY_ERROR
		3.18.2.2 ATT_ERROR_UNSUPPORTED_GROUP_TYPE
3.19	ATT Se	erver
	3.19.1	Detailed Description
	3.19.2	Typedef Documentation
		3.19.2.1 bt_att_mgr_callback_fp
	3.19.3	Function Documentation
		3.19.3.1 bt_att_get_mgr
		3.19.3.2 bt_att_get_remote_address
		3.19.3.3 bt_att_init
		3.19.3.4 bt_att_server_authorize_access
		3.19.3.5 bt_att_server_disconnect
		3.19.3.6 bt_att_server_indicate
		3.19.3.7 bt_att_server_indicate_to_all
		3.19.3.8 bt_att_server_notify
		3.19.3.9 bt_att_server_register_callback
		3.19.3.10 bt_att_server_start
		3.19.3.11 bt_att_server_stop
		3.19.3.12 bt_att_server_write_attribute_value
		3.19.3.13 bt_att_update_conn_parameters
3.20	Configu	uration
	3.20.1	Detailed Description
	3.20.2	Macro Definition Documentation
		3.20.2.1 ATT_MAX_CLIENTS
		3.20.2.2 ATT_MAX_FOUND_ATTRIBUTES
		3.20.2.3 ATT_MAX_QUEUED_WRITE_BUFFER_SIZE
3.21	ATT CI	ient
	3.21.1	Detailed Description
	3.21.2	Function Documentation
		3.21.2.1 bt_att_client_accept_conn_param_update
		3.21.2.2 bt_att_client_allocate_session
		3.21.2.3 bt_att_client_connect

XIV

		3.21.2.4 bt_att_client_disconnect
		3.21.2.5 bt_att_client_exchange_mtu
		3.21.2.6 bt_att_client_execute_write
		3.21.2.7 bt_att_client_find_by_type_value
		3.21.2.8 bt_att_client_find_info
		3.21.2.9 bt_att_client_free_session
		3.21.2.10 bt_att_client_get_remote_address
		3.21.2.11 bt_att_client_init
		3.21.2.12 bt_att_client_prepare_write
		3.21.2.13 bt_att_client_read
		3.21.2.14 bt_att_client_read_blob
		3.21.2.15 bt_att_client_read_by_group_type
		3.21.2.16 bt_att_client_read_by_group_type_80
		3.21.2.17 bt_att_client_read_by_type
		3.21.2.18 bt_att_client_read_by_type_80
		3.21.2.19 bt_att_client_read_multiple
		3.21.2.20 bt_att_client_reject_conn_param_update
		3.21.2.21 bt_att_client_signed_write
		3.21.2.22 bt_att_client_write
		3.21.2.23 bt_att_client_write_cmd
		3.21.2.24 bt_att_get_client_mgr
3.22	Configu	uration
	3.22.1	Detailed Description
	3.22.2	Macro Definition Documentation
		3.22.2.1 ATT_CLIENT_MAX_SESSIONS
3.23	GATT	
	3.23.1	Detailed Description
3.24	GATT S	Server
	3.24.1	Detailed Description
	3.24.2	Macro Definition Documentation
		3.24.2.1 bt_gatt_server_authorize_access
		3.24.2.2 bt_gatt_server_start
	3.24.3	Function Documentation
		3.24.3.1 bt_gatt_server_indicate_char_value
		3.24.3.2 bt_gatt_server_indicate_char_value_80
		3.24.3.3 bt_gatt_server_notify_char_value
		3.24.3.4 bt_gatt_server_notify_char_value_80
		3.24.3.5 bt_gatt_server_register_callback
		3.24.3.6 bt_gatt_server_register_listener
		3.24.3.7 bt_gatt_server_start_ex

CONTENTS xv

	3.24.3.8 bt_gatt_server_unregister_listener	20
	3.24.3.9 bt_gatt_server_write_char_value	20
	3.24.3.10 bt_gatt_server_write_char_value_80	21
3.25 GATT	Client	22
3.25.1	Detailed Description	26
3.25.2	Macro Definition Documentation	26
	3.25.2.1 bt_gatt_client_accept_conn_param_update	26
	3.25.2.2 bt_gatt_client_reject_conn_param_update	26
3.25.3	Function Documentation	27
	3.25.3.1 bt_gatt_client_allocate_session	27
	3.25.3.2 bt_gatt_client_connect	27
	3.25.3.3 bt_gatt_client_disconnect	27
	3.25.3.4 bt_gatt_client_discover_all_chars_ex	28
	3.25.3.5 bt_gatt_client_discover_all_services_ex	28
	3.25.3.6 bt_gatt_client_discover_by_service_uuid	28
	3.25.3.7 bt_gatt_client_discover_by_service_uuid_80	29
	3.25.3.8 bt_gatt_client_discover_char_by_uuid	29
	3.25.3.9 bt_gatt_client_discover_char_by_uuid_80	29
	3.25.3.10 bt_gatt_client_discover_char_descriptors	30
	3.25.3.11 bt_gatt_client_find_included_services_ex	30
	3.25.3.12 bt_gatt_client_free_sessions	31
	3.25.3.13 bt_gatt_client_get_hci_connection	31
	3.25.3.14 bt_gatt_client_init	31
	3.25.3.15 bt_gatt_client_read_by_char_uuid	31
	3.25.3.16 bt_gatt_client_read_by_char_uuid_80	32
	3.25.3.17 bt_gatt_client_read_char_descriptor	32
	3.25.3.18 bt_gatt_client_read_char_long_descriptor	32
	3.25.3.19 bt_gatt_client_read_char_long_value	33
	3.25.3.20 bt_gatt_client_read_char_value	33
	3.25.3.21 bt_gatt_client_read_multiple_char_values	33
	3.25.3.22 bt_gatt_client_set_char_value	34
	3.25.3.23 bt_gatt_client_write_char_descriptor	34
	3.25.3.24 bt_gatt_client_write_char_long_descriptor	34
	3.25.3.25 bt_gatt_client_write_char_long_value	35
	3.25.3.26 bt_gatt_client_write_char_value	35
	3.25.3.27 bt_gatt_exchange_mtu	35
	3.25.3.28 bt_gatt_get_client_mgr	36
3.26 Config	uration	37
3.27 Securi	ty Manager	39
3.28 OEM -	HCI Communication Interface	40

xvi CONTENTS

	3.28.1	Detailed Description
	3.28.2	Typedef Documentation
		3.28.2.1 bt_oem_recv_callback_fp
		3.28.2.2 bt_oem_send_callback_fp
	3.28.3	Function Documentation
		3.28.3.1 bt_oem_recv
		3.28.3.2 bt_oem_send
3.29	OEM -	Timer Interface
	3.29.1	Detailed Description
	3.29.2	Typedef Documentation
		3.29.2.1 bt_timer_callback_fp
	3.29.3	Function Documentation
		3.29.3.1 bt_oem_timer_clear
		3.29.3.2 bt_oem_timer_set
3.30	OEM -	Non-volatile Storage Interface
	3.30.1	Detailed Description
	3.30.2	Function Documentation
		3.30.2.1 bt_oem_storage_get_capacity
		3.30.2.2 bt_oem_storage_read
		3.30.2.3 bt_oem_storage_start
		3.30.2.4 bt_oem_storage_stop
		3.30.2.5 bt_oem_storage_write
3.31	OEM -	Logging Interface
	3.31.1	Detailed Description
	3.31.2	Function Documentation
		3.31.2.1 bt_oem_log_write
3.32	Vendor	specific extensions to HCI
	3.32.1	Detailed Description
3.33	CSR .	
	3.33.1	Detailed Description
	3.33.2	Function Documentation
		3.33.2.1 btx_csr_autobaud
		3.33.2.2 btx_csr_bc7_sel_host_interface_h4
		3.33.2.3 btx_csr_enable_tx
		3.33.2.4 btx_csr_exec_script
		3.33.2.5 btx_csr_get_cached_temperature
		3.33.2.6 btx_csr_get_rssi_acl
		3.33.2.7 btx_csr_init
		3.33.2.8 btx_csr_patch_controller
		3.33.2.9 btx_csr_set_ps_vars

CONTENTS xvii

		3.33.2.10 btx_csr_set_ps_vars_ex	51
		3.33.2.11 btx_csr_warm_reset	51
		3.33.2.12 btx_csr_warm_reset_ex	51
4	Data	Structure Documentation 25	53
7	4.1	bt_spp_port_t::_bt_spp_port_flags_t Struct Reference	
	4.1	bt_a2dp_aac_config_t Struct Reference	
	4.2		
	4.3	bt_a2dp_event_t Union Reference	
		4.3.1 Detailed Description	
	4.4	bt_a2dp_evt_open_and_start_stream_completed_t Struct Reference	
		4.4.1 Detailed Description	
		4.4.2 Field Documentation	
		4.4.2.1 err_code	
		4.4.2.2 failed_cmd	
	4.5	bt_a2dp_mgr_t Struct Reference	
		4.5.1 Detailed Description	56
		4.5.2 Field Documentation	56
		4.5.2.1 state	56
	4.6	bt_a2dp_mpeg_config_t Struct Reference	56
	4.7	bt_a2dp_sbc_config_t Struct Reference	56
	4.8	bt_a2dp_sbc_packet_info_t Struct Reference	56
	4.9	bt_att_attribute_t Struct Reference	56
	4.10	bt_att_client_evt_conn_param_update_completed_t Struct Reference	56
		4.10.1 Detailed Description	57
	4.11	bt_att_client_evt_conn_param_update_t Struct Reference	57
		4.11.1 Detailed Description	57
	4.12	bt_att_client_evt_execute_write_response_t Struct Reference	57
		4.12.1 Detailed Description	57
	4.13	bt_att_client_evt_find_by_type_value_response_t Struct Reference	58
		4.13.1 Detailed Description	58
	4.14	bt_att_client_evt_info_response_t Struct Reference	58
		4.14.1 Detailed Description	58
	4.15	bt_att_client_evt_mtu_response_t Struct Reference	58
		4.15.1 Detailed Description	
	4.16	bt_att_client_evt_prepare_write_response_t Struct Reference	
		4.16.1 Detailed Description	
	4 17	bt_att_client_evt_read_blob_response_t Struct Reference	
	,	4.17.1 Detailed Description	
	<u>4</u> 12	bt_att_client_evt_read_by_group_type_response_t Struct Reference	
	7.10		
		4.18.1 Detailed Description	JU

xviii CONTENTS

4.19	bt_att_client_evt_read_by_type_response_t Struct Reference	260
	4.19.1 Detailed Description	261
4.20	bt_att_client_evt_read_multiple_response_t Struct Reference	261
	4.20.1 Detailed Description	261
4.21	bt_att_client_evt_read_response_t Struct Reference	261
	4.21.1 Detailed Description	261
4.22	bt_att_client_evt_t Union Reference	262
	4.22.1 Detailed Description	262
4.23	bt_att_client_evt_value_indication_t Struct Reference	262
	4.23.1 Detailed Description	263
4.24	bt_att_client_evt_value_notification_t Struct Reference	263
	4.24.1 Detailed Description	263
4.25	bt_att_client_evt_write_response_t Struct Reference	263
	4.25.1 Detailed Description	264
4.26	bt_att_client_mgr_t Struct Reference	264
4.27	bt_att_client_session_t Struct Reference	264
4.28	bt_att_client_uuid_t Struct Reference	264
	4.28.1 Detailed Description	264
	4.28.2 Field Documentation	264
	4.28.2.1 len	264
4.29	bt_att_enum_context_t Struct Reference	264
4.30	bt_att_evt_attr_indication_sent_t Struct Reference	264
	4.30.1 Detailed Description	265
4.31	bt_att_evt_attr_notification_sent_t Struct Reference	265
	4.31.1 Detailed Description	265
4.32	bt_att_evt_attr_value_changed_t Struct Reference	265
	4.32.1 Detailed Description	
4.33	bt_att_evt_attr_value_read_t Struct Reference	
	4.33.1 Detailed Description	
4.34	bt_att_evt_authorization_requested_t Struct Reference	
	4.34.1 Detailed Description	
	bt_att_evt_data_received_t Struct Reference	
4.36	bt_att_evt_session_connected_t Struct Reference	
	4.36.1 Detailed Description	
4.37	bt_att_evt_session_disconnected_t Struct Reference	
	4.37.1 Detailed Description	
4.38	bt_att_evt_tran_timeout_t Struct Reference	
	4.38.1 Detailed Description	
4.39	bt_att_find_by_type_value_response_t Struct Reference	
	4.39.1 Detailed Description	268

CONTENTS xix

4.40	bt_att_find_info_response_t Struct Reference	268
	4.40.1 Detailed Description	268
	4.40.2 Field Documentation	269
	4.40.2.1 format	269
4.41	bt_att_found_attribyte_t Struct Reference	269
4.42	bt_att_listener_t Struct Reference	269
4.43	bt_att_mgr_t Struct Reference	269
	4.43.1 Detailed Description	269
4.44	bt_att_read_by_group_type_response_t Struct Reference	269
	4.44.1 Detailed Description	269
4.45	bt_att_read_by_type_response_t Struct Reference	270
	4.45.1 Detailed Description	270
4.46	bt_att_session_evt_authorization_requested_t Struct Reference	270
4.47	bt_att_session_evt_packet_sent_t Struct Reference	270
4.48	bt_att_session_t Struct Reference	270
4.49	bt_av_add_to_now_playing_t Struct Reference	270
	4.49.1 Detailed Description	270
4.50	bt_av_battery_status_of_ct_t Struct Reference	271
	4.50.1 Detailed Description	271
	4.50.2 Field Documentation	271
	4.50.2.1 status	271
4.51	bt_av_capability_company_id_t Struct Reference	271
	4.51.1 Detailed Description	272
4.52	bt_av_capability_event_id_t Struct Reference	272
	4.52.1 Detailed Description	272
4.53	bt_av_command_t Struct Reference	272
4.54	bt_av_displayable_character_set_t Struct Reference	272
	4.54.1 Detailed Description	273
4.55	bt_av_element_attribute_t Struct Reference	273
	4.55.1 Detailed Description	273
4.56	bt_av_element_attributes_t Struct Reference	273
	4.56.1 Detailed Description	273
4.57	bt_av_element_id_t Struct Reference	274
	4.57.1 Detailed Description	274
4.58	bt_av_get_element_attributes_t Struct Reference	274
	4.58.1 Detailed Description	274
	4.58.2 Field Documentation	274
	4.58.2.1 attributes	274
4.59	bt_av_notification_addressed_player_changed_t Struct Reference	275
	4.59.1 Detailed Description	275

CONTENTS

4.60	bt_av_notification_app_setting_changed_t Struct Reference	275
4.61	bt_av_notification_battery_status_t Struct Reference	275
	4.61.1 Detailed Description	275
	4.61.2 Field Documentation	276
	4.61.2.1 status	276
4.62	bt_av_notification_playback_pos_changed_t Struct Reference	276
	4.62.1 Detailed Description	276
4.63	bt_av_notification_playback_status_changed_t Struct Reference	276
	4.63.1 Detailed Description	276
	4.63.2 Field Documentation	277
	4.63.2.1 status	277
4.64	bt_av_notification_system_status_changed_t Struct Reference	277
	4.64.1 Detailed Description	277
4.65	bt_av_notification_t Struct Reference	277
	4.65.1 Detailed Description	278
4.66	bt_av_notification_track_changed_t Struct Reference	279
	4.66.1 Detailed Description	279
4.67	bt_av_notification_uids_changed_t Struct Reference	279
	4.67.1 Detailed Description	279
4.68	bt_av_notification_volume_changed_t Struct Reference	280
	4.68.1 Detailed Description	280
4.69	bt_av_play_item_t Struct Reference	280
	4.69.1 Detailed Description	280
4.70	bt_av_play_status_t Struct Reference	280
	4.70.1 Detailed Description	281
	4.70.2 Field Documentation	281
	4.70.2.1 play_status	281
4.71	bt_av_player_setting_current_values_t Struct Reference	281
	4.71.1 Detailed Description	281
4.72	bt_av_player_setting_values_t Struct Reference	282
	4.72.1 Detailed Description	282
4.73	bt_av_player_setting_values_text_t Struct Reference	282
	4.73.1 Detailed Description	282
4.74	bt_av_player_settings_t Struct Reference	282
	4.74.1 Detailed Description	283
4.75	bt_av_player_settings_text_t Struct Reference	283
	4.75.1 Detailed Description	283
4.76	bt_av_player_text_t Struct Reference	283
4.77	bt_av_register_notification_t Struct Reference	283
	4.77.1 Detailed Description	284

CONTENTS xxi

	4.77.2 Field Documentation	284
	4.77.2.1 event_id	284
4.78	bt_av_response_t Struct Reference	284
	4.78.1 Detailed Description	285
	4.78.2 Field Documentation	285
	4.78.2.1 ctype	285
4.79	bt_av_set_absolute_volume_t Struct Reference	285
	4.79.1 Detailed Description	286
4.80	bt_av_set_addressed_player_t Struct Reference	286
	4.80.1 Detailed Description	286
4.81	bt_avctp_channel_t Struct Reference	286
	4.81.1 Detailed Description	286
4.82	bt_avctp_event_t Union Reference	286
	4.82.1 Detailed Description	287
4.83	bt_avctp_evt_channel_connected_t Struct Reference	287
	4.83.1 Detailed Description	287
4.84	bt_avctp_evt_channel_disconnected_t Struct Reference	287
	4.84.1 Detailed Description	288
4.85	bt_avctp_evt_command_cancelled_t Struct Reference	288
	4.85.1 Detailed Description	288
4.86	bt_avctp_evt_command_received_t Struct Reference	288
	4.86.1 Detailed Description	288
4.87	bt_avctp_evt_command_sent_t Struct Reference	288
	4.87.1 Detailed Description	289
4.88	bt_avctp_evt_connection_failed_t Struct Reference	289
	4.88.1 Detailed Description	289
4.89	bt_avctp_evt_response_cancelled_t Struct Reference	289
	4.89.1 Detailed Description	289
4.90	bt_avctp_evt_response_received_t Struct Reference	290
	4.90.1 Detailed Description	290
4.91	bt_avctp_evt_response_sent_t Struct Reference	290
	4.91.1 Detailed Description	290
4.92	bt_avctp_message_t Struct Reference	290
	4.92.1 Detailed Description	290
4.93	bt_avctp_mgr_t Struct Reference	291
	4.93.1 Detailed Description	291
	4.93.2 Field Documentation	291
	4.93.2.1 flags	291
	4.93.2.2 state	291
4.94	bt_avctp_transport_t Struct Reference	291

xxii CONTENTS

4.94.1 Detailed Description
4.95 bt_avdtp_codec_op_decode_t Struct Reference
4.96 bt_avdtp_codec_op_encode_t Struct Reference
4.97 bt_avdtp_codec_op_param_t Union Reference
4.97.1 Detailed Description
4.98 bt_avdtp_codec_op_parse_config_t Struct Reference
4.98.1 Detailed Description
4.98.2 Field Documentation
4.98.2.1 codec_config
4.98.2.2 offset
4.99 bt_avdtp_codec_op_parse_packet_t Struct Reference
4.100bt_avdtp_codec_op_serialize_config_t Struct Reference
4.100.1 Detailed Description
4.100.2 Field Documentation
4.100.2.1 codec_config
4.100.2.2 offset
4.101bt_avdtp_codec_t Struct Reference
4.101.1 Detailed Description
4.101.2 Field Documentation
4.101.2.1 codec_type
4.102bt_avdtp_event_t Union Reference
4.102.1 Detailed Description
4.103bt_avdtp_evt_abort_stream_requested_t Struct Reference
4.103.1 Detailed Description
4.103.2 Field Documentation
4.103.2.1 err_code
4.104bt_avdtp_evt_close_stream_completed_t Struct Reference
4.104.1 Detailed Description
4.104.2 Field Documentation
4.104.2.1 err_code
4.105bt_avdtp_evt_close_stream_requested_t Struct Reference
4.105.1 Detailed Description
4.105.2 Field Documentation
4.105.2.1 err_code
4.106bt_avdtp_evt_ctrl_channel_connected_t Struct Reference
4.106.1 Detailed Description
4.107bt_avdtp_evt_ctrl_channel_disconnected_t Struct Reference
4.107.1 Detailed Description
4.108bt_avdtp_evt_delay_report_completed_t Struct Reference
4.108.1 Detailed Description

CONTENTS xxiii

4.108.2 Field Documentation	299
4.108.2.1 err_code	299
4.109bt_avdtp_evt_discover_completed_t Struct Reference	299
4.109.1 Detailed Description	299
4.109.2 Field Documentation	299
4.109.2.1 err_code	299
4.110bt_avdtp_evt_get_sep_capabilities_completed_t Struct Reference	300
4.110.1 Detailed Description	300
4.110.2 Field Documentation	300
4.110.2.1 err_code	300
4.111bt_avdtp_evt_get_stream_configuration_completed_t Struct Reference	300
4.111.1 Detailed Description	300
4.112bt_avdtp_evt_media_packet_received_t Struct Reference	301
4.112.1 Detailed Description	301
4.113bt_avdtp_evt_media_packet_send_failed_t Struct Reference	301
4.113.1 Detailed Description	301
4.114bt_avdtp_evt_media_packet_sent_t Struct Reference	301
4.114.1 Detailed Description	302
4.115bt_avdtp_evt_open_stream_completed_t Struct Reference	302
4.115.1 Detailed Description	302
4.115.2 Field Documentation	302
4.115.2.1 err_code	302
4.116bt_avdtp_evt_open_stream_requested_t Struct Reference	302
4.116.1 Detailed Description	303
4.116.2 Field Documentation	303
4.116.2.1 err_code	303
4.117bt_avdtp_evt_reconfigure_stream_requested_t Struct Reference	303
4.117.1 Detailed Description	303
4.117.2 Field Documentation	304
4.117.2.1 err_code	304
4.118bt_avdtp_evt_sep_capabilities_received_t Struct Reference	304
4.118.1 Detailed Description	304
4.119bt_avdtp_evt_sep_info_received_t Struct Reference	304
4.119.1 Detailed Description	305
4.119.2 Field Documentation	305
4.119.2.1 media_type	305
4.120bt_avdtp_evt_set_stream_configuration_completed_t Struct Reference	305
4.120.1 Detailed Description	305
4.120.2 Field Documentation	305
4.120.2.1 err_code	305

xxiv CONTENTS

4.121bt_avdtp_evt_set_stream_configuration_requested_t Struct Reference	306
4.121.1 Detailed Description	306
4.121.2 Field Documentation	306
4.121.2.1 err_code	306
4.122bt_avdtp_evt_start_stream_completed_t Struct Reference	306
4.122.1 Detailed Description	307
4.122.2 Field Documentation	307
4.122.2.1 err_code	307
4.123bt_avdtp_evt_start_stream_requested_t Struct Reference	307
4.123.1 Detailed Description	307
4.123.2 Field Documentation	307
4.123.2.1 err_code	307
4.124bt_avdtp_evt_stream_aborted_t Struct Reference	307
4.124.1 Detailed Description	308
4.125bt_avdtp_evt_stream_closed_t Struct Reference	308
4.125.1 Detailed Description	308
4.126bt_avdtp_evt_stream_configured_t Struct Reference	308
4.126.1 Detailed Description	308
4.127bt_avdtp_evt_stream_opened_t Struct Reference	309
4.127.1 Detailed Description	309
4.128bt_avdtp_evt_stream_reconfigure_completed_t Struct Reference	309
4.128.1 Detailed Description	309
4.128.2 Field Documentation	309
4.128.2.1 err_code	309
4.129bt_avdtp_evt_stream_reconfigured_t Struct Reference	310
4.129.1 Detailed Description	310
4.130bt_avdtp_evt_stream_security_control_completed_t Struct Reference	310
4.130.1 Detailed Description	310
4.130.2 Field Documentation	310
4.130.2.1 err_code	310
4.131bt_avdtp_evt_stream_started_t Struct Reference	310
4.131.1 Detailed Description	311
4.132bt_avdtp_evt_stream_suspended_t Struct Reference	311
4.132.1 Detailed Description	311
4.133bt_avdtp_evt_suspend_stream_completed_t Struct Reference	311
4.133.1 Detailed Description	311
4.133.2 Field Documentation	312
4.133.2.1 err_code	312
4.134bt_avdtp_evt_suspend_stream_requested_t Struct Reference	312
4.134.1 Detailed Description	312

CONTENTS xxv

4.134.2 Field Documentation	312
4.134.2.1 err_code	312
4.135bt_avdtp_mgr_t Struct Reference	312
4.135.1 Detailed Description	313
4.135.2 Field Documentation	313
4.135.2.1 flags	313
4.135.2.2 state	313
4.136bt_avdtp_sep_capabilities_t Struct Reference	314
4.136.1 Detailed Description	314
4.136.2 Field Documentation	315
4.136.2.1 categories	315
4.136.2.2 codec_type	315
4.136.2.3 cp_specific	315
4.136.2.4 cp_specific_len	315
4.136.2.5 cp_type	315
4.136.2.6 header_compression_options	316
4.136.2.7 max_parity_code_packets	316
4.136.2.8 max_recovery_window	316
4.136.2.9 media_type	316
4.136.2.10multiplexing_options	316
4.136.2.11recovery_type	316
4.137bt_avdtp_sep_t Struct Reference	317
4.137.1 Detailed Description	317
4.137.2 Field Documentation	317
4.137.2.1 state	317
4.137.2.2 type	317
4.138bt_avdtp_stream_t Struct Reference	317
4.138.1 Detailed Description	318
4.138.2 Field Documentation	318
4.138.2.1 flags	318
4.138.2.2 sessions	319
4.138.2.3 state	319
4.139bt_avdtp_transport_channel_t Struct Reference	319
4.139.1 Detailed Description	319
4.139.2 Field Documentation	320
4.139.2.1 ref_count	320
4.139.2.2 type	320
4.140bt_avdtp_transport_session_t Struct Reference	320
4.140.1 Detailed Description	320
4.140.2 Field Documentation	320

XXVI

4.140.2.1 type
4.141bt_avrcp_channel_t Struct Reference
4.141.1 Detailed Description
4.142bt_avrcp_device_t Struct Reference
4.143bt_avrcp_event_t Union Reference
4.143.1 Detailed Description
4.143.2 Field Documentation
4.143.2.1 notification
4.144bt_avrcp_evt_channel_connected_t Struct Reference
4.144.1 Detailed Description
4.145bt_avrcp_evt_channel_disconnected_t Struct Reference
4.145.1 Detailed Description
4.146bt_avrcp_evt_connection_failed_t Struct Reference
4.146.1 Detailed Description
4.147bt_avrcp_evt_panel_command_received_t Struct Reference
4.147.1 Detailed Description
4.148bt_avrcp_evt_panel_response_received_t Struct Reference
4.148.1 Detailed Description
4.149bt_avrcp_evt_register_events_completed_t Struct Reference
4.150bt_avrcp_evt_search_completed_t Struct Reference
4.150.1 Detailed Description
4.151bt_avrcp_mgr_t Struct Reference
4.151.1 Detailed Description
4.151.2 Field Documentation
4.151.2.1 battery_status
4.151.2.2 company_id
4.151.2.3 flags
4.151.2.4 state
4.151.2.5 supported_events
4.152bt_cp_header_t Struct Reference
4.153bt_gatt_client_char_declaration_t Struct Reference
4.153.1 Detailed Description
4.154bt_gatt_client_char_descriptor_t Struct Reference
4.154.1 Detailed Description
4.155bt_gatt_client_char_value_t Struct Reference
4.155.1 Detailed Description
4.156bt_gatt_client_evt_conn_param_update_completed_t Struct Reference
4.156.1 Detailed Description
4.157bt_gatt_client_evt_conn_param_update_t Struct Reference
4.157.1 Detailed Description

CONTENTS xxvii

4.158bt_gatt_client_evt_discover_all_chars_completed_t Struct Reference
4.158.1 Detailed Description
4.158.2 Field Documentation
4.158.2.1 chars
4.159bt_gatt_client_evt_discover_all_services_completed_t Struct Reference
4.159.1 Detailed Description
4.159.2 Field Documentation
4.159.2.1 services
4.160bt_gatt_client_evt_discover_char_by_uuid_completed_t Struct Reference
4.160.1 Detailed Description
4.160.2 Field Documentation
4.160.2.1 chars
4.161bt_gatt_client_evt_discover_descriptors_completed_t Struct Reference
4.161.1 Detailed Description
4.161.2 Field Documentation
4.161.2.1 descriptors
4.162bt_gatt_client_evt_discover_service_by_uuid_completed_t Struct Reference
4.162.1 Detailed Description
4.162.2 Field Documentation
4.162.2.1 services
4.163bt_gatt_client_evt_exchange_mtu_completed_t Struct Reference
4.163.1 Detailed Description
4.164bt_gatt_client_evt_find_included_completed_t Struct Reference
4.164.1 Detailed Description
4.164.2 Field Documentation
4.164.2.1 services
4.165bt_gatt_client_evt_profile_found_t Struct Reference
4.166bt_gatt_client_evt_read_by_char_uuid_completed_t Struct Reference
4.166.1 Detailed Description
4.166.2 Field Documentation
4.166.2.1 values
4.167bt_gatt_client_evt_read_char_descriptor_completed_t Struct Reference
4.167.1 Detailed Description
4.168bt_gatt_client_evt_read_char_long_descriptor_completed_t Struct Reference
4.168.1 Detailed Description
4.169bt_gatt_client_evt_read_char_long_value_completed_t Struct Reference
4.169.1 Detailed Description
4.170bt_gatt_client_evt_read_char_value_completed_t Struct Reference
4.170.1 Detailed Description
4.171 bt_gatt_client_evt_read_multiple_char_values_completed_t Struct Reference

xxviii CONTENTS

4.171.1 Detailed Description	336
4.172bt_gatt_client_evt_t Union Reference	337
4.172.1 Detailed Description	338
4.173bt_gatt_client_evt_value_notification_t Struct Reference	338
4.173.1 Detailed Description	338
4.174bt_gatt_client_evt_write_char_descriptor_completed_t Struct Reference	338
4.174.1 Detailed Description	338
4.175bt_gatt_client_evt_write_char_long_descriptor_completed_t Struct Reference	339
4.175.1 Detailed Description	339
4.176bt_gatt_client_evt_write_char_long_value_completed_t Struct Reference	339
4.176.1 Detailed Description	339
4.177bt_gatt_client_evt_write_char_value_completed_t Struct Reference	339
4.177.1 Detailed Description	340
4.178bt_gatt_client_inc_service_declaration_t Struct Reference	340
4.178.1 Detailed Description	340
4.179bt_gatt_client_listener_t Struct Reference	340
4.180bt_gatt_client_mgr_t Struct Reference	340
4.181 bt_gatt_client_profile_finder_state_t Struct Reference	340
4.182bt_gatt_client_profile_finder_t Struct Reference	340
4.183bt_gatt_client_profile_header_t Struct Reference	340
4.184bt_gatt_client_service_definition_t Struct Reference	340
4.184.1 Detailed Description	341
4.185bt_gatt_client_session_t Struct Reference	341
4.186bt_gatt_evt_client_config_changed_t Struct Reference	341
4.186.1 Detailed Description	341
4.187bt_gatt_evt_ext_properties_changed_t Struct Reference	341
4.187.1 Detailed Description	342
4.188bt_gatt_evt_header_t Struct Reference	342
4.188.1 Detailed Description	342
4.189bt_gatt_evt_server_config_changed_t Struct Reference	342
4.189.1 Detailed Description	342
4.190bt_gatt_evt_value_changed_t Struct Reference	343
4.190.1 Detailed Description	343
4.191bt_gatt_evt_value_read_t Struct Reference	343
4.191.1 Detailed Description	343
4.192bt_gatt_listener_t Struct Reference	343
4.193bt_hci_command_p Struct Reference	343
4.194bt_hci_data_p Struct Reference	344
4.195bt_hci_event_p Struct Reference	344
4.196bt_hci_inquiry_response_t Struct Reference	344

CONTENTS xxix

4.197bt_hf_subscriber_number_t Struct Reference
4.198bt_hfp_ag_evt_search_completed_t Struct Reference
4.199bt_hfp_audio_packet_t Struct Reference
4.199.1 Detailed Description
4.199.2 Field Documentation
4.199.2.1 data
4.200bt_hfp_call_t Struct Reference
4.200.1 Detailed Description
4.200.2 Field Documentation
4.200.2.1 mode
4.200.2.2 multiparty
4.200.2.3 status
4.201 bt_hfp_event_register_t Struct Reference
4.201.1 Detailed Description
4.202bt_hfp_evt_activate_voice_recognition_t Struct Reference
4.203bt_hfp_evt_answer_call_t Struct Reference
4.204bt_hfp_evt_audio_connection_state_changed_t Struct Reference
4.204.1 Detailed Description
4.205bt_hfp_evt_audio_data_t Struct Reference
4.205.1 Detailed Description
4.206bt_hfp_evt_audio_packet_sent_t Struct Reference
4.207bt_hfp_evt_call_waiting_t Struct Reference
4.207.1 Detailed Description
4.208bt_hfp_evt_chld_t Struct Reference
4.209bt_hfp_evt_clip_received_t Struct Reference
4.209.1 Detailed Description
4.210bt_hfp_evt_command_completed_t Struct Reference
4.210.1 Detailed Description
4.211bt_hfp_evt_dial_request_received_t Struct Reference
4.211.1 Detailed Description
4.212bt_hfp_evt_inband_ring_changed_t Struct Reference
4.212.1 Detailed Description
4.213bt_hfp_evt_indicator_received_t Struct Reference
4.213.1 Detailed Description
4.214bt_hfp_evt_mic_volume_changed_t Struct Reference
4.214.1 Detailed Description
4.215bt_hfp_evt_query_operator_completed_t Struct Reference
4.215.1 Detailed Description
4.216bt_hfp_evt_send_dtmf_code_t Struct Reference
4.217bt_hfp_evt_slc_connection_state_changed_t Struct Reference

CONTENTS

4.217.1 Detailed Description
4.218bt_hfp_evt_spk_volume_changed_t Struct Reference
4.218.1 Detailed Description
4.219bt_hfp_evt_terminate_call_t Struct Reference
4.220bt_hfp_evt_voice_recognition_changed_t Struct Reference
4.220.1 Detailed Description
4.221bt_hfp_hf_t Struct Reference
4.222bt_hfp_ind Struct Reference
4.222.1 Detailed Description
4.223bt_hfp_session Struct Reference
4.224bt_hid_pdu_t Struct Reference
4.225bt_hid_report_t Struct Reference
4.226bt_hid_session_t Struct Reference
4.227bt_media_packet_t Struct Reference
4.227.1 Detailed Description
4.227.2 Field Documentation
4.227.2.1 data
4.227.2.2 marker
4.227.2.3 payload_type
4.227.2.4 ssrc
4.228bt_sdp_client_evt_connected_t Struct Reference
4.229bt_sdp_client_evt_disconnected_t Struct Reference
4.230bt_sdp_data_element_t Struct Reference
4.231bt_sdp_found_attr_list_t Struct Reference
4.232bt_sdp_sequence_t Struct Reference
4.233bt_sdp_serialization_state_p Struct Reference
4.234bt_sdp_service_transaction_p Struct Reference
4.235bt_sdp_transaction_t Struct Reference
4.236bt_security_mgr_t Struct Reference
4.237bt_sm_event_t Union Reference
4.238bt_sm_evt_connected_t Struct Reference
4.239bt_sm_evt_disconnected_t Struct Reference
4.240bt_sm_evt_generate_lkt_t Struct Reference
4.241bt_sm_evt_ltk_generated_t Struct Reference
4.242bt_sm_evt_master_csrk_notif_t Struct Reference
4.243bt_sm_evt_master_irk_notif_t Struct Reference
4.244bt_sm_evt_master_ltk_notif_t Struct Reference
4.245bt_sm_evt_master_ltk_request_t Struct Reference
4.246bt_sm_evt_oob_data_request_t Struct Reference
4.247bt_sm_evt_pairing_complete_t Struct Reference

CONTENTS xxxi

4.248bt_sm_evt_pairing_request_t Struct Reference
4.249bt_sm_evt_passkey_notification_t Struct Reference
4.250bt_sm_evt_passkey_request_t Struct Reference
4.251bt_sm_evt_peer_keys_notif_t Struct Reference
4.252bt_sm_evt_slave_ltk_request_t Struct Reference
4.253bt_sm_listener_t Struct Reference
4.254bt_sm_ltk_t Struct Reference
4.255bt_sm_pairing_features_t Struct Reference
4.256bt_sm_session_t Struct Reference
4.257bt_spp_port_t Struct Reference
4.257.1 Detailed Description
4.257.2 Field Documentation
4.257.2.1 state
4.258btx_csr_autobaud_buffer_t Struct Reference
4.259btx_csr_bccmd_header_t Struct Reference
4.260btx_csr_bccmd_listener_t Struct Reference
4.261btx_csr_cached_temperature_t Struct Reference
4.262btx_csr_create_operator_c_t Struct Reference
4.263btx_csr_exec_hq_script_buffer_t Struct Reference
4.264btx_csr_exec_script_buffer_t Struct Reference
4.265btx_csr_pio_direction_mask_t Struct Reference
4.266btx_csr_pio_protection_mask_t Struct Reference
4.267btx_csr_pio_t Struct Reference
4.268btx_csr_rssi_acl_t Struct Reference
4.269btx_csr_script_t Struct Reference
4.270btx_csr_set_ps_vars_buffer_t Struct Reference
4.271 btx_csr_strm_connect_t Struct Reference
4.272btx_csr_strm_get_sink_t Struct Reference
4.273btx_csr_strm_get_source_t Struct Reference
4.274btx csr var t Union Reference

# Chapter 1

## **Module Index**

#### 1.1 Modules

Horo	ie	2	liet	Ωf	ااد	modules
пеге	15	а	IISt	OI	all	modules

,	13
	16 29
SPP Configuration	
<u> </u>	40 41
5	44
	50
	52
	56
	77
Configuration	09
Advanced Audio Distribution Profile (A2DP)	12
Audio/Video Control Transport Protocol (AVCTP)	
Configuration	47
Audio/Video Remote Control Profile (AVRCP)	49
Configuration	85
ATT	87
ATT Server	89
Configuration	97
ATT Client	99
Configuration	11
GATT	12
GATT Server	13
GATT Client	22
Configuration	37
Security Manager	
OEM - HCI Communication Interface	
HCI UART (H4) transport protocol	28
OEM - Timer Interface	
OEM - Non-volatile Storage Interface	
OEM - Logging Interface	
Vendor specific extensions to HCI	
CSB	12

2 **Module Index** 

# Chapter 2

### **Data Structure Index**

#### 2.1 Data Structures

Here	are	the	data	structures	with	brief	descri	otions:

bt_spp_port_t::_bt_spp_port_flags_t
bt_a2dp_aac_config_t
bt_a2dp_event_t Parameter to an application callback
• •
bt_a2dp_evt_open_and_start_stream_completed_t Parameter to A2DP_EVT_OPEN_AND_START_STREAM_COMPLETED event
bt_a2dp_mgr_t A2DP manager
bt_a2dp_mpeg_config_t
bt_a2dp_sbc_config_t
bt_a2dp_sbc_packet_info_t
bt_att_attribute_t
bt_att_client_evt_conn_param_update_completed_t
Parameter to ATT_CLIENT_EVT_CONN_PARAM_UPDATE_COMPLETED event
bt_att_client_evt_conn_param_update_t
Parameter to ATT_CLIENT_EVT_CONN_PARAM_UPDATE_REQUEST event 257
bt_att_client_evt_execute_write_response_t
Parameter to ATT_CLIENT_EVT_EXECUTE_WRITE_RESPONSE event
bt_att_client_evt_find_by_type_value_response_t
Parameter to ATT_CLIENT_EVT_FIND_BY_TYPE_VALUE_RESPONSE event
bt_att_client_evt_info_response_t
Parameter to ATT_CLIENT_EVT_FIND_INFO_RESPONSE event
bt_att_client_evt_mtu_response_t
Parameter to ATT CLIENT EVT EXCHANGE MTU RESPONSE event 258
bt_att_client_evt_prepare_write_response_t
Parameter to ATT_CLIENT_EVT_PREPARE_WRITE_RESPONSE event
bt_att_client_evt_read_blob_response_t
Parameter to ATT_CLIENT_EVT_READ_BLOB_RESPONSE event
bt_att_client_evt_read_by_group_type_response_t
Parameter to ATT_CLIENT_EVT_READ_BY_GROUP_TYPE_RESPONSE event
bt_att_client_evt_read_by_type_response_t
Parameter to ATT_CLIENT_EVT_READ_BY_TYPE_RESPONSE event
bt_att_client_evt_read_multiple_response_t
Parameter to ATT_CLIENT_EVT_READ_MULTIPLE_RESPONSE event
bt_att_client_evt_read_response_t
Parameter to ATT_CLIENT_EVT_READ_RESPONSE event
bt_att_client_evt_t
Parameter to ATT client application callback

4 Data Structure Index

bt_att_client_evt_value_indication_t	
Parameter to ATT_CLIENT_EVT_VALUE_INDICATION event	262
bt_att_client_evt_value_notification_t	
Parameter to ATT_CLIENT_EVT_VALUE_NOTIFICATION event	263
bt_att_client_evt_write_response_t	
Parameter to ATT_CLIENT_EVT_WRITE_RESPONSE event	
bt_att_client_mgr_t	
bt_att_client_session_t	264
bt_att_client_uuid_t	
UUID	264
bt_att_enum_context_t	264
bt_att_evt_attr_indication_sent_t	
Parameter to ATT_SERVER_EVT_ATTR_VALUE_INDICATED event	264
bt_att_evt_attr_notification_sent_t	
Parameter to ATT_SERVER_EVT_ATTR_VALUE_NOTIFIED event	265
bt_att_evt_attr_value_changed_t	
Parameter to ATT_SERVER_EVT_ATTR_VALUE_CHANGED_BY_SERVER and ATT_SER	
VER_EVT_ATTR_VALUE_CHANGED_BY_CLIENT events	265
bt_att_evt_attr_value_read_t	
Parameter to ATT_SERVER_EVT_ATTR_VALUE_READ event	266
bt_att_evt_authorization_requested_t	
Parameter to ATT_SERVER_EVT_AUTHORIZATION_REQUESTED event	
bt_att_evt_data_received_t	267
bt_att_evt_session_connected_t	
Parameter to ATT_SERVER_EVT_SESSION_CONNECTED event	267
bt_att_evt_session_disconnected_t	
Parameter to ATT_SERVER_EVT_SESSION_DISCONNECTED event	267
bt_att_evt_tran_timeout_t	
Parameter to ATT_SERVER_EVT_TRAN_TIMEOUT event	267
bt_att_find_by_type_value_response_t	000
Structure to store response to a "find by type value" request	268
bt_att_find_info_response_t	000
Structure to store response to a "find info" request	
bt_att_found_attribyte_t	
bt_att_listener_t	269
bt_att_mgr_t	000
ATT manager	269
bt_att_read_by_group_type_response_t	000
Structure to store response to a "read by group type" request	269
bt_att_read_by_type_response_t	070
Structure to store response to a "read by type" request	
bt_att_session_evt_authorization_requested_t	
bt_att_session_evt_packet_sent_t	
bt_att_session_t	2/0
bt_av_add_to_now_playing_t	070
Parameter to AVRCP_EVT_ADD_TO_NOW_PLAYING_COMPLETED event	270
bt_av_battery_status_of_ct_t	074
Parameter to AVRCP_EVT_BATTERY_STATUS_OF_CT_RECEIVED event	2/1
bt_av_capability_company_id_t	074
Parameter to AVRCP_EVT_COMPANY_ID_LIST_RECEIVED event	2/1
bt_av_capability_event_id_t	070
Parameter to AVRCP_EVT_EVENT_ID_LIST_RECEIVED event	
bt_av_command_t	272
bt_av_displayable_character_set_t	070
Parameter to AVRCP_EVT_DISPLAYABLE_CHARACTER_SET_RECEIVED event	212
bt_av_element_attribute_t	070
Media element attribute	2/3

2.1 Data Structures 5

bt_av_element_attributes_t	
Parameter to AVRCP_EVT_GET_ELEMENT_ATTRIBUTES_RECEIVED event	273
bt_av_element_id_t	
Media element UID	274
bt_av_get_element_attributes_t	
<del>,</del>	274
bt_av_notification_addressed_player_changed_t	
	275
	275
bt_av_notification_battery_status_t	
	275
bt_av_notification_playback_pos_changed_t	
	276
bt_av_notification_playback_status_changed_t	270
	276
bt_av_notification_system_status_changed_t	2/0
	277
	277
bt_av_notification_t	077
· · · · · · · · · · · · · · · · · · ·	277
bt_av_notification_track_changed_t	
	279
bt_av_notification_uids_changed_t	
	279
bt_av_notification_volume_changed_t	
	280
bt_av_play_item_t	
Parameter to AVRCP_EVT_PLAY_ITEM_COMPLETED event	280
bt_av_play_status_t	
Parameter to AVRCP_EVT_GET_PLAY_STATUS_RECEIVED event	280
bt_av_player_setting_current_values_t	
Parameter to AVRCP_EVT_PLAYER_CURRENT_SETTING_VALUES_RECEIVED event	281
bt_av_player_setting_values_t	
Parameter to AVRCP_EVT_PLAYER_SETTING_VALUES_RECEIVED event	282
bt_av_player_setting_values_text_t	
Parameter to AVRCP_EVT_PLAYER_SETTING_VALUES_TEXT_RECEIVED event	282
bt_av_player_settings_t	
	282
bt_av_player_settings_text_t	
$ \cdot$ $\cdot$ $ \cdot$ $\cdot$ $ \cdot$	283
	283
bt_av_register_notification_t	200
	283
bt_av_response_t	
,	284
bt_av_set_absolute_volume_t	204
	285
bt av set addressed player t	200
	200
	286
bt_avctp_channel_t	000
<b>'</b>	286
bt_avctp_event_t	000
· ·	286
bt_avctp_evt_channel_connected_t	
	287
bt_avctp_evt_channel_disconnected_t	
	287
bt_avctp_evt_command_cancelled_t	
Parameter to AVCTP_EVT_COMMAND_CANCELLED event	288

6 Data Structure Index

hŧ	aveto	out command received t	
		_evt_command_received_t Parameter to AVCTP_EVT_COMMAND_RECEIVED event	288
bt_	_avctp_	_evt_command_sent_t Parameter to AVCTP_EVT_COMMAND_SENT event	288
bt	avctp	evt_connection_failed_t	
		Parameter to AVCTP_EVT_CONNECTION_FAILED event	289
		_evt_response_cancelled_t Parameter to AVCTP_EVT_RESPONSE_CANCELLED event	289
bt_	_avctp_	_evt_response_received_t Parameter to AVCTP_EVT_RESPONSE_RECEIVED event	290
bt_	_avctp_	_evt_response_sent_t Parameter to AVCTP_EVT_RESPONSE_SENT event	290
bt_		message_t AVCTP message description	290
L.		-	230
טנ_	_avctp_	AVCTP manager	291
bt_	_avctp_	_transport_t	
		AVCTP transport description	291
bt	avdtp	_codec_op_decode_t	292
		codec op encode t	292
		codec op param t	202
		Parameter to a codec handler	292
bt_	_avdtp_	_codec_op_parse_config_t	
		Parameter to AVDTP_CODEC_OPCODE_PARSE_CONFIG operation	292
bt	avdtp	_codec_op_parse_packet_t	293
		codec_op_serialize_config_t	
טנ_	_avuip_	= · = · = · = · <del>*</del> =	202
		Parameter to AVDTP_CODEC_OPCODE_SERIALIZE_CONFIG operation	293
טנ_	_avdtp_	_codec_t	004
		Codec handler description	294
bt_	_avdtp_	_event_t Parameter to an application callback	295
bt	avdtp	evt_abort_stream_requested_t	
		Parameter to AVDTP_EVT_ABORT_STREAM_REQUESTED event	296
bt	avdto	evt close stream completed t	
~-	_	Parameter to AVDTP_EVT_CLOSE_STREAM_COMPLETED event	297
bt	avdtp	_evt_close_stream_requested_t	
		Parameter to AVDTP_EVT_CLOSE_STREAM_REQUESTED event	297
pt_	_avdtp_	_evt_ctrl_channel_connected_t	
ht	avdtn	Parameter to AVDTP_EVT_CTRL_CHANNEL_CONNECTED event evt ctrl channel disconnected t	298
		Parameter to AVDTP_EVT_CTRL_CHANNEL_DISCONNECTED event	298
bt_	_avdtp_	_evt_delay_report_completed_t Parameter to AVDTP_EVT_DELAYREPORT_COMPLETED event	299
bt_	_avdtp_	_evt_discover_completed_t Parameter to AVDTP_EVT_DISCOVER_COMPLETED event	299
hŧ	avdta	evt_get_sep_capabilities_completed_t	299
		Parameter to AVDTP_EVT_GET_SEP_CAPABILITIES_COMPLETED event	300
bt_	_avdtp_	_evt_get_stream_configuration_completed_t	
ht	avdto	Parameter to AVDTP_EVT_SET_STREAM_CONFIGURATION_COMPLETED event evt_media_packet_received_t	300
		Parameter to AVDTP_EVT_MEDIA_PACKET_RECEIVED event	301
bt_	_avdtp_	_evt_media_packet_send_failed_t Parameter to AVDTP_EVT_MEDIA_PACKET_SEND_FAILED event	301
bt_	_avdtp_	_evt_media_packet_sent_t	
		Parameter to AVDTP_EVT_MEDIA_PACKET_SENT event	301
bt_	_avdtp_	_evt_open_stream_completed_t Parameter to AVDTP_EVT_OPEN_STREAM_COMPLETED event	302

2.1 Data Structures 7

bt_avdtp	_evt_open_stream_requested_t	
	Parameter to AVDTP_EVT_OPEN_STREAM_REQUESTED event	302
bt_avdtp	_evt_reconfigure_stream_requested_t Parameter to AVDTP_EVT_RECONFIGURE_STREAM_REQUESTED event	303
bt_avdtp	_evt_sep_capabilities_received_t	
	Parameter to AVDTP_EVT_SEP_CAPABILITIES_RECEIVED and AVDTP_EVT_STREAM_C $\leftarrow$ ONFIGURATION_RECEIVED events	304
bt_avdtp	_evt_sep_info_received_t	
ht audto	Parameter to AVDTP_EVT_SEP_INFO_RECEIVED event	304
DI_avuip	Parameter to AVDTP_EVT_SET_STREAM_CONFIGURATION_COMPLETED event	305
bt avdtp	evt set stream configuration requested t	000
	Parameter to AVDTP_EVT_SET_STREAM_CONFIGURATION_REQUESTED event	306
bt_avdtp	_evt_start_stream_completed_t	
	Parameter to AVDTP_EVT_START_STREAM_COMPLETED event	306
bt_avdtp	_evt_start_stream_requested_t	007
bt sudto	Parameter to AVDTP_EVT_START_STREAM_REQUESTED event	307
DI_avuip	_evt_stream_aborted_t Parameter to AVDTP_EVT_STREAM_ABORTED event	307
bt avdtp	_evt_stream_closed_t	007
_ '	Parameter to AVDTP_EVT_STREAM_CLOSED event	308
bt_avdtp	_evt_stream_configured_t	
	Parameter to AVDTP_EVT_STREAM_CONFIGURED event	308
bt_avdtp	_evt_stream_opened_t	000
ht avdtn	Parameter to AVDTP_EVT_STREAM_OPENED event	309
Di_avuip	Parameter to AVDTP_EVT_STREAM_RECONFIGURE_COMPLETED event	309
bt avdtp	_evt_stream_reconfigured_t	
	Parameter to AVDTP_EVT_STREAM_RECONFIGURED event	310
bt_avdtp	_evt_stream_security_control_completed_t	
	Parameter to AVDTP_EVT_STREAM_SECURITY_CONTROL_COMPLETED event	310
ot_avotp	_evt_stream_started_t Parameter to AVDTP_EVT_STREAM_STARTED event	310
bt avdtp	_evt_stream_suspended_t	310
	Parameter to AVDTP_EVT_STREAM_SUSPENDED event	311
bt_avdtp	_evt_suspend_stream_completed_t	
	Parameter to AVDTP_EVT_SUSPEND_STREAM_COMPLETED event	311
bt_avdtp	_evt_suspend_stream_requested_t	040
bt_avdtp	Parameter to AVDTP_EVT_SUSPEND_STREAM_REQUESTED event	312
Di_avuip	AVDTP manager	312
bt avdtp	sep_capabilities_t	0.2
	SEP capabilities	314
bt_avdtp	•	
	SEP description	317
ot_avotp	_stream_t Stream description	317
ht avdtn	transport_channel_t	317
or_avarp	Transport channel description	319
bt_avdtp	_transport_session_t	
	Transport session description	320
bt_avrcp	_channel_t	
bt some	AVRCP channel description	
bt_avrcp	_device_t	321
υι_αντορ	Parameter to an application callback	321
bt_avrcp	_evt_channel_connected_t	
	Parameter to AVRCP_EVT_CONTROL_CHANNEL_CONNECTED event	323

8 Data Structure Index

bt_avrcp_evt_channel_disconnected_t	
Parameter to AVRCP_EVT_CONTROL_CHANNEL_DISCONNECTED event	323
bt_avrcp_evt_connection_failed_t	
Parameter to AVRCP_EVT_CONTROL_CONNECTION_FAILED event	323
bt_avrcp_evt_panel_command_received_t	
Parameter to AVRCP_EVT_PANEL_COMMAND_RECEIVED event	324
bt_avrcp_evt_panel_response_received_t	
Parameter to AVRCP_EVT_PANEL_RESPONSE_RECEIVED event	324
$bt\_avrcp\_evt\_register\_events\_completed\_t  . \ . \ . \ . \ . \ . \ . \ . \ . \ .$	325
bt_avrcp_evt_search_completed_t	
Parameter to AVRCP_EVT_SEARCH_COMPLETED event	325
bt_avrcp_mgr_t	
AVRCP manager	326
$bt\_cp\_header\_t \ \dots $	328
bt_gatt_client_char_declaration_t	
Characteristic Declaration	328
bt_gatt_client_char_descriptor_t	
Characteristic Descriptor Declaration	328
bt_gatt_client_char_value_t	
Characteristic Value	329
bt_gatt_client_evt_conn_param_update_completed_t	
Parameter to GATT_CLIENT_EVT_CONN_PARAM_UPDATE_COMPLETED event	329
bt_gatt_client_evt_conn_param_update_t	
Parameter to GATT_CLIENT_EVT_CONN_PARAM_UPDATE_REQUEST event	329
bt_gatt_client_evt_discover_all_chars_completed_t	
Parameter to GATT_CLIENT_EVT_DISCOVER_ALL_CHARS_COMPLETED event	330
bt_gatt_client_evt_discover_all_services_completed_t	
Parameter to GATT_CLIENT_EVT_DISCOVER_ALL_SERVICES_COMPLETED event	330
bt_gatt_client_evt_discover_char_by_uuid_completed_t	
Parameter to GATT_CLIENT_EVT_DISCOVER_CHAR_BY_UUID_COMPLETED event	331
bt_gatt_client_evt_discover_descriptors_completed_t	
Parameter to GATT_CLIENT_EVT_DISCOVER_CHAR_DESCRIPTORS_COMPLETED event	332
bt_gatt_client_evt_discover_service_by_uuid_completed_t	
Parameter to GATT_CLIENT_EVT_DISCOVER_SERVICE_BY_UUID_COMPLETED event	332
bt_gatt_client_evt_exchange_mtu_completed_t	
Parameter to GATT_CLIENT_EVT_EXCHANGE_MTU_COMPLETED event	333
bt_gatt_client_evt_find_included_completed_t	
Parameter to GATT_CLIENT_EVT_FIND_INCLUDED_SERVICES_COMPLETED event	
bt_gatt_client_evt_profile_found_t	334
bt_gatt_client_evt_read_by_char_uuid_completed_t	
Parameter to GATT_CLIENT_EVT_READ_USING_CHAR_UUID_COMPLETED event	334
bt_gatt_client_evt_read_char_descriptor_completed_t	
Parameter to GATT_CLIENT_EVT_READ_CHAR_DESCRIPTOR_COMPLETED event	334
bt_gatt_client_evt_read_char_long_descriptor_completed_t	
Parameter to GATT_CLIENT_EVT_READ_CHAR_LONG_DESCRIPTOR_COMPLETED event	335
bt_gatt_client_evt_read_char_long_value_completed_t	
Parameter to GATT_CLIENT_EVT_READ_CHAR_LONG_VALUE_COMPLETED event	335
bt_gatt_client_evt_read_char_value_completed_t	
Parameter to GATT_CLIENT_EVT_READ_CHAR_VALUE_COMPLETED event	336
bt_gatt_client_evt_read_multiple_char_values_completed_t	
Parameter to GATT_CLIENT_EVT_READ_MULTIPLE_CHAR_VALUES_COMPLETED event	336
bt_gatt_client_evt_t	٠.
Parameter to GATT client application callback	337
bt_gatt_client_evt_value_notification_t	000
Parameter to GATT_CLIENT_EVT_VALUE_NOTIFICATION event	338
bt_gatt_client_evt_write_char_descriptor_completed_t	000
Parameter to GATT_CLIENT_EVT_WRITE_CHAR_DESCRIPTOR_COMPLETED event	338

2.1 Data Structures 9

bt_gatt_client_evt_write_char_long_descriptor_completed_t	
Parameter to GATT_CLIENT_EVT_WRITE_CHAR_LONG_DESCRIPTOR_COMPLETED event	339
bt_gatt_client_evt_write_char_long_value_completed_t	
Parameter to GATT_CLIENT_EVT_WRITE_CHAR_LONG_VALUE_COMPLETED event	339
bt_gatt_client_evt_write_char_value_completed_t	
	339
bt_gatt_client_inc_service_declaration_t	
- <del>-</del>	340
<del>_</del>	
bt_gatt_client_profile_finder_state_t	340
_ <del></del>	
bt_gatt_client_profile_finder_t	340
	340
bt_gatt_client_service_definition_t	
<del>_</del>	341
bt_gatt_evt_client_config_changed_t	
Parameter to GATT_SERVER_EVT_CLIENT_CONFIG_CHANGED event	341
bt_gatt_evt_ext_properties_changed_t	
Parameter to GATT_SERVER_EVT_EXTENDED_PROPERTIES_CHANGED event	341
bt_gatt_evt_header_t	
Common to all event parameters header	342
bt gatt evt server config changed t	
	342
bt_gatt_evt_value_changed_t	
Parameter to GATT_SERVER_EVT_VALUE_CHANGED event	343
bt_gatt_evt_value_read_t	0.0
Parameter to GATT_SERVER_EVT_VALUE_READ event	343
bt_hfp_ag_evt_search_completed_t	344
bt_hfp_audio_packet_t	
Parameter to HFP_EVENT_AUDIO_DATA_RECEIVED event	344
bt_hfp_call_t	
Stores information about a call	345
bt_hfp_event_register_t	
Stores value of HF registrations	346
bt_hfp_evt_activate_voice_recognition_t	346
	346
bt_hfp_evt_audio_connection_state_changed_t	
HFP_EVENT_AUDIO_CONNECTION_STATE_CHANGED event parameter	346
bt hfp evt audio data t	0.10
	347
	347
bt_hfp_evt_call_waiting_t	0.47
HFP_EVENT_CALL_WAITING event parameter	
bt_hfp_evt_chld_t	348
bt_hfp_evt_clip_received_t	_
HFP_EVENT_CLIP_RECEIVED event parameter	348
bt_hfp_evt_command_completed_t	
HFP_EVENT_CMD_COMPLETED event parameter	348
bt_hfp_evt_dial_request_received_t	
HFP_AG_EVENT_DIAL_REQUEST_RECEIVED event parameter	348

10 Data Structure Index

bt_hfp_evt_inband_ring_changed_t	
HFP_EVENT_INBAND_RING_CHANGED event parameter	349
bt_hfp_evt_indicator_received_t	
HFP_EVENT_INDICATOR_RECEIVED event parameter	349
bt_hfp_evt_mic_volume_changed_t	
HFP_EVENT_MIC_VOLUME_CHANGED event parameter	350
bt_hfp_evt_query_operator_completed_t	
HFP_EVENT_QUERY_OPERATOR_COMPLETED event parameter	350
bt_hfp_evt_send_dtmf_code_t	351
bt_hfp_evt_slc_connection_state_changed_t	
HFP_EVENT_SLC_CONNECTION_STATE_CHANGED event parameter	351
bt_hfp_evt_spk_volume_changed_t	
HFP_EVENT_SPK_VOLUME_CHANGED event parameter	351
bt_hfp_evt_terminate_call_t	351
bt_hfp_evt_voice_recognition_changed_t	
HFP_EVENT_VOICE_RECOGNITION_CHANGED event parameter	351
bt_hfp_hf_t	352
bt_hfp_ind	
	352
bt_hfp_session	352
bt_hid_pdu_t	352
bt_hid_report_t	352
bt_hid_session_t	352
bt_media_packet_t	
	352
bt_sdp_client_evt_connected_t	
_ '- '	354
bt_sdp_data_element_t	354
bt_sdp_found_attr_list_t	354
	354
	354
bt_sdp_service_transaction_p	354
bt_sdp_transaction_t	354
bt_security_mgr_t	354
bt_sm_event_t	354
bt_sm_evt_connected_t	354
bt_sm_evt_disconnected_t	354
bt_sm_evt_generate_lkt_t	354
bt_sm_evt_ltk_generated_t	354
bt_sm_evt_master_csrk_notif_t	354
bt_sm_evt_master_irk_notif_t	354
bt_sm_evt_master_ltk_notif_t	354
bt_sm_evt_master_ltk_request_t	355
bt_sm_evt_oob_data_request_t	355
bt_sm_evt_pairing_complete_t	355
bt_sm_evt_pairing_request_t	355
bt_sm_evt_passkey_notification_t	355
bt_sm_evt_passkey_request_t	355
bt_sm_evt_peer_keys_notif_t	355
bt_sm_evt_slave_ltk_request_t	355
bt_sm_listener_t	355
bt_sm_ltk_t	355
bt_sm_pairing_features_t	355
bt_sm_session_t	355
bt_spp_port_t	
Serial port structure	355
btx_csr_autobaud_buffer_t	356
btx_csr_bccmd_header_t	356

2.1 Data Structures

btx_csr_bccmd_listener_t
btx_csr_cached_temperature_t
btx_csr_create_operator_c_t
btx_csr_exec_hq_script_buffer_t 356
btx_csr_exec_script_buffer_t
btx_csr_pio_direction_mask_t
btx_csr_pio_protection_mask_t
btx_csr_pio_t
btx_csr_rssi_acl_t
btx_csr_script_t
btx_csr_set_ps_vars_buffer_t
btx_csr_strm_connect_t
btx_csr_strm_get_sink_t
btx_csr_strm_get_source_t
btx_csr_var_t

12 Data Structure Index

# **Chapter 3**

# **Module Documentation**

# 3.1 System Functions

Functions in this module provide interface to DotStack functionality that is common to all protocols and profiles.

# **Typedefs**

typedef void(\* bt\_sys\_callback\_fp) (bt\_bool success, void \*param)
 System start callback.

## **Functions**

void bt\_sys\_init (void)

Initialize the Bluetooth system.

void bt\_sys\_init\_ex (bt\_byte default\_link\_policy)

Initialize the Bluetooth system.

void bt\_sys\_start (bt\_bool discoverable, bt\_bool connectable, const bt\_byte \*sdp\_db, bt\_uint sdp\_db\_len, bt\_sys\_callback\_fp callback, void \*callback\_param)

Start the Bluetooth system.

• bt\_l2cap\_mgr\_t \* bt\_sys\_get\_l2cap\_manager (void)

Get the L2CAP manager.

const bt\_byte \* bt\_sys\_get\_version (void)

Get the version of the dotstack library.

## 3.1.1 Detailed Description

Functions in this module provide interface to DotStack functionality that is common to all protocols and profiles.

# 3.1.2 Typedef Documentation

3.1.2.1 typedef void(\* bt\_sys\_callback\_fp) (bt\_bool success, void \*param)

System start callback.

This callback function is called when system start initiated by bt\_sys\_start() has completed.

#### **Parameters**

success	Success of the operation: BT_TRUE if successfull, BT_FALSE otherwise.
param	Callback parameter that was specified when bt_sys_start() was called.

#### 3.1.3 Function Documentation

3.1.3.1 bt\_l2cap\_mgr\_t\* bt\_sys\_get\_l2cap\_manager ( void )

Get the L2CAP manager.

This function returns the L2CAP manager. The L2CAP manager is created as part of the start up sequence.

#### Returns

The L2CAP manager.

3.1.3.2 const bt\_byte\* bt\_sys\_get\_version ( void )

Get the version of the dotstack library.

# Returns

The version of the dotstack library.

3.1.3.3 void bt\_sys\_init ( void )

Initialize the Bluetooth system.

This function initializes all internal variables of HCI, L2CAP and SDP modules. It must be called by the application before it can access any functionality provided by the library. In addition to this initialization function the application must call initialization functions of all other profile modules the application is intended to use. E.g., if the application is using the SPP module the bt\_spp\_init() must be called right after calling bt\_sys\_init().

This function essentially calls bt\_sys\_init\_ex(HCI\_LINK\_POLICY\_ENABLE\_ALL) so all link policy setting are enabled.

3.1.3.4 void bt\_sys\_init\_ex ( bt\_byte default\_link\_policy )

Initialize the Bluetooth system.

This function initializes all internal variables of HCI, L2CAP and SDP modules. It must be called by the application before it can access any functionality provided by the library. In addition to this initialization function the application must call initialization functions of all other profile modules the application is intended to use. E.g., if the application is using the SPP module the bt\_spp\_init() must be called right after calling bt\_sys\_init().

Also, the caller must provide an SDP database.

3.1 System Functions 15

#### **Parameters**

default_link_← policy	default link policy settings. This is a bitmask that defines the initial value of the link policy settings for all new BR/EDR connections. This value can be a combination of the following values:
	HCI_LINK_POLICY_ENABLE_ROLE_SWITCH
	HCI_LINK_POLICY_ENABLE_HOLD_MODE
	HCI_LINK_POLICY_ENABLE_SNIFF_MODE
	HCI_LINK_POLICY_ENABLE_PARK_STATE
	To enable all settings pass HCI_LINK_POLICY_ENABLE_ALL.

3.1.3.5 void bt\_sys\_start ( bt\_bool *discoverable*, bt\_bool *connectable*, const bt\_byte \* *sdp\_db*, bt\_uint *sdp\_db\_len*, bt\_sys\_callback\_fp *callback*, void \* *callback\_param* )

Start the Bluetooth system.

After all modules used by the application have been initialized this function should be called to start the Bluetooth system operation. During the start up sequence it will reset and initialize the HCl controller and then create the L2CAP manager. The application will be notified when the start up sequence completes by calling the provided callback function.

Also, the caller must provide an SDP database.

#### **Parameters**

discoverable	defines whether the device is discoverable after reset.
connectable	defines whether the device is connectable after reset.
sdp_db	SDP database data.
sdp_db_len	Length of SDP database data.
callback	A callback function that will be called when the start up sequence is complete.
callback_param	An arbitrary pointer that will be passed to the callback function.

## 3.2 HCI

The Host Controller Interface (HCI) provides a uniform interface method of accessing a Bluetooth Controller's capabilities.

#### **Macros**

- #define bt\_hci\_add\_param\_uint(cmd, value) bt\_hci\_add\_param\_int(cmd, (bt\_int)(value))
  - Add unsigned int parameter to an HCI command.
- #define bt hci add param ulong(cmd, value) bt hci add param long(cmd, (bt long)(value))
  - Add unsigned long parameter to an HCI command.
- #define bt hci add param hconn(pcmd, value) bt hci add param int(pcmd, value)
  - Add connection handle parameter to an HCI command.
- #define bt hci get param hconn(pcmd, pvalue, poffset) bt hci get param int(pcmd, pvalue, poffset)
  - Get connection handle parameter from HCI command.
- #define bt\_hci\_get\_evt\_param\_hconn(pevt, pvalue, poffset) bt\_hci\_get\_evt\_param\_int(pevt, pvalue, poffset)

  Get connection handle parameter from HCI event.

# **Typedefs**

- typedef void(\* bt\_hci\_start\_callback\_fp) (bt\_bool success, void \*param)
  - HCI initialization callback.
- typedef void(\* bt hci stop callback fp) (void \*param)
  - HCI stop callback.
- typedef void(\* bt\_hci\_connect\_callback\_fp) (bt\_byte status, bt\_hci\_conn\_state\_t \*pconn, void \*param)
  - HCI connect callback.
- typedef void(\* bt\_hci\_disconnect\_callback\_fp) (bt\_byte status, bt\_byte reason, bt\_hci\_conn\_state\_t \*pconn, void \*param)
  - HCI disconnect callback.

#### **Functions**

- bt\_hci\_command\_p bt\_hci\_alloc\_command (bt\_int opcode, bt\_hci\_cmd\_callback\_fp callback)
  - Allocate and initialize an HCI command structure.
- void bt\_hci\_free\_command (bt\_hci\_command\_p cmd)
  - Free HCI command.
- bt\_hci\_command\_p bt\_hci\_alloc\_canned\_command (const bt\_byte \*canned\_command, bt\_hci\_cmd\_ ← callback\_fp callback)
  - Allocate and initialize an HCl command structure for a canned (pre-formatted) command.
- bt\_bool bt\_hci\_add\_param\_byte (bt\_hci\_command\_p pcmd, bt\_byte value)
  - Add byte parameter to an HCl command.
- bt\_bool bt\_hci\_add\_param\_int (bt\_hci\_command\_p pcmd, bt\_int value)
  - Add int parameter to an HCI command.
- bt\_bool bt\_hci\_add\_param\_long (bt\_hci\_command\_p pcmd, bt\_long value)
  - Add long parameter to an HCI command.
- bt\_bool bt\_hci\_add\_param\_bdaddr (bt\_hci\_command\_p pcmd, const bt\_bdaddr\_t \*pbdaddr)
  - Add BD address parameter to an HCI command.
- bt\_bool bt\_hci\_add\_param\_string (bt\_hci\_command\_p pcmd, const char \*ps, bt\_int len)
  - Add string parameter to an HCI command.
- bt\_bool bt\_hci\_add\_param\_cod (bt\_hci\_command\_p pcmd, bt\_long value)

3.2 HCI 17

Add class of device parameter to an HCl command.

• bt\_bool bt\_hci\_add\_param\_linkkey (bt\_hci\_command\_p pcmd, const bt\_linkkey\_t \*linkkey)

Add link key parameter to an HCl command.

• bt\_bool bt\_hci\_get\_param\_byte (bt\_hci\_command\_p pcmd, bt\_byte \*pvalue, bt\_int \*offset)

Get byte parameter from HCl command.

• bt\_bool bt\_hci\_get\_param\_int (bt\_hci\_command\_p pcmd, bt\_int \*pvalue, bt\_int \*poffset)

Get int parameter from HCl command.

• bt\_bool bt\_hci\_get\_param\_long (bt\_hci\_command\_p pcmd, bt\_long \*pvalue, bt\_int \*poffset)

Get long parameter from HCI command.

• bt\_bool bt\_hci\_get\_param\_bdaddr (bt\_hci\_command\_p pcm, bt\_bdaddr\_p pvalue, bt\_int\_p poffset)

Get BD address parameter from HCl command.

• bt\_bool bt\_hci\_get\_param\_linkkey (bt\_hci\_command\_p pcmd, bt\_byte\_p pvalue, bt\_int\_p poffset)

Get link key parameter from HCl command.

• bt\_bool bt\_hci\_get\_evt\_param\_byte (bt\_hci\_event\_p pevt, bt\_byte \*pvalue, bt\_int \*poffset)

Get byte parameter from HCl event.

• bt\_bool bt\_hci\_get\_evt\_param\_int (bt\_hci\_event\_p pevt, bt\_int \*pvalue, bt\_int \*poffset)

Get int parameter from HCl event.

• bt\_bool bt\_hci\_get\_evt\_param\_long (bt\_hci\_event\_p pevt, bt\_long \*pvalue, bt\_int \*poffset)

Get long parameter from HCl event.

• bt\_bool bt\_hci\_get\_evt\_param\_bdaddr (bt\_hci\_event\_p pevt, bt\_bdaddr\_p pvalue, bt\_int\_p poffset)

Get bd address parameter from HCl event.

• bt\_bool bt\_hci\_get\_evt\_param\_devclass (bt\_hci\_event\_p pevt, bt\_long\_p pvalue, bt\_int\_p poffset)

Get class of device parameter from HCl event.

• bt\_bool bt\_hci\_get\_evt\_param\_linkkey (bt\_hci\_event\_p pevt, bt\_linkkey\_p pvalue, bt\_int\_p poffset)

Get link key parameter from HCl event.

void bt\_hci\_init (void)

Initialize the HCI layer.

void bt\_hci\_init\_ex (bt\_byte default\_link\_policy)

Initialize the HCI layer.

bt\_bool bt\_hci\_start (bt\_hci\_start\_callback\_fp callback, void \*callback\_param, bt\_byte enable\_scan)
 Start HCl layer.

void bt\_hci\_start\_no\_init (void)

Start HCI layer without controller configuration.

• void bt\_hci\_stop (bt\_hci\_stop\_callback\_fp callback, void \*callback\_param)

Stop HCI layer.

• bt\_bool bt\_hci\_reset (bt\_hci\_cmd\_callback\_fp callback, void \*callback\_param)

Reset controller.

bt\_bool bt\_hci\_connect (bt\_bdaddr\_p dest, bt\_uint packet\_type, bt\_byte pg\_scan\_rpt\_mode, bt\_byte role\_
 — switch, bt\_uint acl\_config, bt\_hci\_connect\_callback\_fp callback, void \*param)

Connect to a remote device.

bt\_bool bt\_hci\_listen (bt\_hci\_connect\_callback\_fp cb, void \*param)

Listen for incoming connections.

• bt bool bt hci disconnect (bt hci conn state t \*pconn)

Abort connection.

• bt\_bool bt\_hci\_write\_local\_name (const char \*device\_name, bt\_hci\_cmd\_callback\_fp cb)

Write local device name.

#### Link control commands

The Link Control commands allow a Controller to control connections to other BR/EDR Controllers.

- #define HCI INQUIRY HCI OPCODE(OGF LINK CONTROL, 0x0001)
- #define HCI\_INQUIRY\_CANCEL HCI\_OPCODE(OGF\_LINK\_CONTROL, 0x0002)
- #define HCI\_PERIODIC\_INQUIRY\_MODE HCI\_OPCODE(OGF\_LINK\_CONTROL, 0x0003)
- #define HCI EXIT PERIODIC INQUIRY MODE HCI OPCODE(OGF LINK CONTROL, 0x0004)
- #define HCI\_CREATE\_CONNECTION HCI\_OPCODE(OGF\_LINK\_CONTROL, 0x0005)
- #define HCI DISCONNECT HCI OPCODE(OGF LINK CONTROL, 0x0006)
- #define HCI\_CREATE\_CONNECTION\_CANCEL HCI\_OPCODE(OGF\_LINK\_CONTROL, 0x0008)
- #define HCI\_ACCEPT\_CONNECTION\_REQUEST\_HCI\_OPCODE(OGF\_LINK\_CONTROL, 0x0009)
- #define HCI REJECT CONNECTION REQUEST HCI OPCODE(OGF LINK CONTROL, 0x000A)
- #define HCI\_LINK\_KEY\_REQUEST\_REPLY HCI\_OPCODE(OGF\_LINK\_CONTROL, 0x000B)
- #define HCI\_LINK\_KEY\_REQUEST\_NEGATIVE\_REPLY HCI\_OPCODE(OGF\_LINK\_CONTROL, 0x000C)
- #define HCI PIN CODE REQUEST REPLY HCI OPCODE(OGF LINK CONTROL, 0x000D)
- #define HCI\_PIN\_CODE\_REQUEST\_NEGATIVE\_REPLY HCI\_OPCODE(OGF\_LINK\_CONTROL, 0x000E)
- #define HCI CHANGE CONNECTION PACKET TYPE HCI OPCODE(OGF LINK CONTROL, 0x000F)
- #define HCI AUTHENTICATION REQUESTED HCI OPCODE(OGF LINK CONTROL, 0x0011)
- #define HCI\_SET\_CONNECTION\_ENCRYPTION HCI\_OPCODE(OGF\_LINK\_CONTROL, 0x0013)
- #define HCI CHANGE CONNECTION LINK KEY HCI OPCODE(OGF LINK CONTROL, 0x0015)
- #define HCI\_MASTER\_LINK\_KEY HCI\_OPCODE(OGF\_LINK\_CONTROL, 0x0017)
- #define HCI\_REMOTE\_NAME\_REQUEST HCI\_OPCODE(OGF\_LINK\_CONTROL, 0x0019)
- #define HCI\_REMOTE\_NAME\_REQUEST\_CANCEL HCI\_OPCODE(OGF\_LINK\_CONTROL, 0x001A)
- #define  $HCI_READ_REMOTE_SUPPORTED_FEATURES$   $HCI_OPCODE(OGF_LINK_CONTRO \leftarrow L, 0x001B)$
- #define HCI READ REMOTE EXTENDED FEATURES HCI OPCODE(OGF LINK CONTROL, 0x001C)
- #define HCI\_READ\_CLOCK\_OFFSET HCI\_OPCODE(OGF\_LINK\_CONTROL, 0x001F)
- #define HCI\_READ\_LMP\_HANDLE HCI\_OPCODE(OGF\_LINK\_CONTROL, 0x0020)
- #define HCI SETUP SYNCHRONOUS CONNECTION HCI OPCODE(OGF LINK CONTROL, 0x0028)
- #define HCI\_ACCEPT\_SYNCH\_CONNECTION\_REQUEST HCI\_OPCODE(OGF\_LINK\_CONTRO ← L, 0x0029)
- #define HCI\_REJECT\_SYNCH\_CONNECTION\_REQUEST HCI\_OPCODE(OGF\_LINK\_CONTRO← L, 0x002A)
- #define HCI IO CAPABILITY REQUEST REPLY HCI OPCODE(OGF LINK CONTROL, 0x002B)
- #define HCI\_USER\_CONFIRMATION\_REQ\_NEGATIVE\_REPLY HCI\_OPCODE(OGF\_LINK\_CONTRO

   L, 0x002D)
- #define HCI\_USER\_PASSKEY\_REQUEST\_REPLY HCI\_OPCODE(OGF\_LINK\_CONTROL, 0x002E)
- #define HCI\_USER\_PASSKEY\_REQUEST\_NEGATIVE\_REPLY HCI\_OPCODE(OGF\_LINK\_CONTROL, 0x002F)
- #define HCI\_REMOTE\_OOB\_DATA\_REQUEST\_REPLY HCI\_OPCODE(OGF\_LINK\_CONTROL, 0x0030)
- #define HCI\_REMOTE\_OOB\_DATA\_REQUEST\_NEGATIVE\_REPLY HCI\_OPCODE(OGF\_LINK\_CONT
   — ROL, 0x0033)
- #define HCI\_IO\_CAPABILITY\_REQUEST\_NEGATIVE\_REPLY HCI\_OPCODE(OGF\_LINK\_CONTROL, 0x0034)

3.2 HCI 19

# Link policy commands

The Link Policy Commands provide methods for the Host to affect how the Link Manager manages the piconet.

- #define HCI\_HOLD\_MODE HCI\_OPCODE(OGF\_LINK\_POLICY, 0x0001)
- #define HCI\_SNIFF\_MODE HCI\_OPCODE(OGF\_LINK\_POLICY, 0x0003)
- #define HCI\_EXIT\_SNIFF\_MODE HCI\_OPCODE(OGF\_LINK\_POLICY, 0x0004)
- #define HCI\_PARK\_STATE HCI\_OPCODE(OGF\_LINK\_POLICY, 0x0005)
- #define HCI EXIT PARK STATE HCI OPCODE(OGF LINK POLICY, 0x0006)
- #define HCI QOS SETUP HCI OPCODE(OGF LINK POLICY, 0x0007)
- #define HCI\_ROLE\_DISCOVERY HCI\_OPCODE(OGF\_LINK\_POLICY, 0x0009)
- #define HCI SWITCH ROLE HCI OPCODE(OGF LINK POLICY, 0x000B)
- #define HCI\_READ\_LINK\_POLICY\_SETTINGS HCI\_OPCODE(OGF\_LINK\_POLICY, 0x000C)
- #define HCI WRITE LINK POLICY SETTINGS HCI OPCODE(OGF LINK POLICY, 0x000D)
- #define HCI\_READ\_DEFAULT\_POLICY\_SETTINGS HCI\_OPCODE(OGF\_LINK\_POLICY, 0x000E)
- #define HCI WRITE DEFAULT POLICY SETTINGS HCI OPCODE(OGF LINK POLICY, 0x000F)
- #define HCI FLOW SPECIFICATION HCI OPCODE(OGF LINK POLICY, 0x0010)
- #define HCI\_SNIFF\_SUBRATING HCI\_OPCODE(OGF\_LINK\_POLICY, 0x0011)

## **Controller & Baseband commands**

The Controller & Baseband Commands provide access and control to various capabilities of the Bluetooth hardware.

- #define HCI SET EVENT MASK HCI OPCODE(OGF CTRL BASEBAND, 0x0001)
- #define HCI\_RESET HCI\_OPCODE(OGF\_CTRL\_BASEBAND, 0x0003)
- #define HCI SET\_EVENT\_FILTER HCI OPCODE(OGF CTRL BASEBAND, 0x0005)
- #define HCI\_FLUSH HCI\_OPCODE(OGF\_CTRL\_BASEBAND, 0x0008)
- #define HCI\_READ\_PIN\_TYPE HCI\_OPCODE(OGF\_CTRL\_BASEBAND, 0x0009)
- #define HCI\_WRITE\_PIN\_TYPE HCI\_OPCODE(OGF\_CTRL\_BASEBAND, 0x000A)
- #define HCI\_CREATE\_NEW\_UNIT\_KEY HCI\_OPCODE(OGF\_CTRL\_BASEBAND, 0x0000B)
- #define HCI\_READ\_STORED\_LINK\_KEY HCI\_OPCODE(OGF\_CTRL\_BASEBAND, 0x000D)
- #define HCI\_WRITE\_STORED\_LINK\_KEY HCI\_OPCODE(OGF\_CTRL\_BASEBAND, 0x0011)
- #define **HCI\_DELETE\_STORED\_LINK\_KEY** HCI\_OPCODE(OGF\_CTRL\_BASEBAND, 0x0012)
- #define HCI WRITE LOCAL NAME HCI OPCODE(OGF CTRL BASEBAND, 0x0013)
- #define HCI READ LOCAL NAME HCI OPCODE(OGF CTRL BASEBAND, 0x0014)
- #define HCI\_READ\_CONNECTION\_ACCEPT\_TIMEOUT HCI\_OPCODE(OGF\_CTRL\_BASEBAN 

  D, 0x0015)
- #define HCI\_WRITE\_CONNECTION\_ACCEPT\_TIMEOUT HCI\_OPCODE(OGF\_CTRL\_BASEBAN ← D, 0x0016)
- #define HCI READ PAGE TIMEOUT HCI OPCODE(OGF CTRL BASEBAND, 0x0017)
- #define HCI WRITE PAGE TIMEOUT HCI OPCODE(OGF CTRL BASEBAND, 0x0018)
- #define HCI\_READ\_SCAN\_ENABLE HCI\_OPCODE(OGF\_CTRL\_BASEBAND, 0x0019)
- #define HCI WRITE SCAN ENABLE HCI OPCODE(OGF CTRL BASEBAND, 0x001A)
- #define HCI\_READ\_PAGE\_SCAN\_ACTIVITY HCI\_OPCODE(OGF\_CTRL\_BASEBAND, 0x001B)
- #define HCI\_WRITE\_PAGE\_SCAN\_ACTIVITY HCI\_OPCODE(OGF\_CTRL\_BASEBAND, 0x001C)
- #define HCI READ INQUIRY SCAN ACTIVITY HCI OPCODE(OGF CTRL BASEBAND, 0x001D)
- #define HCI\_WRITE\_INQUIRY\_SCAN\_ACTIVITY HCI\_OPCODE(OGF\_CTRL\_BASEBAND, 0x001E)
- #define HCI\_READ\_AUTHENTICATION\_ENABLE HCI\_OPCODE(OGF\_CTRL\_BASEBAND, 0x001F)
- #define HCI\_WRITE\_AUTHENTICATION\_ENABLE HCI\_OPCODE(OGF\_CTRL\_BASEBAND, 0x0020)
- #define HCI\_READ\_ENCRYPTION\_MODE HCI\_OPCODE(OGF\_CTRL\_BASEBAND, 0x0021)
- #define HCI\_WRITE\_ENCRYPTION\_MODE HCI\_OPCODE(OGF\_CTRL\_BASEBAND, 0x0022)
- #define HCI READ CLASS OF DEVICE HCI OPCODE(OGF CTRL BASEBAND, 0x0023)
- #define HCI WRITE CLASS OF DEVICE HCI OPCODE(OGF CTRL BASEBAND, 0x0024)
- #define HCI\_READ\_VOICE\_SETTING HCI\_OPCODE(OGF\_CTRL\_BASEBAND, 0x0025)

- #define HCI\_WRITE\_VOICE\_SETTING HCI\_OPCODE(OGF\_CTRL\_BASEBAND, 0x0026)
- #define HCI\_READ\_AUTOMATIC\_FLASH\_TIMEOUT HCI\_OPCODE(OGF\_CTRL\_BASEBAND, 0x0027)
- #define HCI WRITE AUTOMATIC FLASH TIMEOUT HCI OPCODE(OGF CTRL BASEBAND, 0x0028)
- #define HCI READ NUM BROADCST RETR HCI OPCODE(OGF CTRL BASEBAND, 0x0029)
- #define HCI\_WRITE\_NUM\_BROADCST\_RETR HCI\_OPCODE(OGF\_CTRL\_BASEBAND, 0x002A)
- #define HCI\_READ\_HOLD\_MODE\_ACTIVITY HCI\_OPCODE(OGF\_CTRL\_BASEBAND, 0x002B)
- #define HCI WRITE HOLD MODE ACTIVITY HCI OPCODE(OGF CTRL BASEBAND, 0x002C)
- #define HCI READ TRANSMIT POWER LEVEL HCI OPCODE(OGF CTRL BASEBAND, 0x002D)
- #define HCI\_READ\_SYNC\_FLOW\_CONTROL\_ENABLE HCI\_OPCODE(OGF\_CTRL\_BASEBAN ← D, 0x002E)
- #define HCI\_WRITE\_SYNC\_FLOW\_CONTROL\_ENABLE HCI\_OPCODE(OGF\_CTRL\_BASEBAN ← D, 0x002F)
- #define HCI\_SET\_CTRL\_TO\_HOST\_FLOW\_CONTROL HCI\_OPCODE(OGF\_CTRL\_BASEBAND, 0x0031)
- #define HCI\_HOST\_BUFFER\_SIZE HCI\_OPCODE(OGF\_CTRL\_BASEBAND, 0x0033)
- #define HCI\_HOST\_NUM\_OF\_COMPLETED\_PACKETS HCI\_OPCODE(OGF\_CTRL\_BASEBAN ← D, 0x0035)
- #define HCI READ LINK SUPERVISION TIMEOUT HCI OPCODE(OGF CTRL BASEBAND, 0x0036)
- #define HCI\_WRITE\_LINK\_SUPERVISION\_TIMEOUT HCI\_OPCODE(OGF\_CTRL\_BASEBAND, 0x0037)
- #define HCI READ NUM OF SUPPORTED IAC HCI OPCODE(OGF CTRL BASEBAND, 0x0038)
- #define HCI READ CURRENT IAC LAP HCI OPCODE(OGF CTRL BASEBAND, 0x0039)
- #define HCI WRITE CURRENT IAC LAP HCI OPCODE(OGF CTRL BASEBAND, 0x003A)
- #define HCI\_READ\_PAGE\_SCAN\_PERIOD\_MODE HCI\_OPCODE(OGF\_CTRL\_BASEBAND, 0x003B)
- #define HCI\_WRITE\_PAGE\_SCAN\_PERIOD\_MODE HCI\_OPCODE(OGF\_CTRL\_BASEBAND, 0x003C)
- #define HCI\_SET\_AFH\_HOST\_CHANNEL\_CLASSIFICATION HCI\_OPCODE(OGF\_CTRL\_BASEBAND, 0x003F)
- #define HCI READ INQUIRY SCAN TYPE HCI OPCODE(OGF CTRL BASEBAND, 0x0042)
- #define HCI WRITE INQUIRY SCAN TYPE HCI OPCODE(OGF CTRL BASEBAND, 0x0043)
- #define HCI\_READ\_INQUIRY\_MODE HCI\_OPCODE(OGF\_CTRL\_BASEBAND, 0x0044)
- #define HCI\_WRITE\_INQUIRY\_MODE HCI\_OPCODE(OGF\_CTRL\_BASEBAND, 0x0045)
- #define HCI READ PAGE SCAN TYPE HCI OPCODE(OGF CTRL BASEBAND, 0x0046)
- #define HCI WRITE PAGE SCAN TYPE HCI OPCODE(OGF CTRL BASEBAND, 0x0047)
- #define HCI\_READ\_AFH\_CHANNEL\_ASSESSMENT\_MODE HCI\_OPCODE(OGF\_CTRL\_BASEBAND, 0x0048)
- #define HCI\_WRITE\_AFH\_CHANNEL\_ASSESSMENT\_MODE HCI\_OPCODE(OGF\_CTRL\_BASEBAND, 0x0049)
- #define HCI\_READ\_EXTENDED\_INQUIRY\_RESPONSE HCI\_OPCODE(OGF\_CTRL\_BASEBAN ← D, 0x0051)
- #define HCI\_WRITE\_EXTENDED\_INQUIRY\_RESPONSE HCI\_OPCODE(OGF\_CTRL\_BASEBAN ← D, 0x0052)
- #define HCI READ REFRESH ENCRYPTION KEY HCI OPCODE(OGF CTRL BASEBAND, 0x0053)
- #define HCI\_READ\_SIMPLE\_PAIRING\_MODE HCI\_OPCODE(OGF\_CTRL\_BASEBAND, 0x0055)
- #define HCI\_WRITE\_SIMPLE\_PAIRING\_MODE HCI\_OPCODE(OGF\_CTRL\_BASEBAND, 0x0056)
- #define HCI READ LOCAL OOB DATA HCI OPCODE(OGF CTRL BASEBAND, 0x0057)
- #define HCI\_WRITE\_INQUIRY\_TX\_POWER\_LEVEL HCI\_OPCODE(OGF\_CTRL\_BASEBAND, 0x0059)
- #define HCI\_READ\_DEFAULT\_ERRONEOUS\_DATA\_REPORTING HCI\_OPCODE(OGF\_CTRL\_BASE ← BAND, 0x005A)
- #define HCI\_WRITE\_DEFAULT\_ERRONEOUS\_DATA\_REPORTING HCI\_OPCODE(OGF\_CTRL\_BASE ← BAND, 0x005B)
- #define HCI\_ENHANCED\_FLUSH HCI\_OPCODE(OGF\_CTRL\_BASEBAND, 0x005F)
- #define HCI SEND KEY PRESS NOTIFICATION HCI OPCODE(OGF CTRL BASEBAND, 0x0060)
- #define HCI READ LE HOST SUPPORT HCI OPCODE(OGF CTRL BASEBAND, 0x006C)
- #define HCI\_WRITE\_LE\_HOST\_SUPPORT HCI\_OPCODE(OGF\_CTRL\_BASEBAND, 0x006D)

3.2 HCI 21

## **Informational Parameters**

The Informational Parameters are fixed by the manufacturer of the Bluetooth hardware. These parameters provide information about the BR/EDR Controller and the capabilities of the Link Manager and Baseband in the BR/EDR Controller and PAL in the AMP Controller. The host device cannot modify any of these parameters.

- #define HCI READ LOCAL VERSION INFORMATION HCI OPCODE(OGF INFORMATION, 0x0001)
- #define HCI\_READ\_LOCAL\_SUPPORTED\_COMMANDS HCI\_OPCODE(OGF\_INFORMATION, 0x0002)
- #define HCI\_READ\_LOCAL\_SUPPORTED\_FEATURES HCI\_OPCODE(OGF\_INFORMATION, 0x0003)
- #define HCI READ LOCAL EXTENDED FEATURES HCI OPCODE(OGF INFORMATION, 0x0004)
- #define HCI READ BUFFER SIZE HCI OPCODE(OGF INFORMATION, 0x0005)
- #define HCI\_READ\_BD\_ADDR HCI\_OPCODE(OGF\_INFORMATION, 0x0009)

## **Status Parameters**

The Controller modifies all status parameters. These parameters provide information about the current state of the Link Manager and Baseband in the BR/EDR Controller and the PAL in an AMP Controller. The host device cannot modify any of these parameters other than to reset certain specific parameters.

- #define HCI READ FAILED CONTACT COUNTER HCI OPCODE(OGF STATUS, 0x0001)
- #define HCI RESET FAILED CONTACT COUNTER HCI OPCODE(OGF STATUS, 0x0002)
- #define HCI\_READ\_LINK\_QUALITY HCI\_OPCODE(OGF\_STATUS, 0x0003)
- #define HCI\_READ\_RSSI HCI\_OPCODE(OGF\_STATUS, 0x0005)
- #define HCI READ AFH CHANNEL MAP HCI OPCODE(OGF STATUS, 0x0006)
- #define HCI\_READ\_CLOCK\_COMMAND HCI\_OPCODE(OGF\_STATUS, 0x0007)

# **Testing Commands**

The Testing commands are used to provide the ability to test various functional capabilities of the Bluetooth hardware.

- #define HCI READ LOOPBACK MODE HCI OPCODE(OGF TESTING, 0x0001)
- #define HCI WRITE LOOPBACK MODE HCI OPCODE(OGF TESTING, 0x0002)
- #define HCI\_ENABLE\_DEVICE\_UNDER\_TEST\_MODE HCI\_OPCODE(OGF\_TESTING, 0x0003)
- #define HCI\_WRITE\_SIMPLE\_PAIRING\_DEBUG\_MODE HCI\_OPCODE(OGF\_TESTING, 0x0004)

# LE controller commands

The LE Controller Commands provide access and control to various capabilities of the Bluetooth hardware, as well as methods for the Host to affect how the Link Layer manages the piconet, and controls connections.

- #define HCI\_LE\_SET\_EVENT\_MASK HCI\_OPCODE(OGF\_LE, 0x0001)
- #define HCI\_LE\_READ\_BUFFER\_SIZE HCI\_OPCODE(OGF\_LE, 0x0002)
- #define HCI\_LE\_READ\_LOCAL\_SUPPORTED\_FEATURES HCI\_OPCODE(OGF\_LE, 0x0003)
- #define HCI\_LE\_SET\_RANDOM\_ADDRESS HCI\_OPCODE(OGF\_LE, 0x0005)
- #define HCI\_LE\_SET\_ADVERTISING\_PARAMETERS HCI\_OPCODE(OGF\_LE, 0x0006)
- #define HCI\_LE\_READ\_ADVERTISING\_CHANNEL\_TX\_POWER HCI\_OPCODE(OGF\_LE, 0x0007)
- #define HCI\_LE\_SET\_ADVERTISING\_DATA HCI\_OPCODE(OGF\_LE, 0x00008)
- #define HCI\_LE\_SET\_SCAN\_RESPONSE\_DATA HCI\_OPCODE(OGF\_LE, 0x0009)
- #define HCI\_LE\_SET\_ADVERTISE\_ENABLE HCI\_OPCODE(OGF\_LE, 0x000A)
- #define HCI\_LE\_SET\_SCAN\_PARAMETERS HCI\_OPCODE(OGF\_LE, 0x0000B)
- #define HCI LE SET SCAN ENABLE HCI OPCODE(OGF LE, 0x000C)
- #define HCI\_LE\_CREATE\_CONNECTION HCI\_OPCODE(OGF\_LE, 0x000D)

- #define HCI\_LE\_CREATE\_CONNECTION\_CANCEL HCI\_OPCODE(OGF\_LE, 0x000E)
- #define HCI\_LE\_READ\_WHITE\_LIST\_SIZE HCI\_OPCODE(OGF\_LE, 0x000F)
- #define HCI LE CLEAR WHITE LIST HCI OPCODE(OGF LE, 0x0010)
- #define HCI LE ADD DEVICE TO WHITE LIST HCI OPCODE(OGF LE, 0x0011)
- #define HCI LE REMOVE DEVICE FROM WHITE LIST HCI OPCODE(OGF LE, 0x0012)
- #define HCI LE CONNECTION UPDATE HCI OPCODE(OGF LE, 0x0013)
- #define HCI\_LE\_SET\_HOST\_CHANNEL\_CLASSIFICATION HCI\_OPCODE(OGF\_LE, 0x0014)
- #define HCI LE READ CHANNEL MAP HCI OPCODE(OGF LE, 0x0015)
- #define HCI LE READ REMOTE USED FEATURES HCI OPCODE(OGF LE, 0x0016)
- #define HCI\_LE\_ENCRYPT HCI\_OPCODE(OGF\_LE, 0x0017)
- #define HCI LE RAND HCI OPCODE(OGF LE, 0x0018)
- #define HCI LE START ENCRYPTION HCI OPCODE(OGF LE, 0x0019)
- #define HCI LE LONG TERM KEY REQUEST REPLY HCI OPCODE(OGF LE, 0x001A)
- #define HCI\_LE\_LONG\_TERM\_KEY\_REQUEST\_NEGATIVE\_REPLY HCI\_OPCODE(OGF\_LE, 0x001B)
- #define HCI LE READ SUPPORTED STATES HCI OPCODE(OGF LE, 0x001C)
- #define HCI\_LE\_RECEIVE\_TEST HCI\_OPCODE(OGF\_LE, 0x001D)
- #define HCI\_LE\_TRANSMITTER\_TEST HCI\_OPCODE(OGF\_LE, 0x001E)
- #define HCI LE TEST END HCI OPCODE(OGF LE, 0x001F)

## **Events**

- #define HCI\_EVT\_INQUIRY\_COMPLETE 0x01
- #define HCI EVT INQUIRY RESULT 0x02
- #define HCI EVT CONNECTION COMPLETE 0x03
- #define HCI EVT CONNECTION REQUEST 0x04
- #define HCI EVT DISCONNECTION COMPLETE 0x05
- #define HCI\_EVT\_AUTHENTICATION\_COMPLETE 0x06
- #define HCI EVT REMOTE NAME REQUEST COMPLETE 0x07
- #define HCI EVT ENCRYPTION CHANGE 0x08
- #define HCI EVT CHANGE CONN LINK COMPLETE 0x09
- #define HCI EVT MASTER LINK KEY COMPLETE 0x0A
- #define HCI\_EVT\_READ\_RMT\_SUP\_FEATURES\_COMP 0x0B
- #define HCI EVT READ RMT VERSION INFO COMP 0x0C
- #define HCI EVT QOS SETUP COMPLETE 0x0D
- #define HCI\_EVT\_COMMAND\_COMPLETE 0x0E
- #define HCI\_EVT\_COMMAND\_STATUS 0x0F
- #define HCI\_EVT\_HARDWARE\_ERROR 0x10
- #define HCI\_EVT\_FLUSH\_OCCURED 0x11
- #define HCI\_EVT\_ROLE\_CHANGE 0x12
- #define HCI\_EVT\_NUM\_OF\_COMPLETED\_PACKETS 0x13
- #define HCI\_EVT\_MODE\_CHANGE 0x14
- #define HCI\_EVT\_RETURN\_LINK\_KEYS 0x15
- #define HCI EVT PIN CODE REQUEST 0x16
- #define HCI\_EVT\_LINK\_KEY\_REQUEST 0x17
- #define HCI\_EVT\_LINK\_KEY\_NOTIFICATION 0x18
- #define HCI\_EVT\_LOOPBACK\_COMMAND 0x19
- #define HCI\_EVT\_DATA\_BUFFER\_OVERFLOW 0x1A
- #define HCI\_EVT\_MAX\_SLOTS\_CHANGE 0x1B
- #define HCI\_EVT\_READ\_CLOCK\_OFFSET\_COMPLETE 0x1C
- #define HCI\_EVT\_CONN\_PACKET\_TYPE\_CHANGED 0x1D
- #define HCI\_EVT\_QOS\_VIOLATION 0x1E
- #define HCI\_EVT\_PAGE\_SCAN\_REPET\_MODE\_CHANGE 0x20
- #define HCI\_EVT\_FLOW\_SPECIFICATION\_COMPLETE 0x21
- #define HCI\_EVT\_INQUIRY\_RESULT\_WITH\_RSSI 0x22

3.2 HCI 23

- #define HCI\_EVT\_READ\_RMT\_EXT\_FEATURES\_COMP 0x23
- #define HCI\_EVT\_SYNCH\_CONNECTION\_COMPLETE 0x2C
- #define HCI\_EVT\_SYNCH\_CONNECTION\_CHANGED 0x2D
- #define HCI EVT\_SNIFF SUBRATING 0x2E
- #define HCI EVT EXTENDED INQUIRY RESULT 0x2F
- #define HCI\_EVT\_ENCRYPTION\_KEY\_REFRESH\_COMPLETE 0x30
- #define HCI\_EVT\_IO\_CAPABILITY\_REQUEST 0x31
- #define HCI EVT IO CAPABILITY RESPONSE 0x32
- #define HCI EVT USER CONFIRMATION REQUEST 0x33
- #define HCI\_EVT\_USER\_PASSKEY\_REQUEST\_0x34
- #define HCI EVT REMOTE OOB DATA REQUEST 0x35
- #define HCI\_EVT\_SIMPLE\_PAIRING\_COMPLETE 0x36
- #define HCI\_EVT\_LINK\_SUPERVISION\_TO\_CHANGED 0x38
- #define HCI\_EVT\_ENHANCED\_FLUSH\_COMPLETE 0x39
- #define HCI\_EVT\_USER\_PASSKEY\_NOTIFICATION 0x3B
- #define HCI EVT KEYPRESS NOTIFICATION 0x3C
- #define HCI\_EVT\_RMT\_HOST\_SUPP\_FEATURES\_NTF 0x3D
- #define HCI\_EVT\_LE\_META\_EVENT 0x3E
- #define HCI\_EVT\_LAST HCI\_EVT\_LE\_META\_EVENT
- #define HCI\_EVT\_FIRST HCI\_EVT\_INQUIRY\_COMPLETE

#### **Errors**

- #define HCI ERR SUCCESS 0x00
- #define HCI SUCCESS 0x00
- #define HCI ERR AUTHENTICATION FAILURE 0x05
- #define HCI\_ERR\_MEMORY\_CAPACITY\_EXCEEDED 0x07
- #define HCI\_ERR\_CONNECTION\_TIMEOUT 0x08
- #define HCI ERR SCO CONN LIMIT EXCEEDED 0x0a
- #define HCI ERR ACL CONN ALREADY EXISTS 0x0b
- #define HCI\_ERR\_CONN\_REJECT\_LIMITED\_RESOURCES 0x0d
- #define HCI\_ERR\_INVALID\_PARAMETERS 0x12
- #define HCI\_ERR\_UNSPECIFIED 0x1F
- #define HCI ERR SIMPLE PAIRING NOT SUPPORTED 0x37

# 3.2.1 Detailed Description

The Host Controller Interface (HCI) provides a uniform interface method of accessing a Bluetooth Controller's capabilities.

This module describes function and data structures used to send HCl commands and receive responses.

## 3.2.2 Typedef Documentation

3.2.2.1 typedef void(\* bt\_hci\_connect\_callback\_fp) (bt\_byte status, bt\_hci\_conn\_state\_t \*pconn, void \*param)

HCI connect callback.

This typedef defines a type for the callback function that is called when HCI connect operation initiated by a call to bt\_hci\_connect() is complete.

#### **Parameters**

status	Operation status. It is 0 if connection was successfully established.
pconn	pointer to a structure representing the established connection.
param	pointer to arbitrary data passed to the bt_hci_connect() function through its param parame-
	ter.

3.2.2.2 typedef void(\* bt\_hci\_disconnect\_callback\_fp) (bt\_byte status, bt\_byte reason, bt\_hci\_conn\_state\_t \*pconn, void \*param)

HCI disconnect callback.

This typedef defines a type for the callback function that is called when an HCI connection has been terminated.

#### **Parameters**

status	Operation status. It is 0 if connection has been successfully terminated.
reason	Reason for disconnection.
pconn	pointer to a structure representing the connection.
param	pointer to arbitrary data associated with an event listener.

3.2.2.3 typedef void(\* bt\_hci\_start\_callback\_fp) (bt\_bool success, void \*param)

HCI initialization callback.

This typedef defines a function pointer type for the HCI initialization callback functions. Such a function must be passed to the <a href="https://bci.nction.com/bci.nction.">bt\_hci\_start()</a> function.

#### **Parameters**

success	Specifies whether HCI initialization succeeded or not.

3.2.2.4 typedef void(\* bt\_hci\_stop\_callback\_fp) (void \*param)

HCI stop callback.

This typedef defines a function pointer type for the HCI stop callback functions. Such a function must be passed to the <a href="https://bci.ncbi.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/bci.nlm.nih.gov/

# 3.2.3 Function Documentation

3.2.3.1 bt\_bool bt\_hci\_connect ( bt\_bdaddr\_p dest, bt\_uint packet\_type, bt\_byte pg\_scan\_rpt\_mode, bt\_byte role\_switch, bt\_uint acl\_config, bt\_hci\_connect\_callback\_fp callback, void \* param )

Connect to a remote device.

This function tries to establish an HCI connection with a remote device specified by the Bluetooth address dest. Upon completion, the callback function specified by the callback parameter is called.

# **Parameters**

dest	Bluetooth address of the remote device.
packet_type	

3.2 HCI 25

role_switch	
acl_config	
callback	Pointer to a callback function that is called when the connect operation completes.
param	Pointer to arbitrary data that is to be passed to the callback function.

## Returns

- TRUE when the function succeeds.
- FALSE otherwise. The callback function is not called in this case.

# 3.2.3.2 void bt\_hci\_init ( void )

Initialize the HCI layer.

This function initializes all internal variables of the HCI layer. The application, unless it's going to use only HCI layer, does not need to call this function as it is implicitly called by bt\_sys\_init.

This function essentially calls bt\_hci\_init\_ex(HCI\_LINK\_POLICY\_ENABLE\_ALL) so all link policy setting are enabled.

# 3.2.3.3 void bt\_hci\_init\_ex ( bt\_byte default\_link\_policy )

Initialize the HCI layer.

This function initializes all internal variables of the HCl layer. The application, unless it's going to use only HCl layer, does not need to call this function as it is implicitly called by bt\_sys\_init\_ex.

#### **Parameters**

default_link_← policy	default link policy settings. This is a bitmask that defines the initial value of the link policy settings for all new BR/EDR connections. This value can be a combination of the following values:
	HCI_LINK_POLICY_ENABLE_ROLE_SWITCH
	HCI_LINK_POLICY_ENABLE_HOLD_MODE
	HCI_LINK_POLICY_ENABLE_SNIFF_MODE
	HCI_LINK_POLICY_ENABLE_PARK_STATE
	To enable all settings pass HCI_LINK_POLICY_ENABLE_ALL.

# 3.2.3.4 bt\_bool bt\_hci\_listen ( bt\_hci\_connect\_callback\_fp $\it cb$ , void $\it * param$ )

Listen for incoming connections.

# **Parameters**

callback	Pointer to a callback function that is called when a new incoming connection has been established.
param	Pointer to arbitrary data that is to be passed to the callback function.

# Returns

- TRUE when the function succeeds.
- FALSE otherwise. The callback function is not called in this case.

3.2.3.5 bt\_bool bt\_hci\_reset ( bt\_hci\_cmd\_callback\_fp callback, void \* callback\_param )

Reset controller.

This function resets the BT controller.

#### **Parameters**

ſ	callback	Completion callback. Called when the controller has been reset.
ſ	callback_param	A pointer to arbitrary data to be passed to the callback callback.

3.2.3.6 bt\_bool bt\_hci\_start ( bt\_hci\_start\_callback\_fp callback, void \* callback\_param, bt\_byte enable\_scan )

Start HCI layer.

This function starts the HCI layer of the stack. Starting the HCI layer consists essentially of two steps:

- 1. Make the HCI transport receive packets from the controller. This results in a call to bt\_oem\_recv.
- 2. Reset and configure the controller.

Upon completion of controller initialization the callback function passed in the callback parameter is called.

#### **Parameters**

callback	Completion callback. Called when controller initialization is complete.
callback_param	A pointer to arbitrary data to be passed to the callback callback.
enable_scan	This is a bitmask that defines which scans are enabled during the controller configuration.
	This value can be a combination of the following values:
	HCI_SCAN_INQUIRY (the controller is discoverable)
	HCI_SCAN_PAGE (the controller is connectable)

## Returns

- TRUE when the function succeeds.
- FALSE otherwise. The callback function is not called in this case.

3.2.3.7 void bt\_hci\_start\_no\_init ( void )

Start HCI layer without controller configuration.

This function is similar to bt\_hci\_start but unlike the former it does not perform the controller configuration. I.e., bt\_← hci\_start\_no\_init simply calls the HCI transport and makes it receive packets from the controller. The main purpose of this function is make the HCI transport ready to exchange packets if controller needs some vendor specific configuration before it can be used with the stack. E.g., controllers based on CRS8811 chip need loading various values that configure its operating mode using CSR's proprietary protocol. So the application after configuring the HCI transport would call bt\_hci\_init(), bt\_hci\_start\_no\_init() and then load configuration values. Once the vendor specific configuration is done, the application will re-initialize the HCI transport and perform full start of the stack with bt\_sys\_init() and bt\_sys\_start().

3.2.3.8 void bt\_hci\_stop ( bt\_hci\_stop\_callback\_fp callback, void \* callback\_param )

Stop HCI layer.

This function makes the HCl layer inoperable. After this call the application must perform the full reset of the HCl transport and stack.

3.2 HCI 27

# **Parameters**

callback	Completion callback. Called when the HCI layer has been stopped.
callback_param	A pointer to arbitrary data to be passed to the callback callback.

3.2.3.9 bt\_bool bt\_hci\_write\_local\_name ( const char \* device\_name, bt\_hci\_cmd\_callback\_fp cb )

Write local device name.

The Write\_Local\_Name command provides the ability to modify the userfriendly name for the BR/EDR Controller.

# 3.3 HCI UART (H4) transport protocol

## **Functions**

void bt\_hcitr\_uart\_init (void)

Initialize HCI UART (H4) transport protocol.

void bt\_hcitr\_uart\_reset (void)

Re-initialize HCI UART (H4) transport protocol.

void bt\_hcitr\_uart\_start (void)

Start HCI UART (H4) transport protocol.

# 3.3.1 Detailed Description

This module describes functions used to initialize and start HCI UART transport protocol. The transport uses common interface for exchanging data between the host CPU and HCI controller defined in bt\_hcitr.h. This interface consist of two functions that must be implemented by the application:

- bt\_oem\_send()
- bt\_oem\_recv()

#### 3.3.2 Function Documentation

```
3.3.2.1 void bt_hcitr_uart_init ( void )
```

Initialize HCI UART (H4) transport protocol.

This function initializes internal structures of the transport. The application must call it as early as possible before bt\_hcitr\_uart\_start and before the stack is initialized and started with bt\_sys\_init and bt\_sys\_start.

```
3.3.2.2 void bt_hcitr_uart_reset ( void )
```

Re-initialize HCI UART (H4) transport protocol.

This function re-initializes the transport. Currently it simply calls bt\_hcitr\_uart\_init. After calling this function the application must perform the full initialization of the stack by calling bt\_sys\_init, bt\_sys\_start and initialization functions of all other profile modules the application is intending to use.

```
3.3.2.3 void bt_hcitr_uart_start (void)
```

Start HCI UART (H4) transport protocol.

This function starts the transport, i.e., makes it able to receive and send packets.

# 3.4 Serial Port Profile (SPP)

The DotStack SPP API is a simple API for communicating over a Bluetooth link using the Bluetooth Serial Port Profile.

## **Modules**

• SPP Configuration

This module describes parameters used to configure SPP layer.

RFCOMM Configuration

This module describes parameters used to configure RFCOMM layer.

## **Data Structures**

struct bt\_spp\_port\_t

Serial port structure.

· struct bt\_spp\_port\_t::\_bt\_spp\_port\_flags\_t

# **Typedefs**

- typedef void(\* bt\_spp\_state\_callback\_fp) (bt\_spp\_port\_t \*port, bt\_spp\_port\_event\_e evt, void \*param)

  Serial port state callback.
- typedef void(\* bt\_spp\_send\_callback\_fp) (bt\_spp\_port\_t \*port, bt\_ulong bytes\_sent, bt\_spp\_send\_status\_e status, void \*param)

Serial port send callback.

typedef void(\* bt\_spp\_receive\_callback\_fp) (bt\_spp\_port\_t \*port, bt\_int bytes\_received, void \*param)
 Serial port receive callback.

# **Enumerations**

```
    enum bt_spp_port_state_e {
        SPP_PORT_STATE_FREE, SPP_PORT_STATE_DISCONNECTED, SPP_PORT_STATE_DISCONNEC
        TING, SPP_PORT_STATE_CONNECTING,
        SPP_PORT_STATE_CONNECTED }
```

Serial port state.

enum bt\_spp\_port\_event\_e {
 SPP\_PORT\_EVENT\_CONNECTED = 1, SPP\_PORT\_EVENT\_DISCONNECTED, SPP\_PORT\_EVENT\_
 CONNECTION\_FAILED, SPP\_PORT\_EVENT\_SEND\_PROGRESS,
 SPP\_PORT\_EVENT\_REMOTE\_MODEM\_STATUS\_CHANGED, SPP\_PORT\_EVENT\_LOCAL\_MODEM
 \_STATUS\_CHANGED, SPP\_PORT\_EVENT\_LOCAL\_MODEM\_STATUS\_CHANGE\_FAILED }

Serial port event.

enum bt\_spp\_send\_status\_e {
 SPP\_SEND\_STATUS\_SUCCESS = 0, SPP\_SEND\_STATUS\_TIMEOUT, SPP\_SEND\_STATUS\_NOT\_E
 NOUGH\_RESOURCES, SPP\_SEND\_STATUS\_CANCELED,
 SPP\_SEND\_STATUS\_INTERRUPTED }

Send operation status.

## **Functions**

void bt\_spp\_init (void)

Initialize the SPP module.

 bt\_spp\_port\_t \* bt\_spp\_allocate (bt\_l2cap\_mgr\_t \*l2cap\_mgr, bt\_spp\_state\_callback\_fp callback, void \*callback param)

Allocate a serial port.

bt\_bool bt\_spp\_listen (bt\_spp\_port\_t \*port, bt\_byte channel)

Listen for incoming connections.

bt\_bool bt\_spp\_connect (bt\_spp\_port\_t \*port, bt\_bdaddr\_p remote\_addr, bt\_byte channel)

Connect to a remote device.

void bt\_spp\_disconnect (bt\_spp\_port\_t \*port)

Disconnect from the remote device.

bt\_bool bt\_spp\_deallocate (bt\_spp\_port\_t \*port)

Deallocate serial port.

bt\_bool bt\_spp\_send (bt\_spp\_port\_t \*port, const void \*data, bt\_ulong data\_len, bt\_spp\_send\_callback\_fp callback)

Send data.

bt\_bool bt\_spp\_receive (bt\_spp\_port\_t \*port, void \*buffer, bt\_int buffer\_len, bt\_spp\_receive\_callback\_fp callback)

Receive data.

void bt\_spp\_cancel\_send (bt\_spp\_port\_t \*port)

Cancel send data.

void bt\_spp\_cancel\_receive (bt\_spp\_port\_t \*port)

Cancel receive data.

bt\_int bt\_spp\_get\_frame\_length (bt\_spp\_port\_t \*port)

Get frame length.

• bt\_bdaddr\_t \* bt\_spp\_get\_remote\_address (const bt\_spp\_port\_t \*port)

Get the address of the remote device this device is connected to.

• bt\_byte bt\_spp\_get\_local\_modem\_status (const bt\_spp\_port\_t \*port)

Get local device's TS 07.10 control signals.

• bt\_byte bt\_spp\_set\_local\_modem\_status (bt\_spp\_port\_t \*port, bt\_byte ms)

Set local device's TS 07.10 control signals.

bt\_byte bt\_spp\_get\_remote\_modem\_status (const bt\_spp\_port\_t \*port)

Get remote device's TS 07.10 control signals.

bt\_byte bt\_spp\_set\_dtr (bt\_spp\_port\_t \*port, bt\_bool on)

Set local device's RS-232 DTR signal.

• bt\_byte bt\_spp\_set\_rts (bt\_spp\_port\_t \*port, bt\_bool on)

Set local device's RS-232 RTS signal.

# 3.4.1 Detailed Description

The DotStack SPP API is a simple API for communicating over a Bluetooth link using the Bluetooth Serial Port Profile.

Here are the steps for using this API:

- Call the bt spp init() function.
- Allocate a serial port structure with bt\_spp\_allocate(). One of the parameters to this function is a pointer to
  a callback function. That callback function will be called by the stack whenever the state of the serial port
  changes.

- To connect to a remote device call bt\_spp\_connect(). The stack will notify when the connection is established by calling the state callback function.
- To wait for a connection from a remote device call <a href="bt\_spp\_listen">bt\_spp\_listen</a>(). The stack will notify when the connection is established by calling the state callback function.
- When the port is connected you can send data with bt\_spp\_send() and receive data with bt\_spp\_receive().
- To terminate the connection call bt\_spp\_disconnect().
- When you are finished using the port deallocate it with bt\_spp\_deallocate().

# 3.4.2 Typedef Documentation

3.4.2.1 typedef void(\* bt\_spp\_receive\_callback\_fp) (bt\_spp\_port\_t \*port, bt\_int bytes\_received, void \*param)

Serial port receive callback.

This callback function is called when a receive operation initiated by bt\_spp\_receive() completes.

#### **Parameters**

port	Serial port on which the receive operation completed.
bytes_received	Number of received bytes.
param	Callback parameter that was specified when bt_spp_allocate() was called.

3.4.2.2 typedef void(\* bt\_spp\_send\_callback\_fp) (bt\_spp\_port\_t \*port, bt\_ulong bytes\_sent, bt\_spp\_send\_status\_e status, void \*param)

Serial port send callback.

This callback function is called when a send operation initiated by bt\_spp\_send() completes.

#### **Parameters**

port	Serial port on which the send operation completed.
bytes_sent	Number of bytes sent. This parameter is just a convenience as it always specifies the same
	number of bytes that was passed to bt_spp_send();
status	Completion status. It is one of the values defined in the bt_spp_send_status_e enumeration.
param	Callback parameter that was specified when bt_spp_allocate() was called.

3.4.2.3 typedef void(\* bt\_spp\_state\_callback\_fp) (bt\_spp\_port\_t \*port, bt\_spp\_port\_event\_e evt, void \*param)

Serial port state callback.

This callback function is called whenever the state of a serial port is changed.

#### **Parameters**

port	Serial port which state has changed.
evt	Event describing the nature of state change. It is one of the values defined in the bt_spp_
	port_event_e enumeration.
param	Callback parameter that was specified when bt_spp_allocate() was called.

# 3.4.3 Enumeration Type Documentation

## 3.4.3.1 enum bt\_spp\_port\_event\_e

Serial port event.

Values of this enumeration represent serial port events. Events are reported to the application through a callback function. The callback function is specified when the port is allocated using bt\_spp\_allocate().

#### Enumerator

- **SPP\_PORT\_EVENT\_CONNECTED** Connection with a remote device was successfully established. This event is reported independent of which side (local or remote) initiated the connection.
- **SPP\_PORT\_EVENT\_DISCONNECTED** Connection with the remote device was terminated. This event is reported independent of which side (local or remote) initiated termination of the connection.
- **SPP\_PORT\_EVENT\_CONNECTION\_FAILED** Connection to a remote device failed. This event is reported when an attempt to establish a connection using bt\_spp\_connect() has failed.
- **SPP\_PORT\_EVENT\_SEND\_PROGRESS** Send operation progress. This event is reported periodically during sending data over the serial port connection. It can be used by the application to track send operation progress.
- SPP\_PORT\_EVENT\_REMOTE\_MODEM\_STATUS\_CHANGED Remote modem status. This event is reported when a remote device notifies the local device about the status of its V.24 control signals. The actual value of the signals can be obtained by calling bt\_spp\_get\_remote\_modem\_status(const bt\_spp←\_port\_t\* port)
- **SPP\_PORT\_EVENT\_LOCAL\_MODEM\_STATUS\_CHANGED** Local modem status. This event is reported when the local device has successfully sent the state of its V.24 signals to the remote party.
- **SPP\_PORT\_EVENT\_LOCAL\_MODEM\_STATUS\_CHANGE\_FAILED** Local modem status. This event is reported when the local device failed to send the state of its V.24 signals to the remote party.

3.4.3.2 enum bt\_spp\_port\_state\_e

Serial port state.

Values of this enumeration represent states of the serial port.

# **Enumerator**

**SPP\_PORT\_STATE\_FREE** Used internally to mark a port structure as available for allocation. The application should never encounter a port in this state.

SPP\_PORT\_STATE\_DISCONNECTED Port is not connected.

SPP\_PORT\_STATE\_DISCONNECTING Disconnecting from remote device.

SPP\_PORT\_STATE\_CONNECTING Connecting to a remote device.

SPP\_PORT\_STATE\_CONNECTED Port is connected to a remote device.

3.4.3.3 enum bt\_spp\_send\_status\_e

Send operation status.

#### **Enumerator**

**SPP\_SEND\_STATUS\_SUCCESS** The operation completed successfully.

**SPP\_SEND\_STATUS\_TIMEOUT** The operation timed out.

**SPP\_SEND\_STATUS\_NOT\_ENOUGH\_RESOURCES** There was not enough resources to complete the operation.

SPP\_SEND\_STATUS\_CANCELED The operation was canceled.

SPP\_SEND\_STATUS\_INTERRUPTED The operation was interrupted because the connection closed.

#### 3.4.4 Function Documentation

3.4.4.1 bt\_spp\_port\_t\* bt\_spp\_allocate ( bt\_l2cap\_mgr\_t \* l2cap\_mgr, bt\_spp\_state\_callback\_fp callback, void \* callback\_param )

#### Allocate a serial port.

The returned serial port is initially in the SPP\_PORT\_STATE\_DISCONNECTED state. To establish a connection with a remote device, call bt\_spp\_connect(). To listen for incoming connections from other devices, call bt\_spp\_listen(). The callback parameter must specify a callback function that will be used to notify about serial port events and state changes. When the port is not needed any more it must be deallocated by bt\_spp\_deallocate(). The maximum number of serial ports that can be allocated simultaneously is specified by the SPP\_MAX\_PORTS configuration parameter.

#### **Parameters**

l2cap_mgr	L2CAP manager.
callback	Pointer to a callback function used to notify about serial port events. Cannot be NULL.
callback_param	An arbitrary pointer that is passed as a parameter to the callback function.

#### Returns

A pointer to the ::bt\_spp\_port\_t structure. Returns NULL if the maximum number of ports has been already allocated or the callback parameter is NULL.

3.4.4.2 void bt\_spp\_cancel\_receive ( bt\_spp\_port\_t \* port )

#### Cancel receive data.

If a receive operation is currently in progress this function will cancel it. After calling this function the receive callback specified in <a href="mailto:bt\_spp\_receive">bt\_spp\_receive</a>() will not be called.

If there is no receive operation in progress calling this function has no effect.

#### **Parameters**

port	Serial port.

3.4.4.3 void bt\_spp\_cancel\_send ( bt\_spp\_port\_t \* port )

#### Cancel send data.

If a send operation is currently in progress this function will try to cancel it. When the operation is canceled the send callback function will be called with the SPP\_SEND\_STATUS\_CANCELED status.

If this function is called but the active send operation completes successfuly before the stack can actually cancel it the call back function will still be called with the SPP\_SEND\_STATUS\_CANCELED status.

If there is no send operation in progess calling this function has no effect.

# **Parameters**

port	Serial port.

3.4.4.4 bt\_bool bt\_spp\_connect ( bt\_spp\_port\_t \* port, bt\_bdaddr\_p remote\_addr, bt\_byte channel )

# Connect to a remote device.

This function initiates a connection to a remote device. When the connection is successfully established the port's callback is called with the SPP\_PORT\_EVENT\_CONNECTED event. If connection fails the callback is called with the SPP\_PORT\_EVENT\_CONNECTION\_FAILED event.



#### **Parameters**

	port	Serial port.
remo	te_addr	Bluetooth address of the remote device.
	channel	RFCOMM server channel on which the connection is to be established.

#### Returns

TRUE if successful, FALSE otherwise.

3.4.4.5 bt\_bool bt\_spp\_deallocate ( bt\_spp\_port\_t \* port )

Deallocate serial port.

This function deallocates the specified port structure and other resources associated with it.

The port must be in SPP\_PORT\_STATE\_DISCONNECTED state. Otherwise, the function will fail.

If the function completes successfully the application must not try to access any fields in the structure and must not use it with any other SPP functions. Also, it becomes available for subsequent allocation by bt\_spp\_port\_allocate().

## **Parameters**

port	Serial port structure to deallocate.

#### Returns

TRUE if successful, FALSE otherwise.

3.4.4.6 void bt\_spp\_disconnect ( bt\_spp\_port\_t \* port )

Disconnect from the remote device.

This function initiates the disconnection process. When it is complete the the port's callback is called with the SPP\_PORT\_EVENT\_DISCONNECTED event.

If the port is already in the disconnected state the function does nothing and the callback is not called.

#### **Parameters**

port	Serial port.

3.4.4.7 bt\_int bt\_spp\_get\_frame\_length ( bt\_spp\_port\_t \* port )

Get frame length.

This function returns the RFCOMM frame length used by the RFCOMM protocol. The frame length depends on configuration of DotStack and configuration of the Bluetooth stack running on the remote device. In order to achieve maximum throughput over the serial port connection the application should send and receive data in chunks that are multiple of this frame length.

# Returns

RFCOMM frame length in bytes.

3.4.4.8 bt\_byte bt\_spp\_get\_local\_modem\_status ( const bt\_spp\_port\_t \* port )

Get local device's TS 07.10 control signals.

This function returns current state of the local device's TS 07.10 controls signals. The signals are defined as a mask of the following constants: SPP RS232 DSR SPP RS232 RTS SPP RS232 RI SPP RS232 DCD

#### **Parameters**

port	Serial port.

## Returns

local device's TS 07.10 control signals.

3.4.4.9 bt\_bdaddr\_t\* bt\_spp\_get\_remote\_address ( const bt\_spp\_port\_t \* port )

Get the address of the remote device this device is connected to.

#### **Parameters**

port	Serial port.

## Returns

• A pointer to bt\_bdaddr structure that contains the address of the remote device.

3.4.4.10 bt\_byte bt\_spp\_get\_remote\_modem\_status ( const bt\_spp\_port\_t \* port )

Get remote device's TS 07.10 control signals.

This function returns current state of the remote device's V.24 controls signals. The signals are defined as a mask of the following constants: SPP\_RS232\_DTR SPP\_RS232\_CTS SPP\_RS232\_RI SPP\_RS232\_DCD

# **Parameters**

port	Serial port.
------	--------------

## Returns

remote device's TS 07.10 control signals.

3.4.4.11 void bt\_spp\_init (void )

Initialize the SPP module.

This function initializes all internal variables of the SPP module. It must be called prior to using any other functions in this module.

3.4.4.12 bt\_bool bt\_spp\_listen ( bt\_spp\_port\_t \* port, bt\_byte channel )

Listen for incoming connections.

This function registers the port to accept incoming connections from remote devices on a particular RFCOMM server channel. The specified server channel should be listed in the SDP database. Otherwise, remote devices will not be able to find out which server channel to use.

When a remote device successfully establishes a connection on the specified port the port's callback is called with the SPP\_PORT\_EVENT\_CONNECTED event.

The port must be in SPP\_PORT\_STATE\_DISCONNECTED state. Otherwise, the function will fail.

#### **Parameters**

port	Serial port.
channel	The RFCOMM server channel on which to listen for connections.

#### Returns

TRUE if successful, FALSE otherwise.

3.4.4.13 bt\_bool bt\_spp\_receive ( bt\_spp\_port\_t \* port, void \* buffer, bt\_int buffer\_len, bt\_spp\_receive\_callback\_fp callback )

#### Receive data.

This function receives data from the serial port connection. The caller must provide a buffer and a callback function. Whenever the port receives data they are copied to the provided buffer and the callback function is called. The callback function is passed the length of received data and the same callback parameter that was specified when the port was allocated with <a href="mailto:bt\_spp\_allocate">bt\_spp\_allocate</a>(). This function does not wait until the buffer is filled out completely. Any amount of received data will complete the operation.

The port must be in SPP\_PORT\_STATE\_CONNECTED state. Otherwise, the function will fail. Also, the function will fail if a previously started receive operation is still in progress.

#### **Parameters**

port	Serial port.
data	Pointer to the data to be sent.
data_len	Length of the data.
callback	Send callback function.

#### Returns

TRUE if successful, FALSE otherwise.

3.4.4.14 bt\_bool bt\_spp\_send ( bt\_spp\_port\_t \* port, const void \* data, bt\_ulong data\_len, bt\_spp\_send\_callback\_fp callback )

#### Send data.

This function starts sending data over the serial port connection. Along with the data the caller must provide a callback function that is called when all data has been sent. Also, during execution of this operation the port's state callback function is called periodically with the SPP PORT EVENT SEND PROGRESS event.

The port must be in SPP\_PORT\_STATE\_CONNECTED state. Otherwise, the function will fail. Also, the function will fail if a previously started send operation is still in progress.

The callback function is passed the same callback parameter that was specified when the port was allocated with bt\_spp\_allocate().

# **Parameters**

port	Serial port.
data	Pointer to the data to be sent.
data_len	Length of the data.
callback	Send callback function.

# Returns

TRUE if successful, FALSE otherwise.

3.4.4.15 bt\_byte bt\_spp\_set\_dtr ( bt\_spp\_port\_t \* port, bt\_bool on )

Set local device's RS-232 DTR signal.

Changes the state of local device's RS-232 DTR signal and notifies the remote device of the change. This signal corresponds to the TS 07.10's RFCOMM MODEM STATUS RTC signal.

If there are resources to send a command to the remote device this command will return TRUE.

If the remote party has been successfully notified SPP\_PORT\_EVENT\_LOCAL\_MODEM\_STATUS\_CHANGE ← D event will be reported. If the command could not be send to the remote party for any reason other than lack of resources SPP\_PORT\_EVENT\_LOCAL\_MODEM\_STATUS\_CHANGE\_FAILED will be reported.

If the status cannot be changed because there is no resources to send a command to the remote device this function will return FALSE and no events will be reported.

#### **Parameters**

port	Serial port.
on	Value indicating weather to set or clear the signal.

#### Returns

TRUE if successful, FALSE otherwise.

3.4.4.16 bt\_byte bt\_spp\_set\_local\_modem\_status ( bt\_spp\_port\_t \* port, bt\_byte ms )

Set local device's TS 07.10 control signals.

Changes the state of local device's TS 07.10 control signals and notifies the remote device of the change.

If there are resources to send a command to the remote device this command will return TRUE.

If the remote party has been successfully notified SPP\_PORT\_EVENT\_LOCAL\_MODEM\_STATUS\_CHANGE Devent will be reported. If the command could not be send to the remote party for any reason other than lack of resources SPP\_PORT\_EVENT\_LOCAL\_MODEM\_STATUS\_CHANGE\_FAILED will be reported.

If the status cannot be changed because there is no resources to send a command to the remote device this function will return FALSE and no events will be reported.

#### **Parameters**

port	Serial port.
ms	Signals mask.

# Returns

TRUE if successful, FALSE otherwise.

3.4.4.17 bt\_byte bt\_spp\_set\_rts ( bt\_spp\_port\_t \* port, bt\_bool on )

Set local device's RS-232 RTS signal.

Changes the state of local device's RS-232 DTR signal and notifies the remote device of the change. This signal corresponds to the TS 07.10's RFCOMM MODEM STATUS RTR signal.

If there are resources to send a command to the remote device this command will return TRUE.

If the remote party has been successfully notified SPP\_PORT\_EVENT\_LOCAL\_MODEM\_STATUS\_CHANGE ← D event will be reported. If the command could not be send to the remote party for any reason other than lack of resources SPP\_PORT\_EVENT\_LOCAL\_MODEM\_STATUS\_CHANGE\_FAILED will be reported.

If the status cannot be changed because there is no resources to send a command to the remote device this function will return FALSE and no events will be reported.

# **Parameters**

port	Serial port.
on	Value indicating weather to set or clear the signal.

# Returns

TRUE if successful, FALSE otherwise.

# 3.5 SPP Configuration

This module describes parameters used to configure SPP layer.

#### **Macros**

#define SPP\_MAX\_PORTS
 Maximum number of SPP ports.

# 3.5.1 Detailed Description

This module describes parameters used to configure SPP layer.

dotstack is customized using a configuration file. The configuration file tailors the dotstack to the application being built. It has to have the structure shown below.

```
#include "cdbt/bt/bt_std.h"
// HCI and L2CAP must always be present
// SDP is required only if stack is running in dual mode. This is the default mode.
// To run the stack in single mode (i.e. only BLE is supported) a BT_BLE_SINGLE_MODE symbol
// must be defined:
// #define BT_BLE_SINGLE_MODE
// HCI configuration parameters
#define HCI_MAX_CMD_BUFFERS
#define HCI_MAX_DATA_BUFFERS
#define HCI_MAX_HCI_CONNECTIONS
#define HCI_RX_BUFFER_LEN
#define HCI_TX_BUFFER_LEN
#define HCI L2CAP BUFFER LEN
#define HCI_MAX_CMD_PARAM_LEN
// L2CAP configuration parameters
#define L2CAP_MAX_CMD_BUFFERS
#define L2CAP_MAX_FRAME_BUFFERS
#define L2CAP_MAX_PSMS
#define L2CAP_MAX_CHANNELS
// SDP configuration parameters
#define SDP_MAX_SEARCH_RESULT_LEN
                                       . . .
#define SDP_MAX_ATTRIBUTE_RESULT_LEN ...
// Depending on protocols and profiles used below goes configuration parameters
// for each used module. E.g., to use and configure SPP,
// the following values must be defined:
#define RFCOMM_MAX_SESSIONS
#define RFCOMM_MAX_DLCS
#define RFCOMM_MAX_SERVER_CHANNELS
#define RFCOMM_INFO_LEN
#define RFCOMM_MAX_DATA_BUFFERS
#define RFCOMM_MAX_CMD_BUFFERS
#define RFCOMM_LOCAL_CREDIT
#define SPP_MAX_PORTS
#include "cdbt/bt/bt_oem_config.h"
```

#### 3.5.2 Macro Definition Documentation

# 3.5.2.1 #define SPP\_MAX\_PORTS

Maximum number of SPP ports.

This parameter defines the maximum number of SPP port that can be open between the local and remote devices. If RFCOMM\_ENABLE\_MULTIDEVICE\_CHANNELS is FALSE (default) this value should be equal to RFCOMM  $\leftarrow$  \_MAX\_SERVER\_CHANNELS. If RFCOMM\_ENABLE\_MULTIDEVICE\_CHANNELS is TRUE this value should be between RFCOMM\_MAX\_SERVER\_CHANNELS and RFCOMM\_MAX\_SERVER\_CHANNELS \* RFCOMM\_MA  $\leftarrow$  X\_SESSIONS.

# 3.6 RFCOMM Configuration

This module describes parameters used to configure RFCOMM layer.

### **Macros**

• #define RFCOMM MAX SESSIONS

Maximum number of remote devices a local device can be connected to.

• #define RFCOMM MAX DLCS

Maximum number of DLCs.

#define RFCOMM\_MAX\_SERVER\_CHANNELS

Maximum number of Server channels.

#define RFCOMM INFO LEN

Maximum size of the data portion of a UIH frame.

#define RFCOMM MAX CMD BUFFERS

Maximum number of command buffers.

#define RFCOMM LOCAL CREDIT

The number of receive buffers.

#define RFCOMM\_ENABLE\_MULTIDEVICE\_CHANNELS BT\_FALSE

Enable multi-device server channels.

# 3.6.1 Detailed Description

This module describes parameters used to configure RFCOMM layer.

dotstack is customized using a configuration file. The configuration file tailors the dotstack to the application being built. It has to have the structure shown below.

```
#include "cdbt/bt/bt_std.h"
// HCI and L2CAP must always be present
\ensuremath{//} SDP is required only if stack is running in dual mode. This is the default mode.
// To run the stack in single mode (i.e. only BLE is supported) a BT_BLE_SINGLE_MODE symbol
// must be defined:
// #define BT BLE SINGLE MODE
// HCI configuration parameters
#define HCI_MAX_CMD_BUFFERS
#define HCI_MAX_DATA_BUFFERS
#define HCI_MAX_HCI_CONNECTIONS
#define HCI_RX_BUFFER_LEN
#define HCI_TX_BUFFER_LEN
                                         . . .
#define HCI_L2CAP_BUFFER_LEN
#define HCI_MAX_CMD_PARAM_LEN
// L2CAP configuration parameters
#define L2CAP_MAX_CMD_BUFFERS
#define L2CAP_MAX_FRAME_BUFFERS
#define L2CAP_MAX_PSMS
#define L2CAP_MAX_CHANNELS
// SDP configuration parameters
#define SDP_MAX_SEARCH_RESULT_LEN
#define SDP_MAX_ATTRIBUTE_RESULT_LEN
// Depending on protocols and profiles used below goes configuration parameters
// for each used module. E.g., to use and configure RFCOMM,
\ensuremath{//} the following values must be defined:
#define RFCOMM MAX SESSIONS
#define RFCOMM_MAX_DLCS
                                         . . .
#define RFCOMM_MAX_SERVER_CHANNELS
#define RFCOMM_INFO_LEN
#define RFCOMM_MAX_DATA_BUFFERS
#define RFCOMM_MAX_CMD_BUFFERS
#define RFCOMM LOCAL CREDIT
#include "cdbt/bt/bt_oem_config.h"
```

# 3.6.2 Macro Definition Documentation

# 3.6.2.1 #define RFCOMM\_ENABLE\_MULTIDEVICE\_CHANNELS BT\_FALSE

Enable multi-device server channels.

Normally each server channel can be used only once. I.e. if device A connected to channel 1, device B cannot connect to channel 1 until device A disconnects. With this option it is possible to make channels accept connections from several devices at the same time. I.e., if RFCOMM\_ENABLE\_MULTIDEVICE\_CHANNELS is TRUE both device A and device B can connect to channel 1 at the same time.

### 3.6.2.2 #define RFCOMM\_INFO\_LEN

Maximum size of the data portion of a UIH frame.

This parameter defines the maximum size of the data portion of a UIH frame. If CFC is used the actual length of the data portion will be 1 byte less. This value must be less than or equal to HCI\_L2CAP\_BUFFER\_LEN - RFCOMM\_FRAME\_HEADER\_LEN - L2CAP\_HEADER\_LEN.

### 3.6.2.3 #define RFCOMM\_LOCAL\_CREDIT

The number of receive buffers.

This parameter defines the number of received UIH frames that can be stored on the local device. The flow control mechanism used in RFCOMM ensures that the remote side of the link always knows how many free buffers left on the local device. When the number of free buffers reaches 0, the transmitter stops sending data frames until the receiver frees some buffers. The RFCOMM layer does not actually allocate space for buffers. It uses RFCOMM\_ $\leftarrow$  LOCAL\_CREDIT to keep track of free buffers and report them to the remote side. Actual memory allocation is done in SPP layer.

# 3.6.2.4 #define RFCOMM\_MAX\_CMD\_BUFFERS

Maximum number of command buffers.

This parameter defines the maximum number of commands that can be sent at the same time. It is usually enough to reserve 2 buffers for each DLC excluding control DLC. Therefore, this value can be defined as #define RFCOMM MAX CMD BUFFERS (RFCOMM MAX DLCS - 1) \* 2

### 3.6.2.5 #define RFCOMM MAX DLCS

Maximum number of DLCs.

This parameter defines the maximum number of DLCs on each session. This value should be at least 2 because each session uses one DLC to convey multiplexer control messages. All other DLCs are used to emulate serial ports.

# 3.6.2.6 #define RFCOMM\_MAX\_SERVER\_CHANNELS

Maximum number of Server channels.

This parameter defines the maximum number of server channels exposed by the local device. This value should not exceed RFCOMM\_MAX\_DLCS - 1.

# 3.6.2.7 #define RFCOMM\_MAX\_SESSIONS

Maximum number of remote devices a local device can be connected to.

3.6 RFCOMM Configuration 43 This parameter defines the maximum number of remote devices a local device can have simultaneous connections to. This value should not exceed  $HCI\_MAX\_HCI\_CONNECTIONS$ .

# 3.7 SDP

This module describe functions and data structures used to start the SDP server and perform SDP queries.

### **Modules**

Configuration

This module describes parameters used to configure SDP.

### **Macros**

#define BEGIN\_DE\_SEQUENCE(id, len)

Begin a data element sequence.

#define END\_DE\_SEQUENCE(id) }}

End a data element sequence.

#define INIT\_DE\_SEQUENCE(id) init\_de\_sequence\_##id();

Initialize a data element sequence.

 #define DE\_UINT(value) cur\_de->type = SDP\_DATA\_TYPE\_UINT; cur\_de->data.b = value; cur\_de++; if (++i == max\_len) return;

Declare a 1-byte unsigned integer data element.

• #define DE\_UINT16(value) cur\_de->type = SDP\_DATA\_TYPE\_UINT16; cur\_de->data.ui = value; cur\_← de++; if (++i == max\_len) return;

Declare a 2-byte unsigned integer data element.

• #define DE\_INT(value) cur\_de->type = SDP\_DATA\_TYPE\_INT; cur\_de->data.b = value; cur\_de++; if (++i == max len) return;

Declare a 1-byte signed integer data element.

#define DE\_STRING(value) cur\_de->type = SDP\_DATA\_TYPE\_STRING; cur\_de->data.pstr = value; cur
de++;

Declare a text string data element.

• #define DE\_STRING2(value, len) cur\_de->type = SDP\_DATA\_TYPE\_UINT; cur\_de->data.pstr = value; cur\_de->bytecount = len; cur\_de++; if (++i == max\_len) return;

Declare a text string data element.

• #define DE\_BOOL(value) cur\_de->type = SDP\_DATA\_TYPE\_BOOL; cur\_de->data.b = value; cur\_de++; if (++i == max\_len) return;

Declare a boolean data element.

• #define DE\_UUID16(value) cur\_de->type = SDP\_DATA\_TYPE\_UUID16; cur\_de->data.uuid16 = value; cur\_de++; if (++i == max\_len) return;

Declare a 16-bit UUID data element.

• #define DE\_UUID32(value) cur\_de->type = SDP\_DATA\_TYPE\_UUID32; cur\_de->data.uuid32 = value; cur\_de++; if (++i == max\_len) return;

Declare a 32-bit UUID data element.

• #define DE\_UUID128(value) cur\_de->type = SDP\_DATA\_TYPE\_UUID128; cur\_de->data.uuid128 = (bt\_← uuid t\*)&value; cur\_de++; if (++i == max\_len) return;

Declare a 128-bit UUID data element.

 #define DE\_URL(value) cur\_de->type = SDP\_DATA\_TYPE\_URL; cur\_de->data.purl = value; cur\_de++; if (++i == max\_len) return;

Declare a URL data element.

3.7 SDP 45

### **Functions**

- bt\_bool bt\_sdp\_start (bt\_l2cap\_mgr\_p l2cap\_mgr, const bt\_byte \*sdp\_db, bt\_uint sdp\_db\_len)
   Start SDP server.
- bt\_bool bt\_sdp\_request\_service\_search (bt\_l2cap\_channel\_t \*channel, bt\_sdp\_data\_element\_p pattern, bt\_sdp\_service\_search\_callback\_fp callback, void \*callback\_param)

Search service records.

bt\_bool bt\_sdp\_request\_service\_attribute (bt\_l2cap\_channel\_t \*channel, bt\_sr\_handle\_t sr, bt\_sdp\_data\_
 element\_p pattern, bt\_sdp\_service\_attribute\_callback\_fp callback, void \*callback\_param)

Search attributes.

# 3.7.1 Detailed Description

This module describe functions and data structures used to start the SDP server and perform SDP queries.

# 3.7.2 Macro Definition Documentation

### 3.7.2.1 #define BEGIN DE SEQUENCE( id. len )

### Value:

```
static bt_sdp_data_element_t id[len];
    static bt_sdp_sequence_t seq_##id = { len, (
        bt_sdp_data_element_p)id}; \
    static void init_de_sequence_##id() {
        static bt_bool initialized = FALSE; \
        if (!initialized) {
            bt_int i = 0; \
            bt_int max_len = len; \
            bt_sdp_data_element_t* cur_de = id; \
        initialized = TRUE;
```

Begin a data element sequence.

BEGIN\_DE\_SEQUENCE and END\_DE\_SEQUENCE are used to define a data element sequence which is an array of sdp\_data\_element structures. The array is used a search pattern in bt\_sdp\_request\_service\_search() and bt\_sdp\_request\_service\_attribute(). For example, to find a AVRCP Target the following code can be used:

```
1 const bt_uuid_t AVRCP_AV_REMOTE_CONTROL_CLSID = { 0x5F9B34FB, 0x80000080, 0x00001000,
       SDP_CLSID_AV_REMOTE_CONTROL };
2 const bt_uuid_t AVRCP_AV_REMOTE_CONTROL_TARGET_CLSID = { 0x5F9B34FB, 0x80000080, 0x00001000,
       SDP_CLSID_AV_REMOTE_CONTROL_TARGET };
4 BEGIN_DE_SEQUENCE(avrcp_target_service_search, 2)
5 DE_UUID128(AVRCP_AV_REMOTE_CONTROL_CLSID)
     DE_UUID128 (AVRCP_AV_REMOTE_CONTROL_TARGET_CLSID)
 END_DE_SEQUENCE(avrcp_target_service_search)
9
10 .
11 .
12
13 void findAvrcpTarget(void)
14 {
1.5
      INIT_DE_SEQUENCE(avrcp_target_service_search);
16
      bt_sdp_request_service_search(channel, &seq_avrcp_target_service_search, &callback, NULL);
```

### **Parameters**

id	The data element sequence identifier.
len	The number of elements in the data element sequence.

3.7.2.2 #define DE\_BOOL( value ) cur\_de->type = SDP\_DATA\_TYPE\_BOOL; cur\_de->data.b = value; cur\_de++; if (++i == max\_len) return;

Declare a boolean data element.

This macro adds a boolean data element to a data element sequence. This macro is to be used between BEGIN← \_DE\_SEQUENCE and END\_DE\_SEQUENCE.

#### **Parameters**

value The data element value.	
-------------------------------	--

3.7.2.3 #define DE\_INT( value ) cur\_de->type = SDP\_DATA\_TYPE\_INT; cur\_de->data.b = value; cur\_de++; if (++i == max\_len) return;

Declare a 1-byte signed integer data element.

This macro adds a 1-byte signed integer data element to a data element sequence. This macro is to be used between BEGIN\_DE\_SEQUENCE and END\_DE\_SEQUENCE.

### **Parameters**

value	The data element value.
-------	-------------------------

3.7.2.4 #define DE\_STRING( value ) cur\_de->type = SDP\_DATA\_TYPE\_STRING; cur\_de->data.pstr = value; cur\_de++;

Declare a text string data element.

This macro adds a text string data element to a data element sequence. The length of the generated data element will be the actual length of the string. This macro is to be used between BEGIN\_DE\_SEQUENCE and END\_DE 
\_\_SEQUENCE.

# Parameters

value	The data element value.

3.7.2.5 #define DE\_STRING2( value, len ) cur\_de->type = SDP\_DATA\_TYPE\_UINT; cur\_de->data.pstr = value; cur\_de->bytecount = len; cur\_de++; if (++i == max\_len) return;

Declare a text string data element.

This macro adds a text string data element to a data element sequence. The length of the generated data element will be the value specified by the "len" parameter even if the actual length of the string is not equal to the "len" value. This macro is to be used between BEGIN\_DE\_SEQUENCE and END\_DE\_SEQUENCE.

### **Parameters**

value	The data element value.
len	The length of the data element value.

3.7.2.6 #define DE\_UINT( value ) cur\_de->type = SDP\_DATA\_TYPE\_UINT; cur\_de->data.b = value; cur\_de++; if (++i == max\_len) return;

Declare a 1-byte unsigned integer data element.

3.7 SDP 47

This macro adds a 1-byte unsigned integer data element to a data element sequence. This macro is to be used between BEGIN\_DE\_SEQUENCE and END\_DE\_SEQUENCE.

# **Parameters**

value	The data element value.
-------	-------------------------

3.7.2.7 #define DE\_UINT16( value ) cur\_de->type = SDP\_DATA\_TYPE\_UINT16; cur\_de->data.ui = value; cur\_de++; if (++i == max\_len) return;

Declare a 2-byte unsigned integer data element.

This macro adds a 2-byte unsigned integer data element to a data element sequence. This macro is to be used between BEGIN\_DE\_SEQUENCE and END\_DE\_SEQUENCE.

### **Parameters**

value	The data element value.
-------	-------------------------

3.7.2.8 #define DE\_URL( value ) cur\_de->type = SDP\_DATA\_TYPE\_URL; cur\_de->data.purl = value; cur\_de++; if (++i == max\_len) return;

Declare a URL data element.

This macro adds a URL data element to a data element sequence. This macro is to be used between BEGIN\_D  $\leftarrow$  E\_SEQUENCE and END\_DE\_SEQUENCE.

# **Parameters**

value	The data element value which is a pointer to a string.

3.7.2.9 #define DE\_UUID128( value ) cur\_de->type = SDP\_DATA\_TYPE\_UUID128; cur\_de->data.uuid128 = (bt\_uuid\_t\*)&value; cur\_de++; if (++i == max\_len) return;

Declare a 128-bit UUID data element.

This macro adds a 128-bit UUID data element to a data element sequence. This macro is to be used between BEGIN\_DE\_SEQUENCE and END\_DE\_SEQUENCE.

# **Parameters**

value	The data element value. The value must be a name of a variable of type bt_uuid.
value	The data clement value. The value must be a hame of a variable of type bi_data.

3.7.2.10 #define DE\_UUID16( value ) cur\_de->type = SDP\_DATA\_TYPE\_UUID16; cur\_de->data.uuid16 = value; cur\_de++; if (++i == max\_len) return;

Declare a 16-bit UUID data element.

This macro adds a 16-bit UUID data element to a data element sequence. This macro is to be used between BEGIN\_DE\_SEQUENCE and END\_DE\_SEQUENCE.

# **Parameters**

value	The data element value.

3.7.2.11 #define DE\_UUID32( value ) cur\_de->type = SDP\_DATA\_TYPE\_UUID32; cur\_de->data.uuid32 = value; cur\_de++; if (++i == max\_len) return;

Declare a 32-bit UUID data element.

This macro adds a 32-bit UUID data element to a data element sequence. This macro is to be used between BEGIN\_DE\_SEQUENCE and END\_DE\_SEQUENCE.

### **Parameters**

value	The data element value.

# 3.7.2.12 #define END\_DE\_SEQUENCE( id ) }}

End a data element sequence.

BEGIN\_DE\_SEQUENCE and END\_DE\_SEQUENCE are used to define a data element sequence which is an array of bt sdp data element structures.

### **Parameters**

id	The data element sequence identifier.

# 3.7.2.13 #define INIT\_DE\_SEQUENCE( id ) init\_de\_sequence\_##id();

Initialize a data element sequence.

This macro calls a function defined in BEGIN\_DE\_SEQUENCE which initializes the data element sequence.

# **Parameters**

id	The data element sequence identifier.
----	---------------------------------------

# 3.7.3 Function Documentation

3.7.3.1 bt\_bool bt\_sdp\_request\_service\_attribute ( bt\_l2cap\_channel\_t \* channel, bt\_sr\_handle\_t sr, bt\_sdp\_data\_element\_p pattern, bt\_sdp\_service\_attribute\_callback\_fp callback, void \* callback\_param )

Search attributes.

This function retrieves attribute values from a service record.

# **Parameters**

channel	The L2CAP channel used to communicate to the remote SDP server.
sr	The service record handle specifies the service record from which attribute values are to be
	retrieved.
pattern	The attribute search pattern is a data element sequence where each element in the list is either an attribute ID or a range of attribute IDs. The pattern buffer must be valid for the duration of the search operation, i.e. until callback is called for the first time. To define a data element sequence use the BEGIN_DE_SEQUENCE and END_DE_SEQUENCE macros. These macros will define a variable whose name is the id of the data element sequence passed to the macros prefixed with "seq_". A pointer to this variable can be used as the value for the pattern parameter.

3.7 SDP 49

callback	The callback function that will be called when search has completed.
callback_param	A pointer to arbitrary data to be passed to the callback callback.

# Returns

- TRUE if the function succeeds.
- $\bullet\,$  FALSE otherwise. The callback function is not called in this case.
- 3.7.3.2 bt\_bool bt\_sdp\_request\_service\_search ( bt\_l2cap\_channel\_t \* channel, bt\_sdp\_data\_element\_p pattern, bt\_sdp\_service\_search\_callback\_fp callback, void \* callback\_param )

Search service records.

This function locates service records on a remote SDP server that match the given service search pattern.

### **Parameters**

channel	The L2CAP channel used to communicate to the remote SDP server.
pattern	The service search pattern is a data element sequence where each element in the sequence
	is a UUID. The sequence must contain at least one UUID. The maximum number of UUIDs in
	the sequence is 12. The pattern buffer must be valid for the duration of the search operation,
	i.e. until callback is called. To define a data element sequence use the BEGIN_DE_S↔
	EQUENCE and END_DE_SEQUENCE macros. These macros will define a variable whose
	name is the id of the data element sequence passed to the macros prefixed with "seq_". A
	pointer to this variable can be used as the value for the pattern parameter.
callback	The callback function that will be called when search has completed.
callback_param	A pointer to arbitrary data to be passed to the callback callback.

# Returns

- TRUE if the function succeeds.
- FALSE otherwise. The callback function is not called in this case.
- 3.7.3.3 bt\_bool bt\_sdp\_start ( bt\_l2cap\_mgr\_p l2cap\_mgr, const bt\_byte \* sdp\_db, bt\_uint sdp\_db\_len )

Start SDP server.

This function starts the SDP server.

# **Parameters**

l2cap_mgr	The L2CAP manager on which the SDP server is to be started.
sdp_db	A pointer to the SDP database define with BEGIN_SDP_DB and END_SDP_DB macros.

# Returns

- $\bullet\ \ \mbox{TRUE}$  if the function succeeds.
- FALSE otherwise.

# 3.8 Configuration

This module describes parameters used to configure SDP.

### **Macros**

```
• #define SDP_MAX_PDU_BUFFERS
```

Maximum number of SDP server PDU buffers.

#define SDP\_MAX\_SEARCH\_RESULT\_LEN

Maximum number of service records to find.

#define SDP\_MAX\_ATTRIBUTE\_RESULT\_LEN

Maximum number of attributes to find.

# 3.8.1 Detailed Description

This module describes parameters used to configure SDP.

dotstack is customized using a configuration file. The configuration file tailors the dotstack to the application being built. It has to have the structure shown below.

```
#include "cdbt/bt/bt_std.h"
// HCI and L2CAP must always be present
// SDP is required only if stack is running in dual mode. This is the default mode.
// To run the stack in single mode (i.e. only BLE is supported) a BT_BLE_SINGLE_MODE symbol
// must be defined:
// #define BT_BLE_SINGLE_MODE
// HCI configuration parameters
#define HCI_MAX_CMD_BUFFERS
#define HCI_MAX_DATA_BUFFERS
#define HCI_MAX_HCI_CONNECTIONS
#define HCI_RX_BUFFER_LEN
#define HCI_TX_BUFFER_LEN
#define HCI_MAX_CMD_PARAM_LEN
// L2CAP configuration parameters
#define L2CAP_MAX_CMD_BUFFERS
#define L2CAP_MAX_FRAME_BUFFERS
#define L2CAP_MAX_PSMS
#define L2CAP_MAX_CHANNELS
// SDP configuration parameters
#define SDP_MAX_SEARCH_RESULT_LEN ... #define SDP_MAX_ATTRIBUTE_RESULT_LEN ...
#include "cdbt/bt/bt_oem_config.h"
```

# 3.8.2 Macro Definition Documentation

# 3.8.2.1 #define SDP\_MAX\_ATTRIBUTE\_RESULT\_LEN

Maximum number of attributes to find.

This parameter defines the maximum number of attributes withing a service record the SDP server will return to the client.

# 3.8.2.2 #define SDP\_MAX\_PDU\_BUFFERS

Maximum number of SDP server PDU buffers.

This parameter defines the maximum number of responses the SDP server can send at the same time.

3.8 Configuration 51

0000	#define SDP	BAAV	OFABOLL	DECLUIT	LEN
J.O.Z.J	#deline SUP	IVIAA	SEARGE	RESULI	LEN

Maximum number of service records to find.

This parameter defines the maximum number of service records the SDP server will return to the client.

# 3.9 HID Profile (HIDP)

# **Functions**

bt\_bool bt\_hid\_init (void)

Initialize HID layer.

bt\_bool bt\_hid\_listen (bt\_hid\_session\_p session, bt\_hid\_state\_callback\_fp callback, bt\_hid\_get\_report\_callback fp get report callback, bt hid set report callback fp set report callback)

Listen for incoming connections.

bt\_bool bt\_hid\_connect (bt\_hid\_session\_p session, bt\_bdaddr\_p remote\_addr, bt\_hid\_state\_callback\_
 fp callback, bt\_hid\_get\_report\_callback\_fp get\_report\_callback, bt\_hid\_set\_report\_callback\_fp set\_report
 callback)

Connect to a remote device.

• bt\_bool bt\_hid\_disconnect (bt\_hid\_session\_p session)

Close connection.

void bt\_hid\_unplug (bt\_hid\_session\_p session)

Unplug device.

• bt\_hid\_session\_p bt\_hid\_allocate\_session (bt\_l2cap\_mgr\_p l2cap\_mgr, void \*callback\_param)

Allocate HID session.

void bt\_hid\_free\_session (bt\_hid\_session\_p session)

Release HID session.

• bt\_bool bt\_hid\_add\_report (bt\_hid\_session\_p session, bt\_hid\_report\_t \*report)

Add report definition to local HID device.

bt\_bool bt\_hid\_send\_report (bt\_hid\_session\_p session, bt\_byte channel, bt\_byte report\_type, bt\_byte report\_id, bt\_bool send\_report\_id, bt\_byte\_p data, bt\_int len, bt\_byte tran\_type, bt\_hid\_send\_report\_callback\_fp callback)

Send report.

# 3.9.1 Detailed Description

# 3.9.2 Function Documentation

3.9.2.1 bt\_bool bt\_hid\_add\_report ( bt\_hid\_session\_p session, bt\_hid\_report\_t \* report )

Add report definition to local HID device.

This function add a report definitions to the specified HID session.

# **Parameters**

session	The HID session which a report definition to be added to.
report	The report definition to be added.

# Returns

- TRUE if the function succeeds.
- FALSE otherwise.

3.9.2.2 bt\_hid\_session\_p bt\_hid\_allocate\_session ( bt\_l2cap\_mgr\_p l2cap\_mgr, void \* callback\_param )

Allocate HID session.

This function allocates a new HID session.

3.9 HID Profile (HIDP) 53

### **Parameters**

l2cap_mgr	The L2CAP manager on which the HID session is to be created.
-----------	--------------------------------------------------------------

### Returns

- A pointer to the new HID session structure if the function succeeds.
- NULL otherwise.

3.9.2.3 bt\_bool bt\_hid\_connect ( bt\_hid\_session\_p session, bt\_bdaddr\_p remote\_addr, bt\_hid\_state\_callback\_fp callback, bt\_hid\_get\_report\_callback\_fp get\_report\_callback, bt\_hid\_set\_report\_callback\_fp set\_report\_callback )

Connect to a remote device.

This function establishes a HID connection with a remote device. Changes in the session state are reported through a callback function.

# **Parameters**

session	HID session.	
remote_addr	Address of the remote device.	
callback	The callback function that is called when session state changes.	
get_report_←	The callback function that is called when a remote device requests an INPUT report.	
callback		
set_report_←	The callback function that is called when a remote device sends and OUTPUT report.	
callback		

### Returns

- TRUE if the function succeeds.
- FALSE otherwise. The callback function is not called in this case.

3.9.2.4 bt\_bool bt\_hid\_disconnect ( bt\_hid\_session\_p session )

# Close connection.

This function closes a HID connection with a remote device. Changes in the session state are reported through a callback function set when connection was created by calling bt\_hid\_connect.

# **Parameters**

session	HID session.

3.9.2.5 void bt\_hid\_free\_session ( bt\_hid\_session\_p session )

# Release HID session.

This function deallocated the specified HID session. This function does not disconnect the session. It just frees the memory used by the bt\_hid\_session structure. The session has to be disconnected by calling bt\_hid\_disconnect or bt\_hid\_unplug first.

# **Parameters**

session	The HID session to be deallocated.

3.9.2.6 bt\_bool bt\_hid\_init ( void )

Initialize HID layer.

This function initializes the HID layer of the stack. It must be called prior to any other HID function can be called.

3.9.2.7 bt\_bool bt\_hid\_listen ( bt\_hid\_session\_p session, bt\_hid\_state\_callback\_fp callback, bt\_hid\_get\_report\_callback, bt\_hid\_set\_report\_callback\_fp set\_report\_callback )

Listen for incoming connections.

This function enables incoming connections on the specified HID session. Changes in the session state are reported through a callback function.

# **Parameters**

session	HID session.	
callback	The callback function that is called when session state changes.	
get_report_←	The callback function that is called when a remote device requests an INPUT report.	
callback		
set_report_←	The callback function that is called when a remote device sends and OUTPUT report.	
callback		

# Returns

- TRUE if the function succeeds.
- FALSE otherwise. The callback function is not called in this case.
- 3.9.2.8 bt\_bool bt\_hid\_send\_report ( bt\_hid\_session\_p session, bt\_byte channel, bt\_byte report\_type, bt\_byte report\_id, bt\_bool send\_report\_id, bt\_byte\_p data, bt\_int len, bt\_byte tran\_type, bt\_hid\_send\_report\_callback\_fp callback )

# Send report.

This function sends a report to a remote device

### **Parameters**

session	The HID session.
channel	The type of the channel (CONTROL or INTERRUPT) used to send the report.
report_type	The type of the report (INPUT, OUTPUR, or FEATURE).
report_id	The id of the report.
send_report_id	The flag that specifies weather the report id is included in the data packet.
data	The pointer to the report data.
len	The length of the report data.
tran_type	The type of the transaction (see definition of the HID_TRANTYPE constants).
callback	The callback function that is called when sending the report has been completed.

# Returns

- TRUE if the function succeeds.
- FALSE otherwise. The callback function is not called in this case.
- 3.9.2.9 void bt\_hid\_unplug ( bt\_hid\_session\_p session )

# Unplug device.

This function unplugs a virtually-cabled device Changes in the session state are reported through a callback function set when connection was created by calling bt\_hid\_connect.

3.9 HID Profile (HIDP) 55

# **Parameters**

session	HID session.

# 3.10 Hands Free Profile (HFP)

### **Data Structures**

```
    struct bt_hfp_evt_audio_data_t

     HFP_EVENT_AUDIO_DATA_RECEIVED event parameter.

    struct bt_hfp_evt_slc_connection_state_changed_t

     HFP_EVENT_SLC_CONNECTION_STATE_CHANGED event parameter.

    struct bt_hfp_evt_audio_connection_state_changed_t

     HFP_EVENT_AUDIO_CONNECTION_STATE_CHANGED event parameter.
· struct bt hfp evt mic volume changed t
     HFP_EVENT_MIC_VOLUME_CHANGED event parameter.
struct bt_hfp_evt_spk_volume_changed_t
     HFP_EVENT_SPK_VOLUME_CHANGED event parameter.
· struct bt hfp evt indicator received t
     HFP EVENT INDICATOR RECEIVED event parameter.

    struct bt_hfp_evt_query_operator_completed_t

     HFP_EVENT_QUERY_OPERATOR_COMPLETED event parameter.

    struct bt_hfp_evt_clip_received_t

     HFP_EVENT_CLIP_RECEIVED event parameter.

    struct bt_hfp_evt_call_waiting_t

     HFP_EVENT_CALL_WAITING event parameter.

    struct bt_hfp_evt_command_completed_t

     HFP_EVENT_CMD_COMPLETED event parameter.

    struct bt_hfp_evt_voice_recognition_changed_t

     HFP EVENT VOICE RECOGNITION CHANGED event parameter.

    struct bt_hfp_evt_inband_ring_changed_t

     HFP EVENT INBAND RING CHANGED event parameter.

    struct bt_hfp_evt_dial_request_received_t

     HFP AG EVENT DIAL REQUEST RECEIVED event parameter.
struct bt_hfp_ind
     Stores value of an indicator.

    struct bt_hfp_event_register_t

     Stores value of HF registrations.

    struct bt_hfp_call_t

     Stores information about a call.
struct bt_hfp_audio_packet_t
     Parameter to HFP_EVENT_AUDIO_DATA_RECEIVED event.
```

### **Macros**

- #define bt\_hfp\_allocate\_session\_hf() bt\_hfp\_allocate\_session(HFP\_ROLE\_HF)
   Allocate HFP session.
- $\bullet \ \ \textit{\#define bt\_hfp\_allocate\_session\_ag()} \ \ \textit{bt\_hfp\_allocate\_session(HFP\_ROLE\_AG)}$
- #define bt\_hfp\_hf\_terminate(session) bt\_hfp\_hf\_reject(session)

Terminate the active call.

Allocate HFP session.

#define bt\_hfp\_has\_active\_call(session) (session->call\_status\_mask & HFP\_CALL\_STATUS\_MASK\_AC
 — TIVE)

Determines if there is one or more active calls in the AG.

#define bt\_hfp\_has\_held\_call(session) (session->call\_status\_mask & HFP\_CALL\_STATUS\_MASK\_HELD)

Determines if there is one or more held calls in the AG.

Determines if there is one or more dialing calls in the AG.

#define bt\_hfp\_has\_alerting\_call(session) (session->call\_status\_mask & HFP\_CALL\_STATUS\_MASK\_A
 LERTING)

Determines if there is one or more alerting calls in the AG.

#define bt\_hfp\_has\_outgoing\_call(session) (session->call\_status\_mask & (HFP\_CALL\_STATUS\_MASK\_← ALERTING | HFP\_CALL\_STATUS\_MASK\_DIALING))

Determines if there is one or more outgoing (dialing or alerting) calls in the AG.

 #define bt\_hfp\_has\_incoming\_call(session) (session->call\_status\_mask & HFP\_CALL\_STATUS\_MASK\_← INCOMING)

Determines if there is one or more incoming calls in the AG.

#define bt\_hfp\_has\_waiting\_call(session) (session->call\_status\_mask & HFP\_CALL\_STATUS\_MASK\_W
 — AITING)

Determines if there is one or more waiting calls in the AG.

#define bt\_hfp\_get\_remote\_address(session) (&session->rfcomm\_dlc->psess->pch->hci\_conn->bdaddr
 \_remote)

Get the address of the remote device this device is connected.

• #define bt\_hfp\_ag\_set\_mic\_gain(session, gain) bt\_hfp\_set\_mic\_gain(session, gain)

Set HF microphone gain.

• #define bt\_hfp\_ag\_set\_speaker\_volume(session, volume) bt\_hfp\_set\_speaker\_volume(session, volume) Set HF speaker volume.

# **Typedefs**

• typedef void(\* bt\_hfp\_session\_callback\_pf) (bt\_hfp\_session \*session, bt\_byte evt, void \*evt\_param, void \*callback\_param)

Notify the application of changes in the HFP session.

typedef void(\* bt\_hfp\_find\_remote\_callback\_pf) (bt\_hfp\_session \*session, bt\_byte server\_channel, bt\_bool found)

Notify the application of the result of searching for the Audio Gateway.

• typedef void(\* bt\_hfp\_send\_audio\_callback\_pf) (bt\_hfp\_session \*session, void \*callback\_param)

Notify the application that audio data has been sent to a remote device.

# **Functions**

bt\_bool bt\_hfp\_init (void)

Initialize HFP layer.

bt\_hfp\_session \* bt\_hfp\_allocate\_session (bt\_byte role)

Allocate HFP session.

void bt\_hfp\_free\_session (bt\_hfp\_session \*psess)

Release HFP session.

bt\_bool bt\_hfp\_listen (bt\_hfp\_session \*session, bt\_byte server\_channel, bt\_hfp\_session\_callback\_pf callback, void \*callback\_param)

Listen for incoming connections.

bt\_bool bt\_hfp\_connect (bt\_hfp\_session \*session, bt\_bdaddr\_p pbdaddr\_remote, bt\_hfp\_session\_callback
 —pf callback, void \*callback\_param)

Connect to a remote device.

bt bool bt hfp hf disconnect (bt hfp session \*session)

Close connection.

bt\_bool bt\_hfp\_hf\_query\_operator (bt\_hfp\_session \*session)

Request operator's name from AG.

bt\_bool bt\_hfp\_hf\_answer (bt\_hfp\_session \*session)

Answer the incoming call.

• bt bool bt hfp hf hold (bt hfp session \*session, bt byte cmd, bt byte call index)

Enhanced call control.

bt\_bool bt\_hfp\_hf\_reject (bt\_hfp\_session \*session)

Reject the incoming call.

• bt bool bt hfp hf dial number (bt hfp session \*session, bt char \*number)

Place a call.

bt\_bool bt\_hfp\_hf\_redial (bt\_hfp\_session \*session)

Redial last dialed number.

bt\_bool bt\_hfp\_hf\_enable\_call\_waiting\_notification (bt\_hfp\_session \*session, bt\_bool enable)

Enable/disable call waiting notification.

• bt\_bool bt\_hfp\_set\_mic\_gain (bt\_hfp\_session \*session, bt\_byte gain)

Report microphone gain on the HF to the AG.

• bt\_bool bt\_hfp\_set\_speaker\_volume (bt\_hfp\_session \*session, bt\_byte volume)

Report speaker volume on the HF to the AG.

• bt\_bool bt\_hfp\_hf\_get\_subscriber\_number (bt\_hfp\_session \*session)

Request subscriber number information from the AG.

• bt\_byte bt\_hfp\_get\_indicator\_value (bt\_hfp\_session \*session, bt\_byte indicator\_id)

Get current value of an indicator.

• bt\_bool bt\_hfp\_has\_call\_with\_status (bt\_hfp\_session \*session, bt\_byte status)

Determines if there is one or more calls in the AG with the specified status.

• bt\_bool bt\_hfp\_hf\_refresh\_call\_list (bt\_hfp\_session \*session)

Request a call list from the AG.

bt\_bool bt\_hfp\_hf\_enable\_calling\_line\_identification (bt\_hfp\_session \*session, bt\_bool enable)

Enable/disable calling line identification notification.

bt\_bool bt\_hfp\_hf\_find\_ag (bt\_hfp\_session \*session, bt\_bdaddr\_t \*deviceAddress, bt\_hfp\_find\_remote\_←
 callback pf callback)

Find AG.

• bt bool bt hfp hf enable voice recognition (bt hfp session \*session, bt bool enable)

Enable/disable calling voice control in the AG.

bt\_bool bt\_hfp\_hf\_disable\_nrec (bt\_hfp\_session \*session)

Disable Echo Canceling and Noise Reduction functions in the AG.

bt bool bt hfp connect audio (bt hfp session \*session)

Transfer audio connection from the AG to the HF.

• bt\_bool bt\_hfp\_disconnect\_audio (bt\_hfp\_session \*session)

Transfer audio connection from the HF to the AG.

• bt\_hfp\_call\_t \* bt\_hfp\_ag\_incoming\_call (bt\_hfp\_session \*session, const bt\_char \*number, bt\_int type)

HFP AG incoming call.

• bt hfp call t \* bt hfp ag outgoing call (bt hfp session \*session, const bt char \*number)

HFP AG outgoing call.

void bt\_hfp\_ag\_reject\_outgoing\_call (bt\_hfp\_session \*session)

HFP AG reject outgoing call.

• void bt hfp ag outgoing call alerting (bt hfp session \*session, bt hfp call t \*call)

HFP AG outgoing call alerting.

• void bt\_hfp\_ag\_call\_connected (bt\_hfp\_session \*session, bt\_hfp\_call\_t \*call)

HFP AG call connected.

void bt hfp ag call disconnected (bt hfp session \*session, bt hfp call t \*call)

HFP AG call disconnected.

bt\_bool bt\_hfp\_ag\_set\_service\_state (bt\_hfp\_session \*session, bt\_byte ind\_val)

Set Service State indicator value.

• bt\_bool bt\_hfp\_ag\_set\_signal\_strength (bt\_hfp\_session \*session, bt\_byte ind\_val)

Set Signal Strength indicator value.

• bt\_bool bt\_hfp\_ag\_set\_roam\_state (bt\_hfp\_session \*session, bt\_byte ind\_val)

Set Roaming State indicator value.

• bt\_bool bt\_hfp\_ag\_set\_battery\_level (bt\_hfp\_session \*session, bt\_byte ind\_val)

Set Battery Level indicator value.

• bt\_bool bt\_hfp\_ag\_set\_inband\_ring (bt\_hfp\_session \*session, bt\_byte ind\_val)

Send Inband ring notification to HF device.

• bt\_bool bt\_hfp\_activate\_voice\_recognition (bt\_hfp\_session \*session, bt\_bool activate)

Send voice recognition activation notification to HF device.

• bt\_bool bt\_hfp\_ag\_set\_operator\_name (bt\_hfp\_session \*session, const bt\_byte \*name)

Set Operator name.

bt\_bool bt\_hfp\_ag\_set\_subscriber\_number (bt\_hfp\_session \*session, const bt\_byte \*number, bt\_byte type, bt\_byte service)

Set Subscriber number.

# 3.10.1 Detailed Description

### 3.10.2 Macro Definition Documentation

3.10.2.1 #define bt\_hfp\_ag\_set\_mic\_gain( session, gain ) bt\_hfp\_set\_mic\_gain(session, gain)

Set HF microphone gain.

This function sets microphone gain on the HF. The AG can call this function to set the HF's microphone gain.

# **Parameters**

session	HFP session.
gain	Microphone gain (0 - 15)

### Returns

- TRUE if the function succeeds.
- FALSE otherwise.

3.10.2.2 #define bt\_hfp\_ag\_set\_speaker\_volume( session, volume ) bt\_hfp\_set\_speaker\_volume(session, volume)

Set HF speaker volume.

This function sets speaker volume on the HF. The AG can call this function to set HF's speaker volume.

# **Parameters**

session	HFP session.
gain	Speaker volume (0 - 15)

### Returns

- TRUE if the function succeeds.
- FALSE otherwise.

3.10.2.3 #define bt\_hfp\_allocate\_session\_ag( ) bt\_hfp\_allocate\_session(HFP\_ROLE\_AG)

Allocate HFP session.

This function allocates a new AG HFP session. Currently AG role is not implemented.

# Returns

- · A pointer to the new HFP session structure if the function succeeds.
- NULL otherwise.

3.10.2.4 #define bt\_hfp\_allocate\_session\_hf( ) bt\_hfp\_allocate\_session(HFP\_ROLE\_HF)

Allocate HFP session.

This function allocates a new HF HFP session.

#### Returns

- A pointer to the new HFP session structure if the function succeeds.
- NULL otherwise.

3.10.2.5 #define bt\_hfp\_get\_remote\_address( session ) (&session->rfcomm\_dlc->psess->pch->hci\_conn->bdaddr\_remote)

Get the address of the remote device this device is connected.

#### **Parameters**

session	HFP session.

# Returns

• A pointer to bt bdaddr structure that contains the address of the remote device.

3.10.2.6 #define bt\_hfp\_has\_active\_call( session ) (session->call\_status\_mask & HFP\_CALL\_STATUS\_MASK\_ACTIVE)

Determines if there is one or more active calls in the AG.

This is a define that calls bt\_hfp\_has\_call\_with\_status with HFP\_CALL\_STATUS\_MASK\_ACTIVE as the value for the status parameter.

# **Parameters**

session	HFP session.

### Returns

- TRUE if there are active calls.
- FALSE otherwise.

3.10.2.7 #define bt\_hfp\_has\_alerting\_call( session ) (session->call\_status\_mask & HFP\_CALL\_STATUS\_MASK\_ALERTING)

Determines if there is one or more alerting calls in the AG.

This is a define that calls bt\_hfp\_has\_call\_with\_status with HFP\_CALL\_STATUS\_MASK\_ALERTING as the value for the status parameter.

### **Parameters**

session	HFP session.
---------	--------------

# Returns

- TRUE if there are alerting calls.
- FALSE otherwise.

3.10.2.8 #define bt\_hfp\_has\_dialing\_call( session ) (session->call\_status\_mask & HFP\_CALL\_STATUS\_MASK\_DIALING)

Determines if there is one or more dialing calls in the AG.

This is a define that calls bt\_hfp\_has\_call\_with\_status with HFP\_CALL\_STATUS\_MASK\_DIALING as the value for the status parameter.

### **Parameters**

session	HFP session.

### Returns

- TRUE if there are dialing calls.
- FALSE otherwise.

3.10.2.9 #define bt\_hfp\_has\_held\_call( session ) (session->call\_status\_mask & HFP\_CALL\_STATUS\_MASK\_HELD)

Determines if there is one or more held calls in the AG.

This is a define that calls bt\_hfp\_has\_call\_with\_status with HFP\_CALL\_STATUS\_MASK\_HELD as the value for the status parameter.

### **Parameters**

session	HFP session.

# Returns

- TRUE if there are held calls.
- FALSE otherwise.

3.10.2.10 #define bt\_hfp\_has\_incoming\_call( session ) (session->call\_status\_mask & HFP\_CALL\_STATUS\_MASK\_INCOMING)

Determines if there is one or more incoming calls in the AG.

This is a define that calls bt\_hfp\_has\_call\_with\_status with HFP\_CALL\_STATUS\_MASK\_INCOMING as the value for the status parameter.

# **Parameters**

session
---------

### Returns

- $\bullet\ \ \mbox{TRUE}$  if there are incoming calls.
- FALSE otherwise.

3.10.2.11 #define bt\_hfp\_has\_outgoing\_call( session ) (session->call\_status\_mask & (HFP\_CALL\_STATUS\_MASK\_ALERTI → NG | HFP\_CALL\_STATUS\_MASK\_DIALING))

Determines if there is one or more outgoing (dialing or alerting) calls in the AG.

This is a define that calls bt\_hfp\_has\_call\_with\_status with HFP\_CALL\_STATUS\_MASK\_ALERTING | HFP\_CA ∪ LL STATUS\_MASK\_DIALING as the value for the status parameter.

#### **Parameters**

session	HFP session.

#### Returns

- TRUE if there are outgoing calls.
- FALSE otherwise.

3.10.2.12 #define bt\_hfp\_has\_waiting\_call( session ) (session->call\_status\_mask & HFP\_CALL\_STATUS\_MASK\_WAITING)

Determines if there is one or more waiting calls in the AG.

This is a define that calls bt\_hfp\_has\_call\_with\_status with HFP\_CALL\_STATUS\_MASK\_WAITING as the value for the status parameter.

### **Parameters**

session	HFP session.
---------	--------------

# Returns

- TRUE if there are waiting calls.
- FALSE otherwise.

3.10.2.13 #define bt\_hfp\_hf\_terminate( session ) bt\_hfp\_hf\_reject(session)

Terminate the active call.

If there is an active call in the AG the app may call this function to terminate it. The result of calling this function will be HFP\_EVENT\_CMD\_COMPLETED event.

### **Parameters**

session	HFP session.

# Returns

- TRUE if the function succeeds.
- FALSE otherwise.

# 3.10.3 Typedef Documentation

3.10.3.1 typedef void(\* bt\_hfp\_find\_remote\_callback\_pf) (bt\_hfp\_session \*session, bt\_byte server\_channel, bt\_bool found)

Notify the application of the result of searching for the Audio Gateway.

This function is called by the HFP layer when searching for the Audio Gateway on a remote device has completed.

#### **Parameters**

	session	HFP session.
ĺ	server_channel	The RFCOMM server channel used by the Audio Gateway. The application must pass this
		value to bt_hfp_connect. The value is valid only if the found parameter is TRUE.
ĺ	found	TRUE if Audio Gateway has been found on the remote device. FALSE otherwise.

3.10.3.2 typedef void(\* bt\_hfp\_send\_audio\_callback\_pf) (bt\_hfp\_session \*session, void \*callback\_param)

Notify the application that audio data has been sent to a remote device.

This function is called by the HFP layer when audio data has been sent to a remote device.

## **Parameters**

session	HFP session.
callback_param	The value of the callback_param parameter passed to the call of bt_hfp_send_audio.

3.10.3.3 typedef void(\* bt\_hfp\_session\_callback\_pf) (bt\_hfp\_session \*session, bt\_byte evt, void \*evt\_param, void \*callback\_param)

Notify the application of changes in the HFP session.

This function is called by the HFP layer when various session parameters have changed. It is also called when commands from the remote device or response codes to the commands sent to the the remote device are received.

### **Parameters**

sess	sion	HFP session.
	evt	Specifies the id of the event. The value of this arguments is one of the values defined by H←
		FP_EVENT_ constants. Each event has a parameter that points to an event specific structure
		that clarifies the event. The event parameter is passed through the evt_param parameter.

- HSP\_SESSION\_RFCOMM\_CONNECTION\_STATE\_CHANGED The RFCOMM connection state has
  changed. The application should examine the value of the bt\_hsp\_session::state member to determine
  weather the RFCOMM session has connected or disconnected. The RFCOMM session has connected if the
  HSP SESSION STATE RFCOMM CONNECTED bit is set in the bt\_hsp\_session::state member.
- HSP\_SESSION\_SCO\_CONNECTION\_STATE\_CHANGED The SCO connection state has changed. The application should examine the value of the bt\_hsp\_session::state member to determine weather the RFC← OMM session has connected or disconnected. The RFCOMM session has connected if the HSP\_SESSIO← N STATE SCO CONNECTED bit is set in the bt\_hsp\_session::state member.
- HSP\_SESSION\_CMD\_RESPONSE\_RECEIVED The Headset has received a response to the command it sent to the Audio Gateway. The response is a string pointed to by the what\_param parameter. It can be either HS RES OK or HS RES ERROR.
- HSP\_SESSION\_RING The Audio gateway has sent the RING unsolicited result code. The application must respond to this result code by sending the button press (call bt\_hsp\_hs\_send\_button\_press).
- HSP\_SESSION\_AUDIO\_DATA\_RECEIVED The audio data has been received. The pointer to a buffer is
  passed in the what\_param parameter. The length of the data is passed in the what\_param2 parameter
  (cast to bt\_int).
- HSP\_SESSION\_HEADSET\_CONNECT\_ATTEMPT The attempt to connect to the Audio Gateway has failed.
- HSP\_SESSION\_MIC\_VOLUME\_CHANGED The Audio Gateway has changed the microphone gain. The new value is passed in the what\_param parameter (cast to bt\_byte).
- HSP\_SESSION\_SPK\_VOLUME\_CHANGED The Audio Gateway has changed the speaker volume. The new value is passed in the what\_param parameter (cast to bt byte)

### **Parameters**

evt_param	The event's parameter.
callback_param	The value of the callback_param passed to the call of bt_hfp_listen or bt_hfp_connect.

# 3.10.4 Function Documentation

3.10.4.1 bt\_bool bt\_hfp\_activate\_voice\_recognition ( bt\_hfp\_session \* session, bt\_bool activate )

Send voice recognition activation notification to HF device.

This function should be called by the application to indicate to the HF that the voice recognition in the AG has been activated or deactivated.

This function should be called when Service Level Connection has been established.

### **Parameters**

session	HFP session.
activate	FALSE - voice recognition has been activated in the AG. TRUE - voice recognition has been
	deactivated in the AG.

#### Returns

- TRUE if the function succeeds.
- FALSE if parameter if out of range, or RFCOMM connection is not established.

3.10.4.2 void bt\_hfp\_ag\_call\_connected ( bt\_hfp\_session \* session, bt\_hfp\_call\_t \* call )

# HFP AG call connected.

This function should be called when incoming call has been answered locally, or outgoing call has been answered by remote party. The call is in the connected state. As a result HFP AG sends notification to connected HF device.

# **Parameters**

sessio	HFP session.
ca	HFP call.

3.10.4.3 void bt\_hfp\_ag\_call\_disconnected ( bt\_hfp\_session \* session, bt\_hfp\_call\_t \* call )

# HFP AG call disconnected.

This function should be called when a call has been terminated. It can be local or remote termination of the outgoing or connected call, as well as rejection of the incoming call. As a result HFP AG sends notification to connected HF device.

### **Parameters**

session	HFP session.
call	HFP call.

 $3.10.4.4 \quad \text{bt\_hfp\_call\_t} * \ \text{bt\_hfp\_ag\_incoming\_call} \ ( \ \text{bt\_hfp\_session} * \textit{session}, \ \text{const} \ \text{bt\_char} * \textit{number}, \ \text{bt\_int} \ \textit{type} \ )$ 

# HFP AG incoming call.

This function should be called when AG receives notification from the cellular baseband about an incoming call. As a result HFP AG saves information about the call and send notification to all connected HF devices.

### **Parameters**

session	HFP session.
number	string type phone number of format specified by type
type	
	• 128 - both the type of number and the numbering plan are unknown
	<ul> <li>129 - unknown type of number and ISDN/Telephony numbering plan</li> </ul>
	• 145 - international type of number and ISDN/Telephony numbering plan (contains the character "+")

### Returns

- A pointer to the new HFP call structure if the function succeeds.
- NULL otherwise.

3.10.4.5 bt\_hfp\_call\_t\* bt\_hfp\_ag\_outgoing\_call ( bt\_hfp\_session \* session, const bt\_char \* number )

HFP AG outgoing call.

This function should be called when AG receives notification from the cellular baseband about an outgoing call. As a result HFP AG saves information about the call and send notification to all connected HF devices.

### **Parameters**

session	HFP session.
number	string type phone number of format specified by type

### Returns

- A pointer to the new HFP call structure if the function succeeds.
- NULL otherwise.

3.10.4.6 void bt\_hfp\_ag\_outgoing\_call\_alerting ( bt\_hfp\_session \* session, bt\_hfp\_call\_t \* call )

HFP AG outgoing call alerting.

This function should be called when AG receives notification that the outgoing call is alerting remote party. As a result HFP AG sends notification to all connected HF devices.

### **Parameters**

session	HFP session.
call	HFP call.

3.10.4.7 void bt\_hfp\_ag\_reject\_outgoing\_call ( bt\_hfp\_session \* session )

HFP AG reject outgoing call.

This function should be called by AG to abort a request from HF to dial a number stored in memory or re-dial last number dialed.

### **Parameters**

session	HFP session.
---------	--------------

3.10.4.8 bt\_bool bt\_hfp\_ag\_set\_battery\_level ( bt\_hfp\_session \* session, bt\_byte ind\_val )

Set Battery Level indicator value.

This function should be called by the application during session initialization and when battery level changes.

#### **Parameters**

session	HFP session.
ind_val	Current battery level. Valid values are between 0 for low battery and 5 when battery is full.

### Returns

- TRUE if the function succeeds.
- FALSE if parameter if out of range.

3.10.4.9 bt\_bool bt\_hfp\_ag\_set\_inband\_ring ( bt\_hfp\_session \* session, bt\_byte ind\_val )

Send Inband ring notification to HF device.

This function should be called by the application to indicate to the HF that the in-band ring tone setting has been locally changed.

This function should be called when Service Level Connection has been established.

### **Parameters**

session	HFP session.
ind_val	Indicator value. 0 - the AG provides no in-band ring tone. 1 - the AG provides an in-band ring
	tone.

### Returns

- TRUE if the function succeeds.
- ${\tt FALSE}$  if parameter if out of range, or RFCOMM connection is not established.

3.10.4.10 bt\_bool bt\_hfp\_ag\_set\_operator\_name ( bt\_hfp\_session \* session, const bt\_byte \* name )

Set Operator name.

This function should be called by the application during session initialization.

# **Parameters**

session	HFP session.
name	Operator's name.

### Returns

- TRUE if the function succeeds.
- FALSE if parameter if out of range.

 $3.10.4.11 \quad bt\_bool\ bt\_hfp\_ag\_set\_roam\_state\ (\ bt\_hfp\_session * \textit{session},\ bt\_byte\ \textit{ind\_val}\ )$ 

Set Roaming State indicator value.

This function should be called by the application during session initialization and every time roaming state changes.

### **Parameters**

session	HFP session.
ind_val	Current roaming state. Valid values are 1 when the phone is roaming, 0 when phone is in
	home network.

### Returns

- TRUE if the function succeeds.
- FALSE if parameter if out of range.

3.10.4.12 bt\_bool bt\_hfp\_ag\_set\_service\_state ( bt\_hfp\_session \* session, bt\_byte ind\_val )

Set Service State indicator value.

This function should be called by the application during session initialization and every time service state goes up or down.

### **Parameters**

session	HFP session.
ind_val	Current service state. Valid values are 1 for service present, 0 otherwise.

### Returns

- TRUE if the function succeeds.
- FALSE if parameter if out of range.

3.10.4.13 bt\_bool bt\_hfp\_ag\_set\_signal\_strength ( bt\_hfp\_session \* session, bt\_byte ind\_val )

Set Signal Strength indicator value.

This function should be called by the application during session initialization and every time signal strength value changes.

# **Parameters**

session	HFP session.
ind_val	Current signal strength. Valid values are between 0 for low signal and 5 for maximum signal.

### Returns

- TRUE if the function succeeds.
- FALSE if parameter if out of range.

3.10.4.14 bt\_bool bt\_hfp\_ag\_set\_subscriber\_number ( bt\_hfp\_session \* session, const bt\_byte \* number, bt\_byte type, bt\_byte service )

Set Subscriber number.

This function should be called by the application during session initialization.

### **Parameters**

session	HFP session.
number	string type phone number of format specified by type
type	
	128 - both the type of number and the numbering plan are unknown
	• 129 - unknown type of number and ISDN/Telephony numbering plan
	• 145 - international type of number and ISDN/Telephony numbering plan (contains the character "+")
service	Indicates which service this phone number relates to. Shall be either 4 (voice) or 5 (fax).

# Returns

- TRUE if the function succeeds.
- FALSE if parameter if out of range.

3.10.4.15 bt\_hfp\_session\* bt\_hfp\_allocate\_session( bt\_byte role )

Allocate HFP session.

This function allocates a new HFP session.

# **Parameters**

role	HFP_ROLE_HF or HFP_ROLE_AG.
------	-----------------------------

# Returns

- A pointer to the new HFP session structure if the function succeeds.
- NULL otherwise.

3.10.4.16 bt\_bool bt\_hfp\_connect ( bt\_hfp\_session \* session, bt\_bdaddr\_p pbdaddr\_remote, bt\_hfp\_session\_callback\_pf callback, void \* callback\_param )

Connect to a remote device.

This function establishes a HFP connection with a remote device which is running in Audio Gateway mode. Changes in the session state are reported through a callback function.

# **Parameters**

session	HFP session.
remote_addr	Address of the remote device.
callback	A callback function that is called when session state changes.
callback_param	An arbitrary data pointer that will be passed to the callback function specified by the
	callback parameter.

### Returns

- TRUE if the function succeeds.
- FALSE otherwise. The callback function is not called in this case.

3.10.4.17 bt\_bool bt\_hfp\_connect\_audio ( bt\_hfp\_session \* session )

Transfer audio connection from the AG to the HF.

This function establishes a SCO connection to the AG. As a result the AG will route its audio path to the HF. The result of calling this function will be HFP\_EVENT\_AUDIO\_CONNECTION\_STATE\_CHANGED event.

# **Parameters**

	LIED :
session	HFP session.

### Returns

- TRUE if the function succeeds.
- FALSE otherwise. the HFP\_EVENT\_AUDIO\_CONNECTION\_STATE\_CHANGED event is not reported in this case.

3.10.4.18 bt\_bool bt\_hfp\_disconnect\_audio ( bt\_hfp\_session \* session )

Transfer audio connection from the HF to the AG.

This function terminates the SCO connection to the AG. As a result the AG will route its audio path to itself. The result of calling this function will be HFP\_EVENT\_AUDIO\_CONNECTION\_STATE\_CHANGED event.

### **Parameters**

session	HFP session.
---------	--------------

# Returns

- TRUE if the function succeeds.
- FALSE otherwise. the HFP\_EVENT\_AUDIO\_CONNECTION\_STATE\_CHANGED event is not reported in this case.

3.10.4.19 void bt\_hfp\_free\_session ( bt\_hfp\_session \* psess )

Release HFP session.

This function deallocated the specified HFP session. This function does not disconnect the session. It just frees the memory used by the bt\_hfp\_session structure. The session has to be disconnected by a remote device or by calling bt hfp hf disconnect.

# Parameters

session	The HFP session to be deallocated.

3.10.4.20 bt\_byte bt\_hfp\_get\_indicator\_value ( bt\_hfp\_session \* session, bt\_byte indicator\_id )

Get current value of an indicator.

This function returns a value stored on the local device.

**Parameters** 

session	HFP session.
indicator_id	the ID if the indicator

### Returns

• Current indicator's value.

3.10.4.21 bt\_bool bt\_hfp\_has\_call\_with\_status ( bt\_hfp\_session \* session, bt\_byte status )

Determines if there is one or more calls in the AG with the specified status.

This function does not send any commands to the AG. It uses a list of calls stored on the local device.

# **Parameters**

session	HFP session.
status	a bit mask of HFP_CALL_STATUS_ constants that defines statuses of calls to be checked.

# **Returns**

- TRUE if there are calls with specified statuses.
- FALSE otherwise.

3.10.4.22 bt\_bool bt\_hfp\_hf\_answer ( bt\_hfp\_session \* session )

Answer the incoming call.

If there is an incoming call in the AG the app may call this function to accept it. The result of calling this function will be HFP\_EVENT\_CMD\_COMPLETED event followed by HFP\_EVENT\_CALL\_STARTED event.

### **Parameters**

session	HSP session.

### Returns

- TRUE if the function succeeds.
- FALSE otherwise.

3.10.4.23 bt\_bool bt\_hfp\_hf\_dial\_number ( bt\_hfp\_session \* session, bt\_char \* number )

### Place a call.

Intiate an outgoing call. The result of calling this function will be HFP\_EVENT\_CMD\_COMPLETED event.

# **Parameters**

session	HFP session.
number	Phone number to dial.

### Returns

- TRUE if the function succeeds.
- FALSE otherwise.

3.10.4.24 bt\_bool bt\_hfp\_hf\_disable\_nrec ( bt\_hfp\_session \* session )

Disable Echo Canceling and Noise Reduction functions in the AG.

This function should be called by the application to disable any Echo Canceling and Noise Reduction functions embedded in the AG. The result of calling this function will be HFP\_EVENT\_NREC\_DISABLED event.

This function should be called when Service Level Connection has been established and before any Audio Connection between the HF and the AG is established.

#### **Parameters**

session	HFP session.

#### Returns

- TRUE if the function succeeds.
- FALSE if parameter if out of range, or RFCOMM connection is not established.

3.10.4.25 bt\_bool bt\_hfp\_hf\_disconnect ( bt\_hfp\_session \* session )

Close connection.

This function closes a HFP connection with a remote device. Changes in the session state are reported through a callback function set when connection was created by calling bt\_hfp\_connect or bt\_hfp\_listen.

# **Parameters**

session	HSP session.
---------	--------------

### Returns

- TRUE if the function succeeds.
- FALSE otherwise.

3.10.4.26 bt\_bool bt\_hfp\_hf\_enable\_call\_waiting\_notification ( bt\_hfp\_session \* session, bt\_bool enable )

Enable/disable call waiting notification.

Request AG to enable or disable call waiting notification. If call waiting notification is enabled, the AG will notify the HF whenever an incoming call is waiting during an ongoing call. The result of calling this function will be HFP\_EV ENT\_CMD\_COMPLETED event. There is usually no need to call this function directly as the call waiting notification is enabled during SLC setup.

### **Parameters**

session	HFP session.
enable	TRUE - to enable call waiting notification, FALSE - to disable it.

### Returns

- TRUE if the function succeeds.
- FALSE otherwise.

3.10.4.27 bt\_bool bt\_hfp\_hf\_enable\_calling\_line\_identification ( bt\_hfp\_session \* session, bt\_bool enable )

Enable/disable calling line identification notification.

Request AG to enable or disable calling line identification notification. If calling line identification notification is enabled and the calling number is available from the network, the AG will notify the HF whenever there is an incoming call. The application will receive HFP\_EVENT\_CLIP\_RECEIVED event. The result of calling this function will be HFP\_EVENT\_CMD\_COMPLETED event. There is usually no need to call this function directly as the calling line identification notification is enabled during SLC setup.

#### **Parameters**

session	HFP session.
enable	TRUE - to enable calling line identification notification, FALSE - to disable it.

# Returns

- · TRUE if the function succeeds.
- FALSE otherwise.

3.10.4.28 bt\_bool bt\_hfp\_hf\_enable\_voice\_recognition ( bt\_hfp\_session \* session, bt\_bool enable )

Enable/disable calling voice control in the AG.

Request AG to enable or disable voice control. The result of calling this function will be HFP\_EVENT\_CMD\_CO

MPLETED event.

#### **Parameters**

session	HFP session.
enable	TRUE - to enable voice control, FALSE - to disable it.

### Returns

- TRUE if the function succeeds.
- FALSE otherwise.

3.10.4.29 bt\_bool bt\_hfp\_hf\_find\_ag ( bt\_hfp\_session \* session, bt\_bdaddr\_t \* deviceAddress, bt\_hfp\_find\_remote\_callback\_pf callback )

# Find AG.

This function connects to a remote device and tries to find the AG on it. If the remote device implements HFP AG this function returns (via callback) the RFCOMM channel which should be used to connect to it.

### **Parameters**

	session	HFP session.
	deviceAddress	Remote device's address.
Ī	callback	A pointer to a function which is called when AG search is complete.

# Returns

- TRUE if the function succeeds.
- FALSE otherwise. the callback is not called in this case.

3.10.4.30 bt\_bool bt\_hfp\_hf\_get\_subscriber\_number ( bt\_hfp\_session \* session )

Request subscriber number information from the AG.

This function sends a command to the AG which causes it to send back subscriber number information. The result of calling this function will be HFP\_EVENT\_CMD\_COMPLETED event followed by HFP\_EVENT\_SUBSCRIBER 

NUMBER RECEIVED event. The caller can retrieve the information from the bt hfp session::subscriber number.

### **Parameters**

session	HFP session.
---------	--------------

# Returns

- TRUE if the function succeeds.
- FALSE otherwise. The callback function is not called in this case.

3.10.4.31 bt\_bool bt\_hfp\_hf\_hold ( bt\_hfp\_session \* session, bt\_byte cmd, bt\_byte call\_index )

Enhanced call control.

If there is an incoming call in the AG the app may call this function to reject it. The result of calling this function will be HFP\_EVENT\_CMD\_COMPLETED event.

### **Parameters**

session	HFP session.
cmd	Command Id.
call	index Call index.

#### Returns

- TRUE if the function succeeds.
- FALSE otherwise.

3.10.4.32 bt\_bool bt\_hfp\_hf\_query\_operator ( bt\_hfp\_session \* session )

Request operator's name from AG.

This function requests operator's name from the AG. The result of this call is reported to the app in the form of the HFP EVENT QUERY OPERATOR COMPLETED event.

There is usually no need to call this function directly as the operator's name is requested and reported to the app during SLC setup.

# **Parameters**

session	HSP session.
---------	--------------

# Returns

- TRUE if the function succeeds.
- FALSE otherwise.

3.10.4.33 bt\_bool bt\_hfp\_hf\_redial ( bt\_hfp\_session \* session )

Redial last dialed number.

Request AG to dial last dialed number. The result of calling this function will be HFP\_EVENT\_CMD\_COMPLETED event.

### **Parameters**

session	HFP session.
---------	--------------

# Returns

- TRUE if the function succeeds.
- FALSE otherwise.

# 3.10.4.34 bt\_bool bt\_hfp\_hf\_refresh\_call\_list ( bt\_hfp\_session \* session )

Request a call list from the AG.

This function send a command to the AG which causes it to send back a list of all calls it currently has. The result of calling this function will be HFP\_EVENT\_CMD\_COMPLETED event followed by HFP\_EVENT\_CALL\_LIST\_R ← ECEIVED event.

### **Parameters**

session   HFP sess	ion.

# **Returns**

- TRUE if the function succeeds.
- FALSE otherwise.

# 3.10.4.35 bt\_bool bt\_hfp\_hf\_reject ( bt\_hfp\_session \* session )

Reject the incoming call.

If there is an incoming call in the AG the app may call this function to reject it. The result of calling this function will be HFP EVENT CMD COMPLETED event.

# **Parameters**

session
---------

### Returns

- TRUE if the function succeeds.
- FALSE otherwise.

# 3.10.4.36 bt bool bt hfp\_init ( void )

Initialize HFP layer.

This function initializes the HFP layer of the stack. It must be called prior to any other HFP function can be called.

3.10.4.37 bt\_bool bt\_hfp\_listen ( bt\_hfp\_session \* session, bt\_byte server\_channel, bt\_hfp\_session\_callback\_pf callback, void \* callback\_param )

Listen for incoming connections.

This function enables incoming connections on the specified HFP session. Changes in the session state are reported through a callback function.

### **Parameters**

session	HFP session.
server_channel	An RFCOMM server channel on which the RFCOMM session specified in the call to bt_hfp←
	_allocate_session will be listening.
callback	A callback function that is called when session state changes or data (command or response)
	is received from the remote party.
callback_param	An arbitrary data pointer that will be passed to the callback function specified by the
	callback parameter.

### Returns

- TRUE if the function succeeds.
- FALSE otherwise. The callback function is not called in this case.

3.10.4.38 bt\_bool bt\_hfp\_set\_mic\_gain ( bt\_hfp\_session \* session, bt\_byte gain )

Report microphone gain on the HF to the AG.

This function reports microphone gain on the HF to the AF. This function is called by the HF upon SLC setup to notify the AG about the current level of the microphone gain on the HF. In case physical mechanisms (buttons, dials etc.) means are implemented on the HF to control the volume levels, the HF calls this function to inform the AG of any changes in the microphone gain.

#### **Parameters**

session	HFP session.
gain	Microphone gain (0 - 15)

# Returns

- TRUE if the function succeeds.
- FALSE otherwise.

3.10.4.39 bt\_bool bt\_hfp\_set\_speaker\_volume ( bt\_hfp\_session \* session, bt\_byte volume )

Report speaker volume on the HF to the AG.

This function reports speaker volume on the HF to the AG. This function is called by the HF upon SLC setup to notify the AG about the current level of the speaker volume on the HF. In case physical mechanisms (buttons, dials etc.) means are implemented on the HS to control the volume levels, the HF calls this function to inform the AG of any changes in the speaker volume.

# **Parameters**

session	HFP session.
gain	Speaker volume (0 - 15)

### Returns

- TRUE if the function succeeds.
- FALSE otherwise.

# 3.11 Audio/Video Distribution Protocol (AVDP)

AVDTP is the transport protocol for audio and/or video distribution connections and streaming of audio or video media over the Bluetooth air interface.

### Modules

Configuration

This module describes parameters used to configure AVDTP layer.

### **Data Structures**

· struct bt media packet t

Media packet buffer.

struct bt\_avdtp\_sep\_capabilities\_t

SEP capabilities.

struct bt\_avdtp\_sep\_t

SEP description.

• struct bt\_avdtp\_evt\_ctrl\_channel\_connected\_t

Parameter to AVDTP\_EVT\_CTRL\_CHANNEL\_CONNECTED event.

struct bt\_avdtp\_evt\_ctrl\_channel\_disconnected\_t

Parameter to AVDTP\_EVT\_CTRL\_CHANNEL\_DISCONNECTED event.

• struct bt\_avdtp\_evt\_discover\_completed\_t

Parameter to AVDTP\_EVT\_DISCOVER\_COMPLETED event.

struct bt\_avdtp\_evt\_sep\_info\_received\_t

Parameter to AVDTP\_EVT\_SEP\_INFO\_RECEIVED event.

• struct bt\_avdtp\_evt\_get\_sep\_capabilities\_completed\_t

 ${\it Parameter\ to\ AVDTP\_EVT\_GET\_SEP\_CAPABILITIES\_COMPLETED\ event.}$ 

• struct bt\_avdtp\_evt\_sep\_capabilities\_received\_t

Parameter to AVDTP\_EVT\_SEP\_CAPABILITIES\_RECEIVED and AVDTP\_EVT\_STREAM\_CONFIGURATION\_R ← ECEIVED events.

struct bt\_avdtp\_evt\_set\_stream\_configuration\_completed\_t

Parameter to AVDTP\_EVT\_SET\_STREAM\_CONFIGURATION\_COMPLETED event.

struct bt\_avdtp\_evt\_get\_stream\_configuration\_completed\_t

Parameter to AVDTP EVT SET STREAM CONFIGURATION COMPLETED event.

• struct bt\_avdtp\_evt\_stream\_reconfigure\_completed\_t

Parameter to AVDTP\_EVT\_STREAM\_RECONFIGURE\_COMPLETED event.

struct bt\_avdtp\_evt\_open\_stream\_completed\_t

Parameter to AVDTP\_EVT\_OPEN\_STREAM\_COMPLETED event.

struct bt\_avdtp\_evt\_start\_stream\_completed\_t

Parameter to AVDTP\_EVT\_START\_STREAM\_COMPLETED event.

struct bt\_avdtp\_evt\_close\_stream\_completed\_t

Parameter to AVDTP\_EVT\_CLOSE\_STREAM\_COMPLETED event.

struct bt\_avdtp\_evt\_suspend\_stream\_completed\_t

Parameter to AVDTP\_EVT\_SUSPEND\_STREAM\_COMPLETED event.

struct bt\_avdtp\_evt\_stream\_security\_control\_completed\_t

Parameter to AVDTP\_EVT\_STREAM\_SECURITY\_CONTROL\_COMPLETED event.

• struct bt\_avdtp\_evt\_set\_stream\_configuration\_requested\_t

Parameter to AVDTP\_EVT\_SET\_STREAM\_CONFIGURATION\_REQUESTED event.

struct bt\_avdtp\_evt\_reconfigure\_stream\_requested\_t

```
Parameter to AVDTP EVT RECONFIGURE STREAM REQUESTED event.

    struct bt_avdtp_evt_open_stream_requested_t

     Parameter to AVDTP_EVT_OPEN_STREAM_REQUESTED event.
· struct bt avdtp evt start stream requested t
     Parameter to AVDTP_EVT_START_STREAM_REQUESTED event.

    struct bt avdtp evt suspend stream requested t

     Parameter to AVDTP EVT SUSPEND STREAM REQUESTED event.
· struct bt avdtp evt close stream requested t
     Parameter to AVDTP EVT CLOSE STREAM REQUESTED event.
struct bt_avdtp_evt_abort_stream_requested_t
     Parameter to AVDTP EVT ABORT STREAM REQUESTED event.
· struct bt_avdtp_evt_delay_report_completed_t
     Parameter to AVDTP_EVT_DELAYREPORT_COMPLETED event.
• struct bt_avdtp_evt_stream_configured_t
     Parameter to AVDTP_EVT_STREAM_CONFIGURED event.

    struct bt_avdtp_evt_stream_reconfigured_t

     Parameter to AVDTP_EVT_STREAM_RECONFIGURED event.

    struct bt_avdtp_evt_stream_opened_t

     Parameter to AVDTP_EVT_STREAM_OPENED event.
· struct bt avdtp evt stream started t
     Parameter to AVDTP EVT STREAM STARTED event.

    struct bt_avdtp_evt_stream_suspended_t

     Parameter to AVDTP EVT STREAM SUSPENDED event.

    struct bt_avdtp_evt_stream_closed_t

     Parameter to AVDTP_EVT_STREAM_CLOSED event.
· struct bt_avdtp_evt_stream_aborted_t
     Parameter to AVDTP_EVT_STREAM_ABORTED event.

    struct bt_avdtp_evt_media_packet_received_t

     Parameter to AVDTP_EVT_MEDIA_PACKET_RECEIVED event.
· struct bt avdtp evt media packet sent t
     Parameter to AVDTP EVT MEDIA PACKET SENT event.
· struct bt avdtp evt media packet send failed t
     Parameter to AVDTP EVT MEDIA PACKET SEND FAILED event.
· union bt avdtp event t
     Parameter to an application callback.

    struct bt_avdtp_transport_channel_t

     Transport channel description.
· struct bt_avdtp_transport_session_t
     Transport session description.

    struct bt_avdtp_stream_t

     Stream description.
· struct bt avdtp codec t
     Codec handler description.

    struct bt_avdtp_codec_op_parse_config_t

     Parameter to AVDTP CODEC OPCODE PARSE CONFIG operation.

    struct bt_avdtp_codec_op_serialize_config_t

     Parameter to AVDTP CODEC OPCODE SERIALIZE CONFIG operation.
· union bt_avdtp_codec_op_param_t
     Parameter to a codec handler.
struct bt_avdtp_mgr_t
     AVDTP manager.
```

### **Macros**

Connect to a remote device.

• #define bt\_avdtp\_connect\_ex(mgr, remote\_addr, acl\_config) \_bt\_avdtp\_open\_control\_channel\_ex(mgr, remote\_addr, acl\_config)

Connect to a remote device.

# **Typedefs**

typedef bt\_byte(\* bt\_avdtp\_codec\_handler\_fp) (struct \_bt\_avdtp\_codec\_t \*codec, bt\_byte opcode, bt\_
 avdtp\_codec\_op\_param\_t \*op\_param, struct \_bt\_avdtp\_mgr\_t \*mgr)

Codec handler.

typedef void(\* bt\_avdtp\_mgr\_callback\_fp) (struct \_bt\_avdtp\_mgr\_t \*mgr, bt\_byte evt, bt\_avdtp\_event\_
 t \*evt\_param, void \*callback\_param)

AVDTP application callback.

### **Functions**

bt\_avdtp\_mgr\_t \* bt\_avdtp\_get\_mgr (void)

Return a pointer to an instance of the AVDTP manager.

void bt avdtp init (void)

Initialize the AVDTP layer.

bt\_bool bt\_avdtp\_start (bt\_avdtp\_mgr\_t \*mgr)

Start the AVDTP layer.

void bt\_avdtp\_register\_callback (bt\_avdtp\_mgr\_t \*mgr, bt\_avdtp\_mgr\_callback\_fp callback, void \*callback 
 \_param)

Register a AVDTP application callback.

- bt\_byte bt\_avdtp\_register\_sep (bt\_avdtp\_mgr\_t \*mgr, bt\_byte type, const bt\_avdtp\_sep\_capabilities\_t \*caps)

  \*\*Register a SEP with the local AVDTP manager.
- bt\_avdtp\_sep\_t \* bt\_avdtp\_get\_sep (bt\_avdtp\_mgr\_t \*mgr, bt\_byte sep\_id)

Get a SEP info by its ID.

• bt\_bool bt\_avdtp\_disconnect (bt\_avdtp\_mgr\_t \*mgr, bt\_bdaddr\_t \*remote\_addr)

Disconnect from a remote device.

• bt\_bool bt\_avdtp\_discover (bt\_avdtp\_mgr\_t \*mgr, bt\_bdaddr\_t \*remote\_addr)

Discover SEPs on a remote device.

- bt\_bool bt\_avdtp\_get\_capabilities (bt\_avdtp\_mgr\_t \*mgr, bt\_bdaddr\_t \*remote\_addr, bt\_byte seid\_acp)

  Get remote SEP capabilities.
- bt\_bool bt\_avdtp\_get\_all\_capabilities (bt\_avdtp\_mgr\_t \*mgr, bt\_bdaddr\_t \*remote\_addr, bt\_byte seid\_acp)

  Get remote SEP capabilities.
- bt\_byte bt\_avdtp\_create\_stream (bt\_avdtp\_mgr\_t \*mgr)

Create a stream.

• bt\_bool bt\_avdtp\_destroy\_stream (bt\_avdtp\_mgr\_t \*mgr, bt\_byte strm\_handle)

Destroy a stream.

• bt\_bool bt\_avdtp\_listen (bt\_avdtp\_mgr\_t \*mgr, bt\_byte strm\_handle, bt\_byte sep\_id)

Listen for incoming connections.

• void bt\_avdtp\_cancel\_listen (bt\_avdtp\_mgr\_t \*mgr, bt\_byte strm\_handle, bt\_byte sep\_id)

Cancel listening for incoming connections.

 bt\_bool bt\_avdtp\_set\_configuration (bt\_avdtp\_mgr\_t \*mgr, bt\_byte strm\_handle, bt\_bdaddr\_t \*remote\_addr, bt\_byte seid\_int, bt\_byte seid\_acp, const bt\_avdtp\_sep\_capabilities\_t \*caps)

Set stream configuration.

bt\_bool bt\_avdtp\_get\_configuration (bt\_avdtp\_mgr\_t \*mgr, bt\_byte strm\_handle)

Get stream configuration.

bt\_bool bt\_avdtp\_reconfigure\_stream (bt\_avdtp\_mgr\_t \*mgr, bt\_byte strm\_handle, bt\_avdtp\_sep\_
 capabilities\_t \*caps)

Reconfigure stream.

• bt byte bt avdtp get stream state (bt avdtp mgr t \*mgr, bt byte strm handle)

Get local stream state.

• bt\_byte bt\_avdtp\_get\_stream\_local\_sep\_id (bt\_avdtp\_mgr\_t \*mgr, bt\_byte strm\_handle)

Get stream's local SEP ID.

• bt\_byte bt\_avdtp\_get\_stream\_remote\_sep\_id (bt\_avdtp\_mgr\_t \*mgr, bt\_byte strm\_handle)

Get stream's remote SEP ID.

bt\_bdaddr\_t \* bt\_avdtp\_get\_stream\_remote\_address (bt\_avdtp\_mgr\_t \*mgr, bt\_byte strm\_handle)

Get stream's remote BT address.

• bt\_byte bt\_avdtp\_get\_stream\_codec\_type (bt\_avdtp\_mgr\_t \*mgr, bt\_byte strm\_handle)

Get the type of the codec currently used with the stream.

void \* bt avdtp get stream codec config (bt avdtp mgr t \*mgr, bt byte strm handle)

Get the configuration of the codec currently used with the stream.

• bt\_avdtp\_sep\_capabilities\_t \* bt\_avdtp\_get\_stream\_config (bt\_avdtp\_mgr\_t \*mgr, bt\_byte strm\_handle)

Get stream's configuration.

• bt\_bool bt\_avdtp\_open\_stream (bt\_avdtp\_mgr\_t \*mgr, bt\_byte strm\_handle)

Open a stream

• bt\_bool bt\_avdtp\_start\_stream (bt\_avdtp\_mgr\_t \*mgr, bt\_byte strm\_handle)

Start a stream.

• bt\_bool bt\_avdtp\_close\_stream (bt\_avdtp\_mgr\_t \*mgr, bt\_byte strm\_handle)

Close a stream.

• bt\_bool bt\_avdtp\_suspend\_stream (bt\_avdtp\_mgr\_t \*mgr, bt\_byte strm\_handle)

Suspend a stream.

• bt\_bool bt\_avdtp\_abort\_stream (bt\_avdtp\_mgr\_t \*mgr, bt\_byte strm\_handle)

Suspend a stream.

bt\_bool bt\_avdtp\_security\_control (bt\_avdtp\_mgr\_t \*mgr, bt\_byte strm\_handle, bt\_byte \*sc\_data, bt\_byte sc\_data\_len)

Exchange content protection control data.

• bt bool bt avdtp report delay (bt avdtp mgr t \*mgr, bt byte strm handle, bt uint delay)

Report delay value of a Sink to a Source.

bt\_bool bt\_avdtp\_register\_codec (bt\_avdtp\_mgr\_t \*mgr, bt\_avdtp\_codec\_t \*codec)

Register a codec.

bt\_bool bt\_avdtp\_unregister\_codec (bt\_avdtp\_mgr\_t \*mgr, bt\_byte codec\_type)

Unregister a codec.

• bt avdtp codec t \* bt avdtp find codec (bt avdtp mgr t \*mgr, bt byte codec type)

Find a codec.

bt\_bool bt\_avdtp\_add\_media\_rx\_buffer (bt\_avdtp\_mgr\_t \*mgr, bt\_byte strm\_handle, bt\_media\_packet\_

 t \*buffer)

Add a media packet buffer to a receive queue.

bt\_bool bt\_avdtp\_remove\_media\_rx\_buffer (bt\_avdtp\_mgr\_t \*mgr, bt\_byte strm\_handle, bt\_media\_packet ← t \*buffer)

Remove a media packet buffer from a receive queue.

bt\_bool bt\_avdtp\_add\_media\_tx\_buffer (bt\_avdtp\_mgr\_t \*mgr, bt\_byte strm\_handle, bt\_media\_packet\_
 t \*buffer)

Add a media packet buffer to a send queue.

bt\_bool bt\_avdtp\_remove\_media\_tx\_buffer (bt\_avdtp\_mgr\_t \*mgr, bt\_byte strm\_handle, bt\_media\_packet ←
 t \*buffer)

Remove a media packet buffer from a send queue.

bt\_hci\_conn\_state\_t \* bt\_avdtp\_get\_hci\_connection (bt\_avdtp\_mgr\_t \*mgr, bt\_byte strm\_handle)
 Get HCl connection for a stream.

### **Events**

The following is a list of events AVDTP layer generates and can report to the upper layer when it completes executing an operation initiated by either local or remote device.

• #define AVDTP EVT CTRL CHANNEL CONNECTED 1

This event is generated when a control channel between two AVDTP entities has been established.

#define AVDTP\_EVT\_CTRL\_CHANNEL\_DISCONNECTED 2

This event is generated when a control channel between two AVDTP entities has been terminated.

#define AVDTP EVT CTRL CONNECTION FAILED 3

This event is generated when a local device failed to create a control channel between two AVDTP entities.

#define AVDTP\_EVT\_DISCOVER\_COMPLETED 4

This event is generated when a local device received a response (either positive or negative) to a "discover" request.

#define AVDTP EVT GET SEP CAPABILITIES COMPLETED 5

This event is generated when a local device received a response (either positive or negative) to a "get SEP capabilities" request.

• #define AVDTP\_EVT\_SET\_STREAM\_CONFIGURATION\_COMPLETED 6

This event is generated when a local device received a response (either positive or negative) to a "set stream configuration" request.

#define AVDTP\_EVT\_GET\_STREAM\_CONFIGURATION\_COMPLETED 7

This event is generated when a local device received a response (either positive or negative) to a "get stream configuration" request.

#define AVDTP\_EVT\_STREAM\_RECONFIGURE\_COMPLETED 8

This event is generated when a local device received a response (either positive or negative) to a "change stream configuration" request.

#define AVDTP\_EVT\_OPEN\_STREAM\_COMPLETED 9

This event is generated when a local device received a response (either positive or negative) to a "open stream" request.

• #define AVDTP EVT START STREAM COMPLETED 10

This event is generated when a local device received a response (either positive or negative) to a "start stream" request

#define AVDTP EVT CLOSE STREAM COMPLETED 11

This event is generated when a local device received a response (either positive or negative) to a "close stream" request

#define AVDTP\_EVT\_SUSPEND\_STREAM\_COMPLETED 12

This event is generated when a local device received a response (either positive or negative) to a "suspend stream" request.

#define AVDTP\_EVT\_STREAM\_SECURITY\_CONTROL\_COMPLETED 13

This event is generated when a local device received a response (either positive or negative) to a "exchange content protection control data" request.

• #define AVDTP EVT ABORT STREAM COMPLETED 14

This event is generated when a local device received a response (either positive or negative) to a "abort stream" request

#define AVDTP EVT SEP INFO RECEIVED 15

This event is generated for each SEP contained in a positive response to a "discover" request.

• #define AVDTP EVT SEP CAPABILITIES RECEIVED 16

This event is generated when a local device received a positive response to a "get SEP capabilities" request.

• #define AVDTP EVT STREAM CONFIGURATION RECEIVED 17

This event is generated when a local device received a positive response to a "get stream configuration" request.

• #define AVDTP EVT DELAYREPORT COMPLETED 18

This event is generated when a local device received a response (either positive or negative) to a "delay report" request.

#define AVDTP EVT SET STREAM CONFIGURATION REQUESTED 50

This event is generated when a local device received "set stream configuration" request.

• #define AVDTP EVT OPEN STREAM REQUESTED 51

This event is generated when a local device received "open stream" request.

#define AVDTP EVT START STREAM REQUESTED 52

This event is generated when a local device received "start stream" request.

#define AVDTP EVT CLOSE STREAM REQUESTED 53

This event is generated when a local device received "close stream" request.

#define AVDTP\_EVT\_SUSPEND\_STREAM\_REQUESTED 54

This event is generated when a local device received "suspend stream" request.

• #define AVDTP\_EVT\_ABORT\_STREAM\_REQUESTED 55

This event is generated when a local device received "abort stream" request.

#define AVDTP\_EVT\_RECONFIGURE\_STREAM\_REQUESTED 56

This event is generated when a local device received "change stream configuration" request.

#define AVDTP EVT MEDIA PACKET RECEIVED 57

This event is generated when a local device received a media packet.

#define AVDTP EVT STREAM CONFIGURED 58

This event is generated when a local device has successfully configured a stream.

#define AVDTP EVT STREAM RECONFIGURED 59

This event is generated when a local device has successfully reconfigured a stream.

#define AVDTP\_EVT\_STREAM\_OPENED 60

This event is generated when a local device has successfully opened a stream.

• #define AVDTP\_EVT\_STREAM\_STARTED 61

This event is generated when a local device has successfully started a stream.

• #define AVDTP\_EVT\_STREAM\_CLOSED 62

This event is generated when a local device has successfully closed a stream.

• #define AVDTP\_EVT\_STREAM\_SUSPENDED 63

This event is generated when a local device has successfully suspended a stream.

#define AVDTP\_EVT\_STREAM\_ABORTED 64

This event is generated when a local device has successfully aborted a stream.

#define AVDTP\_EVT\_MEDIA\_PACKET\_SENT 65

This event is generated when a local device sent a media packet.

#define AVDTP\_EVT\_MEDIA\_PACKET\_SEND\_FAILED 66

This event is generated when a local device failed to send a media packet.

## **Stream States**

#define AVDTP\_STREAM\_STATE\_IDLE 0

The stream is idle.

#define AVDTP STREAM OPENING TRANSPORT CHANNELS 1

The stream is opening transport channels.

#define AVDTP\_STREAM\_CLOSING\_TRANSPORT\_CHANNELS 2

The stream is closing transport channels.

#define AVDTP STREAM STATE CONFIGURED 3

The stream has been configured.

• #define AVDTP\_STREAM\_STATE\_OPEN 4

The stream has been opened.

• #define AVDTP\_STREAM\_STATE\_STREAMING 5

The stream has been started.

#define AVDTP\_STREAM\_STATE\_CLOSING 6

The stream is closing.

#define AVDTP\_STREAM\_STATE\_ABORTING 7

The stream is aborting.

### **SEP Type**

The following is a list of SEP types.

• #define AVDTP SEP TYPE SOURCE 0

Source (usually a device like a phone, desktop or laptop).

• #define AVDTP SEP TYPE SINK 1

Sink (usually a device like a headphones or BMW).

## **Service Categories**

The following is a list of service categories a SEP supports.

Note

dotstack supports only AVDTP\_SEP\_SERVICE\_CAPABILITY\_MEDIA\_TRANSPORT and AVDTP\_SEP\_S ← ERVICE CAPABILITY MEDIA CODEC.

These constants define values that are transferred OTA. They are not use in API. Constants for initializing bt\_avdtp\_sep\_capabilities\_t structure that is used to define a SEP's capabilities are defined with AVDTP\_S← EP CAPABILITY FLAG ... constants.

• #define AVDTP SEP SERVICE CAPABILITY MEDIA TRANSPORT 1

Media. A SEP is capable of transferring media (audio, video or both) packets.

• #define AVDTP SEP SERVICE CAPABILITY REPORTING 2

Reporting. A SEP is capable of transferring reporting packets.

• #define AVDTP\_SEP\_SERVICE\_CAPABILITY\_RECOVERY 3

Recovery. A SEP is capable of transferring recovery packets.

#define AVDTP SEP SERVICE CAPABILITY CONTENT PROTECTION 4

Content Prortection. A SEP is capable of transferring content protection packets.

• #define AVDTP SEP SERVICE CAPABILITY HEADER COMPRESSION 5

Header Compression. A SEP can use header compression for transferring Media or Recovery packets.

• #define AVDTP SEP SERVICE CAPABILITY MULTIPLEXING 6

Multiplexing. Multiple transport sessions, belonging to the same or to a different stream, can share a common transport (L2CAP) channel.

#define AVDTP SEP SERVICE CAPABILITY MEDIA CODEC 7

Media Codec. Defines which codec a SEP supports. A SEP can support only one codec.

#define AVDTP SEP SERVICE CAPABILITY DELAY REPORTING 8

Delay Reporting.

## **Service Categories Flags**

The following is a list of constants that can be used to initialize bt\_avdtp\_sep\_capabilities\_t::categories. A combination of these constants defines service capabilities exposed by a SEP to a remote party.

Note

dotstack supports only AVDTP\_SEP\_CAPABILITY\_FLAG\_MEDIA\_TRANSPORT and AVDTP\_SEP\_CAP ← ABILITY\_FLAG\_MEDIA\_CODEC. All other service capabilities are ignored. They are defined here only for completeness.

#define AVDTP SEP CAPABILITY FLAG MEDIA TRANSPORT 0x01

Media. A SEP is capable of transferring media (audio, video or both) packets.

#define AVDTP\_SEP\_CAPABILITY\_FLAG\_REPORTING 0x02

Reporting. A SEP is capable of transferring reporting packets.

#define AVDTP SEP CAPABILITY FLAG RECOVERY 0x04

Recovery. A SEP is capable of transferring recovery packets.

#define AVDTP SEP CAPABILITY FLAG CONTENT PROTECTION 0x08

Content Prortection. A SEP is capable of transferring content protection packets.

#define AVDTP\_SEP\_CAPABILITY\_FLAG\_HEADER\_COMPRESSION 0x10

Header Compression. A SEP can use header compression for transferring Media or Recovery packets.

• #define AVDTP SEP CAPABILITY FLAG MULTIPLEXING 0x20

Multiplexing. Multiple transport sessions, belonging to the same or to a different stream, can share a common transport (L2CAP) channel.

#define AVDTP\_SEP\_CAPABILITY\_FLAG\_MEDIA\_CODEC 0x40

Media Codec. Defines which codec a SEP supports. A SEP can support only one codec.

• #define AVDTP\_SEP\_CAPABILITY\_FLAG\_DELAY\_REPORTING 0x80

Delat reporitng.

## **Transport Session Types**

The following is a list of transport sessions a SEP supports.

Note

The only transport session cyrrently supported by dotstack is AVDTP\_TRANSPORT\_SESSION\_TYPE\_M← EDIA.

- #define AVDTP\_TRANSPORT\_SESSION\_TYPE\_MEDIA 0
   Media (audio or video).
- #define AVDTP\_TRANSPORT\_SESSION\_TYPE\_REPORTING 1

Reporting (currently not supported).

• #define AVDTP\_TRANSPORT\_SESSION\_TYPE\_RECOVERY 2

Recovery (currently not supported).

## **Codec Types**

The following is a list of codecs a SEP supports.

- #define AVDTP\_CODEC\_TYPE\_SBC 0x00
   SBC (mandatory to support in A2DP profile).
- #define AVDTP\_CODEC\_TYPE\_MPEG1\_2\_AUDIO 0x01

MPEG-1,2 (optional).

• #define AVDTP\_CODEC\_TYPE\_MPEG2\_4\_AAC 0x02

MPEG-2,4 AAC (optional, used in Apple's products).

• #define AVDTP CODEC TYPE ATRAC 0x04

ATRAC (proprietary codec owned by Sony Corporation).

#define AVDTP\_CODEC\_TYPE\_NON\_A2DP 0xFF

Vendor specific.

# **Media Types**

The following is a list of media types a SEP can support.

#define AVDTP\_MEDIA\_TYPE\_AUDIO 0

Audio.

#define AVDTP\_MEDIA\_TYPE\_VIDEO 1

Video.

• #define AVDTP MEDIA TYPE MULTIMEDIA 2

Both Audio & Video.

## **Codec Handler Operations**

#define AVDTP CODEC OPCODE PARSE CONFIG 0

Parse codec configuration.

#define AVDTP CODEC OPCODE SERIALIZE CONFIG 1

Serialize codec configuration.

### **Error Codes**

• #define AVDTP ERROR SUCCESS 0

The operation completed with no errors.

#define AVDTP\_ERROR\_BAD\_HEADER\_FORMAT 0x01

The request packet header format is invalid.

• #define AVDTP\_ERROR\_BAD\_LENGTH 0x11

The request packet length is not match the assumed length.

#define AVDTP\_ERROR\_BAD\_ACP\_SEID 0x12

The requested command indicates an invalid ACP SEP ID (not addressable)

• #define AVDTP\_ERROR\_SEP\_IN\_USE 0x13

The SEP is in use.

#define AVDTP\_ERROR\_SEP\_NOT\_IN\_USE 0x14

The SEP is not in use.

#define AVDTP\_ERROR\_BAD\_SERV\_CATEGORY 0x17

The value of Service Category in the request packet is not defined in AVDTP.

#define AVDTP\_ERROR\_BAD\_PAYLOAD\_FORMAT 0x18

The requested command has an incorrect payload format.

• #define AVDTP ERROR NOT SUPPORTED COMMAND 0x19

The requested command is not supported by the device.

#define AVDTP\_ERROR\_INVALID\_CAPABILITIES 0x1a

The reconfigure command is an attempt to reconfigure a transport service capabilities of the SEP. Reconfigure is only permitted for application service capabilities.

#define AVDTP ERROR BAD RECOVERY TYPE 0x22

The requested Recovery Type is not defined in AVDTP.

#define AVDTP\_ERROR\_BAD\_MEDIA\_TRANSPORT\_FORMAT 0x23

The format of Media Transport Capability is not correct.

#define AVDTP ERROR BAD RECOVERY FORMAT 0x25

The format of Recovery Service Capability is not correct.

#define AVDTP\_ERROR\_BAD\_ROHC\_FORMAT 0x26

The format of Header Compression Service Capability is not correct.

#define AVDTP\_ERROR\_BAD\_CP\_FORMAT 0x27

The format of Content Protection Service Capability is not correct.

#define AVDTP ERROR BAD MULTIPLEXING FORMAT 0x28

The format of Multiplexing Service Capability is not correct.

#define AVDTP ERROR UNSUPPORTED CONFIGURAION 0x29

Configuration not supported.

#define AVDTP\_ERROR\_BAD\_STATE 0x31

The stream is in state that does not permit executing commands.

#define AVDTP\_ERROR\_FAILED\_TO\_CONNECT\_TRANSPORT 0x40

An attempt to esctablish a transport channel has failed.

#define AVDTP ERROR FAILED TO CONNECT CONTROL 0x41

An attempt to esctablish a control channel has failed.

### 3.11.1 Detailed Description

AVDTP is the transport protocol for audio and/or video distribution connections and streaming of audio or video media over the Bluetooth air interface.

### 3.11.2 Macro Definition Documentation

### 3.11.2.1 #define AVDTP\_EVT\_STREAM\_ABORTED 64

This event is generated when a local device has successfully aborted a stream.

This event follows the AVDTP\_EVT\_ABORT\_STREAM\_REQUESTED if the upper layer has accepted it. This event is not generated if stream abortion was initiated by the local device.

### 3.11.2.2 #define AVDTP\_EVT\_STREAM\_CLOSED 62

This event is generated when a local device has successfully closed a stream.

This event follows the AVDTP\_EVT\_CLOSE\_STREAM\_REQUESTED if the upper layer has accepted it. This event is not generated if stream closing was initiated by the local device.

### 3.11.2.3 #define AVDTP\_EVT\_STREAM\_CONFIGURED 58

This event is generated when a local device has successfully configured a stream.

This event follows the AVDTP\_EVT\_SET\_STREAM\_CONFIGURATION\_REQUESTED if the upper layer has accepted it. This event is not generated if stream configuration was initiated by the local device.

### 3.11.2.4 #define AVDTP\_EVT\_STREAM\_OPENED 60

This event is generated when a local device has successfully opened a stream.

This event follows the AVDTP\_EVT\_OPEN\_STREAM\_REQUESTED if the upper layer has accepted it. This event is not generated if stream opening was initiated by the local device.

### 3.11.2.5 #define AVDTP\_EVT\_STREAM\_RECONFIGURED 59

This event is generated when a local device has successfully reconfigured a stream.

This event follows the AVDTP\_EVT\_RECONFIGURE\_STREAM\_REQUESTED if the upper layer has accepted it. This event is not generated if stream reconfiguration was initiated by the local device.

### 3.11.2.6 #define AVDTP\_EVT\_STREAM\_STARTED 61

This event is generated when a local device has successfully started a stream.

This event follows the AVDTP\_EVT\_START\_STREAM\_REQUESTED if the upper layer has accepted it. This event is not generated if stream starting was initiated by the local device.

### 3.11.2.7 #define AVDTP\_EVT\_STREAM\_SUSPENDED 63

This event is generated when a local device has successfully suspended a stream.

This event follows the AVDTP\_EVT\_SUSPEND\_STREAM\_REQUESTED if the upper layer has accepted it. This event is not generated if stream suspension was initiated by the local device.

### 3.11.2.8 #define AVDTP\_STREAM\_STATE\_ABORTING 7

The stream is aborting.

This means that all transport channels associated with the stream are being closed. After they have been closed the stream goes to AVDTP\_STREAM\_STATE\_IDLE state.

## 3.11.2.9 #define AVDTP\_STREAM\_STATE\_CLOSING 6

The stream is closing.

This means that all transport channels associated with the stream are being closed. After they have been closed the stream goes to AVDTP\_STREAM\_STATE\_IDLE state.

### 3.11.2.10 #define AVDTP\_STREAM\_STATE\_IDLE 0

The stream is idle.

This can mean two things. The stream specified by strm\_handle does not exist or the stream is closed.

## 3.11.2.11 #define AVDTP\_STREAM\_STATE\_STREAMING 5

The stream has been started.

Depending on the local SEP type (source or sink) it means that the stream can send or receive media packets.

3.11.2.12 #define bt\_avdtp\_connect( *mgr, remote\_addr* ) \_bt\_avdtp\_open\_control\_channel\_ex(mgr, remote\_addr, HCI\_CONFIG\_ENABLE\_AUTHENTICATION | HCI\_CONFIG\_ENABLE\_ENCRYPTION)

Connect to a remote device.

This function opens a control channel connection to a remote device specified by the remote\_addr. If connection cannot be initiated for some reason, for example, there is not enough resources, it returns FALSE and not events are generated. Otherwise the result of an attempt to connect to the remote device is reported via the AVDTP callback. The events generated will either be AVDTP\_EVT\_CTRL\_CHANNEL\_CONNECTED or AVDTP\_EVT\_CTRL\_CH← ANNEL CONNECTION FAILED.

### **Parameters**

mgr	AVDTP manager.
remote_addr	The address of a remote device.

#### Returns

- · TRUE if connection establishment has been started.
- FALSE otherwise.

3.11.2.13 #define bt\_avdtp\_connect\_ex( mgr, remote\_addr, acl\_config ) \_bt\_avdtp\_open\_control\_channel\_ex(mgr, remote\_addr, acl\_config)

#### Connect to a remote device.

This function opens a control channel connection to a remote device specified by the remote\_addr. If connection cannot be initiated for some reason, for example, there is not enough resources, it returns FALSE and not events are generated. Otherwise the result of an attempt to connect to the remote device is reported via the AVDTP callback. The events generated will either be AVDTP\_EVT\_CTRL\_CHANNEL\_CONNECTED or AVDTP\_EVT\_CTRL\_CH← ANNEL\_CONNECTION\_FAILED.

### **Parameters**

mgr	AVDTP manager.
remote_addr	The address of a remote device.
acl_config	ACL link configuration. This can be a combination of the following values:
	HCI_CONFIG_ENABLE_AUTHENTICATION
	HCI_CONFIG_ENABLE_ENCRYPTION
	HCI_CONFIG_BECOME_MASTER

### Returns

- TRUE if connection establishment has been started.
- FALSE otherwise.

## 3.11.3 Typedef Documentation

3.11.3.1 typedef bt\_byte(\* bt\_avdtp\_codec\_handler\_fp) (struct\_bt\_avdtp\_codec\_t \*codec, bt\_byte opcode, bt\_avdtp\_codec\_op\_param\_t \*op\_param, struct\_bt\_avdtp\_mgr\_t \*mgr)

## Codec handler.

AVDTP in theory can support any type of codec. Each codec uses its own format for exchanging capabilities and configuration information. In order to make out implementation do not care about these formats we use a simple way of telling AVDTP how to parse and serialize codec's configuration. The consumer of AVDTP (e.g. A2DP) for each codec it wishes to support has to register a callback function (one per codec type). That callback has to perform two function. The first one is to read the configuration received from the remote device and store it in a structure defined by the consumer. The second one is to serialize the data from a structure to a format (in case of standard A2DP codecs the format is defined in A2DP specification, vendor specific codecs can define their own formats) suitable for sending as a part of a AVDTP request.

This typedef defines the interace for the callback function.

### **Parameters**

codec	A pointer to a structure that describes a codec.
opcode	<ul> <li>The code of an operation to execute. The opcode can be one of the following values:</li> <li>AVDTP_CODEC_OPCODE_PARSE_CONFIG: The handler has to parse configuration received from the remote device and store it in a structure defined by the consumer.</li> <li>AVDTP_CODEC_OPCODE_SERIALIZE_CONFIG: The handler has to serialize the data from a consumer defined structure to a format suitable for sending as a part of a AVDTP request.</li> </ul>
op_param	A pointer to the operation's specific parameters. Parameters are passed as a pointer to the bt_avdtp_codec_op_param_t union. Which member of the union points to a valid structure depends on the value of the opcode:  • bt_avdtp_codec_op_param_t::parse: Valid if opcode == AVDTP_CODEC_OPCO  DE_PARSE_CONFIG.  • bt_avdtp_codec_op_param_t::serialize: Valid if opcode == AVDTP_CODEC_OPC  ODE_SERIALIZE_CONFIG.
mgr	AVDTP manager.

3.11.3.2 typedef void(\* bt\_avdtp\_mgr\_callback\_fp) (struct \_bt\_avdtp\_mgr\_t \*mgr, bt\_byte evt, bt\_avdtp\_event\_t \*evt\_param, void \*callback\_param)

# AVDTP application callback.

In order to be notified of various events a consumer of the AVDTP layer has to register a callback function. The stack will call that function whenever a new event has been generated.

## **Parameters**

mgr	AVDTP manager.
evt	AVDTP event. The event can be one of the following values:
	AVDTP_EVT_CTRL_CHANNEL_CONNECTED: Control channel connected.
	AVDTP_EVT_CTRL_CHANNEL_DISCONNECTED: Control channel disconnected.
	AVDTP_EVT_CTRL_CONNECTION_FAILED: Control channel connection failed (generated only if control connection has been initiated by the local device).
	AVDTP_EVT_DISCOVER_COMPLETED: Local device completed discovering remote SEPs.
	<ul> <li>AVDTP_EVT_GET_SEP_CAPABILITIES_COMPLETED: Local device received a response to Get SEP capabilities operation.</li> </ul>
	AVDTP_EVT_SET_STREAM_CONFIGURATION_COMPLETED: Local device received a response to Set stream configuration operation.
	AVDTP_EVT_GET_STREAM_CONFIGURATION_COMPLETED: Local device received a response to Get stream configuration operation.
	AVDTP_EVT_STREAM_RECONFIGURE_COMPLETED: Local device received a response to Reconfigure stream operation.
	AVDTP_EVT_OPEN_STREAM_COMPLETED: Local device received a response to Open stream operation.
	AVDTP_EVT_START_STREAM_COMPLETED: Local device received a response to Start stream operation.
	<ul> <li>AVDTP_EVT_CLOSE_STREAM_COMPLETED: Local device received a response to Close stream operation.</li> </ul>
	<ul> <li>AVDTP_EVT_SUSPEND_STREAM_COMPLETED: Local device received a response to Suspend stream operation.</li> </ul>
	<ul> <li>AVDTP_EVT_STREAM_SECURITY_CONTROL_COMPLETED: Local device received a response to Stream security control operation.</li> </ul>
	AVDTP_EVT_ABORT_STREAM_COMPLETED: Local device received a response to Abort stream operation.
	AVDTP_EVT_SEP_INFO_RECEIVED: SEP information received.
	AVDTP_EVT_SEP_CAPABILITIES_RECEIVED: SEP capabilities received.
	<ul> <li>AVDTP_EVT_STREAM_CONFIGURATION_RECEIVED: Stream configuration received.</li> </ul>
	<ul> <li>AVDTP_EVT_SET_STREAM_CONFIGURATION_REQUESTED: Remote device requested stream configuration.</li> </ul>
	<ul> <li>AVDTP_EVT_OPEN_STREAM_REQUESTED: Remote device requested to open a stream.</li> </ul>
	<ul> <li>AVDTP_EVT_START_STREAM_REQUESTED: Remote device requested to start a stream.</li> </ul>
	<ul> <li>AVDTP_EVT_CLOSE_STREAM_REQUESTED: Remote device requested to close a stream.</li> </ul>
	<ul> <li>AVDTP_EVT_SUSPEND_STREAM_REQUESTED: Remote device requested to suspend a stream.</li> </ul>
	AVDTP_EVT_ABORT_STREAM_REQUESTED: Remote device requested to abort a stream.  Copyright © 2015 SEARAN LLC. All right reserved.
	AVDTP_EVT_RECONFIGURE_STREAM_REQUESTED: Remote device requested to reconfigure a stream.

callback\_param A pointer to an arbitrary data set by a call to bt\_avdtp\_register\_callback.

### 3.11.4 Function Documentation

3.11.4.1 bt\_bool bt\_avdtp\_abort\_stream ( bt\_avdtp\_mgr\_t \* mgr, bt\_byte strm\_handle )

### Suspend a stream.

This function tries to suspend a stream by sending a request to the remote party. The stream can be in any state state except AVDTP\_STREAM\_STATE\_IDLE. As a result of this operation the AVDTP\_EVT\_ABORT\_STREAM\_

COMPLETED event will be generated. This operation cannot be rejected. The evt\_param.abort\_stream

\_requested.err\_code is always == AVDTP\_ERROR\_SUCCESS.

#### **Parameters**

mgr	AVDTP manager.
strm_handle	Stream handle.

#### Returns

- TRUE if the function succeeds, i.e. the actual request has been sent to the remote party.
- FALSE otherwise. No events will be generated.

3.11.4.2 bt\_bool bt\_avdtp\_add\_media\_rx\_buffer ( bt\_avdtp\_mgr\_t \* mgr, bt\_byte strm\_handle, bt\_media\_packet\_t \* buffer )

Add a media packet buffer to a receive queue.

The consumer of AVDTP is responsible for allocating and supplying AVDTP with buffers used to store received packets. AVDTP itself only has a queue for storing pointers to buffers supplied by the consumer. When a packet comes in AVDTP finds the first buffer large enough to hold the received packet, copies the packet to the buffer and generates a AVDTP\_EVT\_MEDIA\_PACKET\_RECEIVED event. The consumer then has to process the data in the buffer and return it back to the queue. If there is no buffers in the queue or none of the buffers is large enough the received packets is dropped. Each buffer has a field (data\_len) that holds the length of the received buffer. This field is never 0 if the buffer contains a packet. If a channel closed regardless of what has caused that and there are still buffers in the queue AVDTP generates a AVDTP\_EVT\_MEDIA\_PACKET\_RECEIVED event for each buffer and sets the data\_len to 0. This is to inform the AVDTP consumer that the buffer has not been used and can be, for example, deallocated. This function adds a buffer to the receive queue.

### **Parameters**

mgr	AVDTP manager.
strm_handle	Stream handle.
buffer	Pointer to a structure that holds the buffer and its parameters.

### Returns

- TRUE if the function succeeds.
- FALSE otherwise. The function fails only if a stream specified by the strm\_handle parameter
- does not exist. The stream can be in any state to call this function.

3.11.4.3 bt\_bool bt\_avdtp\_add\_media\_tx\_buffer ( bt\_avdtp\_mgr\_t \* mgr, bt\_byte strm\_handle, bt\_media\_packet\_t \* buffer )

Add a media packet buffer to a send queue.

When the consumer of AVDTP wants to send a packet to a remote device it calls this function. The function adds the packet to a queue and tells AVDTP that it has something to send. The packet will be send as soon as the stream goes to AVDTP\_STREAM\_STATE\_STREAMING state. When the packet has been successfully sent a A VDTP\_EVT\_MEDIA\_PACKET\_SENT is generated. Otherwise a AVDTP\_EVT\_MEDIA\_PACKET\_SEND\_FAILED is generated. Regardless of the event generated the consumer can re-use the buffer as AVDTP has removed it from the queue and gave up control over it. As in the case of received buffers, if a channel closed regardless of what has caused that and there are still buffers in the queue AVDTP generates a AVDTP\_EVT\_MEDIA\_PACKET\_SENT event for each buffer and sets the data\_len field to 0. This is to inform the AVDTP consumer that the buffer has not been used and can be, for example, deallocated.

### **Parameters**

mgr	AVDTP manager.
strm_handle	Stream handle.
buffer	Pointer to a structure that holds the buffer and its parameters.

#### Returns

- TRUE if the function succeeds.
- FALSE otherwise. The function fails only if a stream specified by the strm\_handle parameter
- does not exist. The stream can be in any state to call this function.

3.11.4.4 void bt\_avdtp\_cancel\_listen ( bt\_avdtp\_mgr\_t \* mgr, bt\_byte strm\_handle, bt\_byte sep\_id )

Cancel listening for incoming connections.

This function removes a SEP from a list of SEPS which a stream can use for incoming requests.

### **Parameters**

mgr	AVDTP manager.
strm_handle	Stream handle.
sep_id	Local SEP ID.

## Returns

- TRUE if the function succeeds.
- FALSE otherwise.

3.11.4.5 bt\_bool bt\_avdtp\_close\_stream ( bt\_avdtp\_mgr\_t \* mgr, bt\_byte strm\_handle )

## Close a stream.

This function tries to close a stream by sending a request to the remote party. The stream has to be in AVD  $\leftarrow$  TP\_STREAM\_STATE\_OPEN or AVDTP\_STREAM\_STATE\_STREAMING state. As a result of this operation the AVDTP\_EVT\_CLOSE\_STREAM\_COMPLETED event will be generated. If the stream has been closed the evt  $\leftarrow$  \_param.bt\_avdtp\_evt\_close\_stream\_completed\_t.err\_code == AVDTP\_ERROR\_SUCCES  $\leftarrow$  S. Otherwise, if the remote device for any reason cannot or does not wish to close the stream, the evt\_param.  $\leftarrow$  bt\_avdtp\_evt\_close\_stream\_completed\_t.err\_code == the error code sent by the remote.

### **Parameters**

mgr   AVDTP manager.
----------------------

strm_handle	Stream handle.

#### Returns

- TRUE if the function succeeds, i.e. the actual request has been sent to the remote party.
- FALSE otherwise. No events will be generated.

3.11.4.6 bt\_byte bt\_avdtp\_create\_stream ( bt\_avdtp\_mgr\_t \* mgr )

#### Create a stream.

This function allocates memory for storing stream's data and assigns a stream handle. The stream handle is used to manipulate the stream - open, close, configure, suspend, abort.

#### **Parameters**

	AVDTD
mgr	AVDIP manager.

### Returns

- Stream handle if the function succeeds.
- · 0 otherwise.

3.11.4.7 bt\_bool bt\_avdtp\_destroy\_stream ( bt\_avdtp\_mgr\_t \* mgr, bt\_byte strm\_handle )

### Destroy a stream.

This function frees memory used by the stream. The stream has to exist and be in the "idle" state for this function to succeed. I.e. the stream has to be closed or aborted before this function can be called.

## Parameters

mgr	AVDTP manager.
strm_handle	Stream handle.

### Returns

- TRUE if the function succeeds.
- FALSE otherwise.

3.11.4.8 bt\_bool bt\_avdtp\_disconnect ( bt\_avdtp\_mgr\_t \* mgr, bt\_bdaddr\_t \* remote\_addr )

### Disconnect from a remote device.

This function closes a control and transport channels on all streams associated with the remote device specified by the remote\_addr. As a result of this operation the following events will be generated:

- AVDTP\_EVT\_MEDIA\_PACKET\_RECEIVED: if a stream's receive queue is not empty this event is generated for each buffer with bt\_media\_packet\_t::data\_len set to 0
- AVDTP\_EVT\_MEDIA\_PACKET\_SENT: if a stream's send queue is not empty this event is generated for each buffer with bt\_media\_packet\_t::data\_len set to 0
- AVDTP\_EVT\_STREAM\_CLOSED: this event is generate if a stream is in "closing" state as a result of a request from the remote device or bt\_avdtp\_close\_stream call before bt\_avdtp\_disconnect call
- AVDTP\_EVT\_STREAM\_ABORTED: this event is generated if a stream is in "active" state at the time of bt\_avdtp\_disconnect call.

### **Parameters**

mgr	AVDTP manager.
remote_addr	The address of a remote device.

### Returns

- TRUE if disconnection has been started.
- FALSE otherwise. No events will be generated.

3.11.4.9 bt\_bool bt\_avdtp\_discover ( bt\_avdtp\_mgr\_t \* mgr, bt\_bdaddr\_t \* remote\_addr )

Discover SEPs on a remote device.

This function asks the remote device to send a list of all available SEPs. As a result of this operation the following events will be generated:

- AVDTP\_EVT\_SEP\_INFO\_RECEIVED: this event is generated for every SEP received from the remote device. the evt\_param.sep\_info\_received contains SEP information.
- AVDTP\_EVT\_DISCOVER\_COMPLETED: this event is generated after last AVDTP\_EVT\_SEP\_INFO\_R 
  ECEIVED if the remote accepted the request and the evt\_param.discover\_completed.err\_←
  code == AVDTP\_ERROR\_SUCCESS. if the remote rejected the request the evt\_param.discover←
  \_completed.err\_code == the error code sent by the remote.

### **Parameters**

mgr	AVDTP manager.
remote_addr	The address of a remote device.

## Returns

- TRUE if discover request has been sent.
- FALSE otherwise. No events will be generated.

3.11.4.10 bt\_avdtp\_codec\_t\* bt\_avdtp\_find\_codec ( bt\_avdtp\_mgr\_t \* mgr, bt\_byte codec\_type )

### Find a codec.

AVDTP in theory can support any type of codec. Each codec uses its own format for exchanging capabilities and configuration information. In order to make out implementation do not care about these formats we use a simple way of telling AVDTP how to parse and serialize codec's configuration. The consumer of AVDTP (e.g. A2DP) for each codec it wishes to support has to register a callback function (one per codec type). That callback has to perform two function. The first one is to read the configuration received from the remote device and store it in a structure defined by the consumer. The second one is to serialize the data from a structure to a format (in case of standard A2DP codecs the format is defined in A2DP specification, vendor specific codecs can define their own formats) suitable for sending as a part of a AVDTP request. This function returns a pointer to a structure that holds a pointer to a codec's callback function.

#### **Parameters**

mgr	AVDTP manager.
codec_type	Codec type. The codec_type can be one of the following values:
	AVDTP_CODEC_TYPE_SBC: SBC
	AVDTP_CODEC_TYPE_MPEG1_2_AUDIO: MPEG-1,2 (used in MP3 files)
	AVDTP_CODEC_TYPE_MPEG2_4_AAC: MPEG-2,4 AAC (used in Apple products)
	AVDTP_CODEC_TYPE_ATRAC: ATRAC (used in Sony products)
	AVDTP_CODEC_TYPE_NON_A2DP: Non-A2DP Codec

### Returns

- TRUE if the function succeeds.
- FALSE otherwise. The function fails if a callback for a codec type specified in the codec parameter
- has not been previously registered with bt\_avdtp\_register\_codec.

3.11.4.11 bt\_bool bt\_avdtp\_get\_all\_capabilities ( bt avdtp\_mgr\_t \* mgr, bt\_bdaddr\_t \* remote\_addr, bt\_byte seid\_acp )

Get remote SEP capabilities.

This function asks the remote device to send capabilities of a SEP specified by the seid\_acp. As a result of this operation the following events will be generated:

- AVDTP\_EVT\_SEP\_CAPABILITIES\_RECEIVED: this event is generated if the remote device accepted the request. the evt\_param.sep\_capabilities\_received contains SEP capabilities.
- AVDTP\_EVT\_GET\_SEP\_CAPABILITIES\_COMPLETED: this event is generated right after AVDTP\_EV← T\_SEP\_CAPABILITIES\_RECEIVED if the remote accepted the request the evt\_param.get\_sep\_← capabilities\_completed.err\_code == AVDTP\_ERROR\_SUCCESS. if the remote rejected the request the evt\_param.get\_sep\_capabilities\_completed.err\_code == the error code sent by the remote.

## **Parameters**

mgr	AVDTP manager.
remote_addr	The address of a remote device.
seid_acp	The ID of a remote SEP.

## Returns

- TRUE if discover request has been sent.
- · FALSE otherwise. No events will be generated.

 $3.11.4.12 \quad \text{bt\_bool bt\_avdtp\_get\_capabilities (} \quad \text{bt\_avdtp\_mgr\_t} * \textit{mgr}, \quad \text{bt\_bdaddr\_t} * \textit{remote\_addr}, \quad \text{bt\_byte } \textit{seid\_acp} \text{ )}$ 

Get remote SEP capabilities.

This function asks the remote device to send capabilities of a SEP specified by the  $seid\_acp$ . As a result of this operation the following events will be generated:

• AVDTP\_EVT\_SEP\_CAPABILITIES\_RECEIVED: this event is generated if the remote device accepted the request. the evt\_param.sep\_capabilities\_received contains SEP capabilities.

• AVDTP\_EVT\_GET\_SEP\_CAPABILITIES\_COMPLETED: this event is generated right after AVDTP\_EV← T\_SEP\_CAPABILITIES\_RECEIVED if the remote accepted the request the evt\_param.get\_sep\_← capabilities\_completed.err\_code == AVDTP\_ERROR\_SUCCESS. if the remote rejected the request the evt\_param.get\_sep\_capabilities\_completed.err\_code == the error code sent by the remote.

#### **Parameters**

mgr	AVDTP manager.
remote_addr	The address of a remote device.
seid_acp	The ID of a remote SEP.

### Returns

- TRUE if discover request has been sent.
- FALSE otherwise. No events will be generated.

3.11.4.13 bt\_bool bt\_avdtp\_get\_configuration ( bt avdtp mgr t \* mgr, bt\_byte strm\_handle )

Get stream configuration.

This function requests stream configuration from a remote device. As a result of this operation the following events will be generated:

- AVDTP\_EVT\_STREAM\_CONFIGURATION\_RECEIVED: this event is generated if the remote accepted the request. the <code>ebt\_para.sep\_capabilities\_received.caps</code> will contain current stream configuration.
- AVDTP\_EVT\_GET\_STREAM\_CONFIGURATION\_COMPLETED: If the remote accepted the request the evt\_param.get\_stream\_configuration\_completed.err\_code == AVDTP\_ERROR\_S↔ UCCESS. if the remote rejected the request the evt\_param.get\_stream\_configuration\_↔ completed.err\_code == the error code sent by the remote.

# Parameters

mgr	AVDTP manager.
strm_handle	Stream handle.

### Returns

- TRUE if the function succeeds, i.e. the actual request has been sent to the remote party.
- FALSE otherwise. No events will be generated.

3.11.4.14 bt\_hci\_conn\_state\_t\* bt\_avdtp\_get\_hci\_connection ( bt\_avdtp\_mgr\_t \* mgr, bt\_byte strm\_handle )

Get HCI connection for a stream.

This function returns a pointer to a structure that describes an HCl connection a stream is open on. The return value can be used to call various function from the HCl layer. For example, if an app wants to force disconnection from a remote device it can call bt\_hci\_disconnect.

## **Parameters**

mgr	AVDTP manager.
strm_handle	Stream handle.

### Returns

- Pointer to a structure that describes an HCI connection if the function succeeds.
- NULL otherwise. The function fails only if a stream specified by the strm\_handle parameter
- does not exist or there is no HCI connection between local and remote devices associated with the stream

#### Note

This function has not been implemented.

```
3.11.4.15 bt_avdtp_mgr_t* bt_avdtp_get_mgr ( void )
```

Return a pointer to an instance of the AVDTP manager.

This function returns a pointer to an instance of the AVDTP manager. There is only one instance of the manager allocated by the stack. The pointer is passed as the first parameter to all AVDTP functions.

```
3.11.4.16 bt_avdtp_sep_t* bt_avdtp_get_sep ( bt_avdtp_mgr_t * mgr, bt_byte sep_id )
```

Get a SEP info by its ID.

This function returns a pointer to bt\_avdtp\_sep\_t structure that describes a SEP previously registered with bt\_← avdtp\_register\_sep.

### **Parameters**

mgr	AVDTP manager.
sep_id	The ID of a SEP.

## Returns

- Pointer to bt\_avdtp\_sep\_t if the SEP is in the list of registered SEPs.
- NULL otherwise.

3.11.4.17 void\* bt\_avdtp\_get\_stream\_codec\_config ( bt avdtp\_mgr t \* mgr, bt\_byte strm\_handle )

Get the configuration of the codec currently used with the stream.

This function returns a pointer to a structure that contains configuration of the codec currently used with the stream. The structure returned depends on the codec. The dotstack defines structures only for SBC, MPEG-1,2 and MP← EG-2,4 AAC codecs:

- SBC: bt\_a2dp\_sbc\_config\_t (defined in a2dp\_sbc\_codec.h)
- MPEG-1,2: bt\_a2dp\_mpeg\_config\_t (defined in a2dp\_mpeg\_codec.h)
- MPEG-2,4 AAC: bt\_a2dp\_aac\_config\_t (defined in a2dp\_aac\_codec.h)

### **Parameters**

mgr	AVDTP manager.
strm_handle	Stream handle.

### Returns

 The codec's configuration if strm\_handle specifies a valid stream and the stream is in one of the following state:

```
AVDTP_STREAM_STATE_CONFIGURED
AVDTP_STREAM_STATE_OPEN
AVDTP_STREAM_STATE_STREAMING
\li NULL otherwise.
```

3.11.4.18 bt\_byte bt\_avdtp\_get\_stream\_codec\_type ( bt\_avdtp\_mgr\_t \* mgr, bt\_byte strm\_handle )

Get the type of the codec currently used with the stream.

This function returns the type of the codec currently used with the stream.

### **Parameters**

mgr	AVDTP manager.
strm_handle	Stream handle.

### Returns

 The type of the codec if strm\_handle specifies a valid stream and the stream is in one of the following states:

```
AVDTP_STREAM_STATE_CONFIGURED
AVDTP_STREAM_STATE_OPEN
AVDTP_STREAM_STATE_STREAMING

@arg The result will be one of the following values:

AVDTP_CODEC_TYPE_SBC: SBC
AVDTP_CODEC_TYPE_MPEG1_2_AUDIO: MPEG-1,2 (used in MP3 files)
AVDTP_CODEC_TYPE_MPEG2_4_AAC: MPEG-2,4 AAC (used in Apple products)
AVDTP_CODEC_TYPE_ATRAC: ATRAC (used in Sony products)
AVDTP_CODEC_TYPE_NON_A2DP: Non-A2DP Codec

@arg 0xFF otherwise.
```

3.11.4.19 bt\_avdtp\_sep\_capabilities\_t\* bt\_avdtp\_get\_stream\_config( bt\_avdtp\_mgr\_t \* mgr, bt\_byte strm\_handle)

Get stream's configuration.

This function returns a pointer to a structure holding the current configuration of stream.

### **Parameters**

mgr	AVDTP manager.
strm_handle	Stream handle.

### Returns

The stream's configuration if strm\_handle specifies a valid stream and the stream is in one of the following state:

```
AVDTP_STREAM_STATE_CONFIGURED
AVDTP_STREAM_STATE_OPEN
AVDTP_STREAM_STATE_STREAMING
\li NULL otherwise.
```

3.11.4.20 bt\_byte bt\_avdtp\_get\_stream\_local\_sep\_id ( bt\_avdtp\_mgr\_t \* mgr, bt\_byte strm\_handle )

Get stream's local SEP ID.

This function returns the ID of the local SEP associated with the stream.

### **Parameters**

mgr	AVDTP manager.
strm_handle	Stream handle.

### Returns

- The ID of the local SEP if strm handle specifies a valid stream.
- · 0 otherwise.

3.11.4.21 bt\_bdaddr\_t\* bt\_avdtp\_get\_stream\_remote\_address ( bt avdtp mgr\_t \* mgr, bt\_byte strm\_handle )

Get stream's remote BT address.

This function returns the address of the remote device associated with the stream.

#### **Parameters**

mgr	AVDTP manager.
strm_handle	Stream handle.

### Returns

- The address of the remote device if strm\_handle specifies a valid stream.
- · NULL otherwise.

3.11.4.22 bt\_byte bt\_avdtp\_get\_stream\_remote\_sep\_id ( bt\_avdtp\_mgr\_t \* mgr, bt\_byte strm\_handle )

Get stream's remote SEP ID.

This function returns the ID of the remote SEP associated with the stream.

### **Parameters**

mgr	AVDTP manager.
strm_handle	Stream handle.

### Returns

- The ID of the remote SEP if strm\_handle specifies a valid stream.
- · 0 otherwise.

3.11.4.23 bt\_byte bt\_avdtp\_get\_stream\_state ( bt\_avdtp\_mgr\_t \* mgr, bt\_byte strm\_handle )

Get local stream state.

This function returns local state of a stream specified by the strm\_handle. No request is sent to the remote party.

#### **Parameters**

mgr	AVDTP manager.
strm_handle	Stream handle.

#### Returns

The state of the stream. The result will be one of the following values:

- AVDTP\_STREAM\_STATE\_IDLE: The stream is idle. This can mean two things. The stream specified by strm handle does not exist or the stream is closed.
- AVDTP\_STREAM\_OPENING\_TRANSPORT\_CHANNELS: The stream is opening transport channels.
- AVDTP STREAM CLOSING TRANSPORT CHANNELS: The stream is closing transport channels.
- AVDTP\_STREAM\_STATE\_CONFIGURED: The stream has been configured.
- AVDTP STREAM STATE OPEN: The stream has been opened.
- AVDTP\_STREAM\_STATE\_STREAMING: The stream has been started. Depending on the local SEP type (source or sink) it means that the stream is can send or receive media packets.
- AVDTP\_STREAM\_STATE\_CLOSING: The stream is closing. This means that all transport channels
  associated with the stream are being closed. After they have been closed the stream goes to AVDTP

  \_STREAM\_STATE\_IDLE state.
- AVDTP\_STREAM\_STATE\_ABORTING: The stream is aborting. This means that all transport channels
  associated with the stream are being closed. After they have been closed the stream goes to AVDTP

  \_STREAM\_STATE\_IDLE state.

3.11.4.24 void bt\_avdtp\_init ( void )

Initialize the AVDTP layer.

This function initializes the AVDTP layer of the stack. It must be called prior to any other AVDTP function can be called.

3.11.4.25 bt\_bool bt\_avdtp\_listen ( bt\_avdtp\_mgr\_t \* mgr, bt\_byte strm\_handle, bt\_byte sep\_id )

Listen for incoming connections.

This function tells a stream that it can use a particular SEP to accept incoming requests to open it. The SEP can be associated with multiple streams but used with only one. The stream has to be closed before the SEP can be used with another stream. For outgoing connections this is not needed. Any SEP can be used with any stream given that the SEP is not already in use by another stream.

### **Parameters**

mgr	AVDTP manager.
strm_handle	Stream handle.
sep_id	Local SEP ID.

### Returns

- TRUE if the function succeeds.
- FALSE otherwise.

3.11.4.26 bt\_bool bt\_avdtp\_open\_stream ( bt avdtp\_mgr\_t \* mgr, bt\_byte strm\_handle )

Open a stream.

This function tries to open a stream by sending a request to the remote party. The stream has to be already configured with a bt\_avdtp\_set\_configuration call. As a result of this operation the AVDTP\_EVT\_OPEN\_STRE AM\_COMPLETED event will be generated. If the stream has been open the evt\_param.open\_stream\_completed.err\_code == AVDTP\_ERROR\_SUCCESS. Otherwise, if the remote device for any reason cannot or does not wish to open the stream, the evt\_param.open\_stream\_completed.err\_code == the error code sent by the remote.

#### **Parameters**

mgr	AVDTP manager.
strm_handle	Stream handle.

### Returns

- TRUE if the function succeeds, i.e. the actual request has been sent to the remote party.
- · FALSE otherwise. No events will be generated.

```
3.11.4.27 bt_bool bt_avdtp_reconfigure_stream ( bt_avdtp_mgr_t * mgr, bt_byte strm_handle, bt_avdtp_sep_capabilities_t * caps_)
```

### Reconfigure stream.

This function tries to change the stream's configuration. For this function to succeed the stream has to be open. As a result of this operation the AVDTP\_EVT\_STREAM\_RECONFIGURE\_COMPLETED event will be generated. If reconfiguration was a success the evt\_param.stream\_reconfigure\_completed.err\_code ==  $A \leftarrow VDTP_ERROR_SUCCESS$ . Otherwise the evt\_param.stream\_reconfigure\_completed.err\_code == the error code sent by the remote.

### **Parameters**

mgr	AVDTP manager.
strm_handle	Stream handle.
caps	New stream configuration.

### Returns

- TRUE if the function succeeds, i.e. the actual request has been sent to the remote party.
- FALSE otherwise. No events will be generated.
- 3.11.4.28 void bt\_avdtp\_register\_callback ( bt\_avdtp\_mgr\_t \* mgr, bt\_avdtp\_mgr\_callback\_fp callback, void \* callback\_param )

## Register a AVDTP application callback.

In order to be notified of various events a consumer of the AVDTP layer has to register a callback function. The stack will call this function whenever a new event has been generated passing the code of the event as the second parameter. The event can be one of the following values:

- AVDTP\_EVT\_CTRL\_CHANNEL\_CONNECTED: Control channel connected.
- AVDTP\_EVT\_CTRL\_CHANNEL\_DISCONNECTED: Control channel disconnected.
- AVDTP\_EVT\_CTRL\_CONNECTION\_FAILED: Control channel connection failed (generated only if control connection has been initiated by the local device).
- AVDTP\_EVT\_DISCOVER\_COMPLETED: Local device completed discovering remote SEPs.
- AVDTP\_EVT\_GET\_SEP\_CAPABILITIES\_COMPLETED: Local device received a response to Get SEP capabilities operation.

 AVDTP\_EVT\_SET\_STREAM\_CONFIGURATION\_COMPLETED: Local device received a response to Set stream configuration operation.

- AVDTP\_EVT\_GET\_STREAM\_CONFIGURATION\_COMPLETED: Local device received a response to Get stream configuration operation.
- AVDTP\_EVT\_STREAM\_RECONFIGURE\_COMPLETED: Local device received a response to Reconfigure stream operation.
- AVDTP\_EVT\_OPEN\_STREAM\_COMPLETED: Local device received a response to Open stream operation.
- AVDTP EVT START STREAM COMPLETED: Local device received a response to Start stream operation.
- AVDTP\_EVT\_CLOSE\_STREAM\_COMPLETED: Local device received a response to Close stream operation.
- AVDTP\_EVT\_SUSPEND\_STREAM\_COMPLETED: Local device received a response to Suspend stream operation.
- AVDTP\_EVT\_STREAM\_SECURITY\_CONTROL\_COMPLETED: Local device received a response to Stream security control operation.
- AVDTP\_EVT\_ABORT\_STREAM\_COMPLETED: Local device received a response to Abort stream operation.
- AVDTP EVT SEP INFO RECEIVED: SEP information received.
- AVDTP EVT SEP CAPABILITIES RECEIVED: SEP capabilities received.
- AVDTP\_EVT\_STREAM\_CONFIGURATION\_RECEIVED: Stream configuration received.
- AVDTP\_EVT\_SET\_STREAM\_CONFIGURATION\_REQUESTED: Remote device requested stream configuration.
- AVDTP EVT OPEN STREAM REQUESTED: Remote device requested to open a stream.
- AVDTP\_EVT\_START\_STREAM\_REQUESTED: Remote device requested to start a stream.
- AVDTP\_EVT\_CLOSE\_STREAM\_REQUESTED: Remote device requested to close a stream.
- · AVDTP EVT SUSPEND STREAM REQUESTED: Remote device requested to suspend a stream.
- AVDTP EVT\_ABORT\_STREAM\_REQUESTED: Remote device requested to abort a stream.
- AVDTP\_EVT\_RECONFIGURE\_STREAM\_REQUESTED: Remote device requested to reconfigure a stream.
- AVDTP\_EVT\_MEDIA\_PACKET\_RECEIVED: Remote device sent a media packet.
- AVDTP\_EVT\_STREAM\_CONFIGURED: A stream has been configured (This event is generated right after AVDTP\_EVT\_SET\_STREAM\_CONFIGURATION\_REQUESTED if the local devices accepted the request).
- AVDTP\_EVT\_STREAM\_RECONFIGURED: A stream has been re-configured (This event is generated right after AVDTP\_EVT\_RECONFIGURE\_STREAM\_REQUESTED if the local devices accepted the request).
- AVDTP\_EVT\_STREAM\_OPENED: A stream has been opened (This event is generated as a result of local
  or remote stream opening request).
- AVDTP\_EVT\_STREAM\_STARTED: A stream has been started (This event is generated right after AVDT
   — P\_EVT\_START\_STREAM\_REQUESTED if the local devices accepted the request).
- AVDTP\_EVT\_STREAM\_CLOSED: A stream has been close (This event is generated right after AVDTP\_

  EVT\_CLOSE\_STREAM\_REQUESTED if the local devices accepted the request).
- AVDTP\_EVT\_STREAM\_SUSPENDED: A stream has been suspended (This event is generated right after AVDTP\_EVT\_SUSPEND\_STREAM\_REQUESTED if the local devices accepted the request).

- AVDTP\_EVT\_STREAM\_ABORTED: A stream has been aborted (This event is generated right after AVDT
   — P\_EVT\_SUSPEND\_STREAM\_REQUESTED if the local devices accepted the request. It is also generated if
   connection between devices has been terminated by means other than AVDTP signaling, e.g. devices going
   out of rage).
- AVDTP\_EVT\_MEDIA\_PACKET\_SENT: The local device has successfully sent a media packet to the remote device.
- AVDTP\_EVT\_MEDIA\_PACKET\_SEND\_FAILED: The local device was not able to send a media packet to the remote device.

#### **Parameters**

mgr	AVDTP manager.
callback	The callback function that will be called when the AVDTP generates an event.
callback_param	A pointer to arbitrary data to be passed to the callback callback.

3.11.4.29 bt\_bool bt\_avdtp\_register\_codec ( bt\_avdtp\_mgr\_t \* mgr, bt\_avdtp\_codec\_t \* codec )

### Register a codec.

AVDTP in theory can support any type of codec. Each codec uses its own format for exchanging capabilities and configuration information. In order to make out implementation do not care about these formats we use a simple way of telling AVDTP how to parse and serialize codec's configuration. The consumer of AVDTP (e.g. A2DP) for each codec it wishes to support has to register a callback function (one per codec type). That callback has to perform two function. The first one is to read the configuration received from the remote device and store it in a structure defined by the consumer. The second one is to serialize the data from a structure to a format (in case of standard A2DP codecs the format is defined in A2DP specification, vendor specific codecs can define their own formats) suitable for sending as a part of a AVDTP request. This function adds a codec's callback function to an internal list.

## **Parameters**

mgr	AVDTP manager.
codec	Pointer to a structure specifying codec type and a callback.

### Returns

- TRUE if the function succeeds.
- FALSE otherwise. The function fails if there already is a callback for a codec type specified in the codec parameter.

3.11.4.30 bt\_byte bt\_avdtp\_register\_sep ( bt\_avdtp\_mgr\_t \* mgr, bt\_byte type, const bt\_avdtp\_sep\_capabilities\_t \* caps )

Register a SEP with the local AVDTP manager.

This function is used to make a list of SEPs supported by the local ADVTP entity.

### **Parameters**

mgr	AVDTP manager.
type	The type of a SEP. The type can be one of the following values:
	<ul><li>AVDTP_SEP_TYPE_SOURCE: The SEP is a source.</li><li>AVDTP_SEP_TYPE_SINK: The SEP is a sink.</li></ul>
caps	The capabilities of a SEP.

#### Returns

- ID of a SEP if the function succeeds.
- · FALSE otherwise.

3.11.4.31 bt\_bool bt\_avdtp\_remove\_media\_rx\_buffer ( bt\_avdtp\_mgr\_t \* mgr, bt\_byte strm\_handle, bt\_media\_packet\_t \* buffer )

Remove a media packet buffer from a receive queue.

The consumer of AVDTP is responsible for allocating and supplying AVDTP with buffers used to store received packets. AVDTP itself only has a queue for storing pointers to buffers supplied by the consumer. When a packet comes in AVDTP finds the first buffer large enough to hold the received packet, copies the packet to the buffer and generates a AVDTP\_EVT\_MEDIA\_PACKET\_RECEIVED event. The consumer then has to process the data in the buffer and return it back to the queue. If there is no buffers in the queue or none of the buffers is large enough the received packets is dropped. Each buffer has a field (data\_len) that holds the length of the received buffer. This field is never 0 if the buffer contains a packet. If a channel closed regardless of what has caused that and there are still buffers in the queue AVDTP generates a AVDTP\_EVT\_MEDIA\_PACKET\_RECEIVED event for each buffer and sets the data\_len to 0. This is to inform the AVDTP consumer that the buffer has not been used and can be, for example, deallocated. This function removes a buffer from the receive queue.

### **Parameters**

mgr	AVDTP manager.
strm_handle	Stream handle.
buffer	Pointer to a structure that holds the buffer and its parameters.

#### Returns

- TRUE if the function succeeds.
- FALSE otherwise. The function fails only if a stream specified by the strm\_handle parameter
- · does not exist. The stream can be in any state to call this function.

3.11.4.32 bt\_bool bt\_avdtp\_remove\_media\_tx\_buffer ( bt\_avdtp\_mgr\_t \* mgr, bt\_byte strm\_handle, bt\_media\_packet\_t \* buffer )

Remove a media packet buffer from a send queue.

When the consumer of AVDTP wants to send a packet to a remote device it calls bt\_avdtp\_add\_media\_tx\_buffer function. The function adds the packet to a queue and tells AVDTP that it has something to send. The packet will be send as soon as the stream goes to AVDTP\_STREAM\_STATE\_STREAMING state. The consumer has a chance to remove a packet from the queue before it has been sent to a remote device by calling bt\_avdtp\_remove\_media \_\_tx\_buffer.

## **Parameters**

mgr	AVDTP manager.
strm_handle	Stream handle.
buffer	Pointer to a structure that holds the buffer and its parameters.

### Returns

- TRUE if the function succeeds.
- FALSE otherwise. The function fails only if a stream specified by the strm\_handle parameter
- does not exist. The stream can be in any state to call this function.

### Note

This function has not been implemented.

3.11.4.33 bt\_bool bt\_avdtp\_report\_delay ( bt\_avdtp\_mgr\_t \* mgr, bt\_byte strm\_handle, bt\_uint delay )

Report delay value of a Sink to a Source.

This function sends the delay value of a Sink to a Source. This enables synchronous playback of audio and video. Delay reports are always sent from the Sink to the Source. If the Sink's delay report has been accepted by the Source the evt\_param.delay\_report\_completed.err\_code == AVDTP\_ERROR\_SUCCESS. Otherwise the evt\_param.delay\_report\_completed.err\_code == the error code sent by the Source.

#### **Parameters**

mgr	AVDTP manager.
strm_handle	Stream handle.
delay	The delay value in 1/10 milliseconds.

#### Returns

- TRUE if the function succeeds, i.e. the actual request has been sent to the remote party.
- FALSE otherwise. No events will be generated.

3.11.4.34 bt\_bool bt\_avdtp\_security\_control ( bt\_avdtp\_mgr\_t \* mgr, bt\_byte strm\_handle, bt\_byte \* sc\_data, bt\_byte \* sc\_data\_len )

Exchange content protection control data.

This function tries to establish content protection by sending a request to the remote party. The stream can be in any state state except AVDTP\_STREAM\_STATE\_IDLE, AVDTP\_STREAM\_STATE\_CLOSING, AVDTP\_STCREAM\_STATE\_ABORTING. As a result of this operation the AVDTP\_EVT\_STREAM\_SECURITY\_CONTROLCOMPLETED event will be generated. If the stream's content protection data has been accepted by the remote party the evt\_param.security\_control\_completed.err\_code == AVDTP\_ERROR\_SUCCCESS. Otherwise the evt\_param.security\_control\_completed.err\_code == the error code sent by the remote.

### Note

The dotstack does not support content protection. Although the request can be sent it will not affect the operation of the AVDTP in any way.

## Parameters

mgr	AVDTP manager.
strm_handle	Stream handle.

### Returns

- TRUE if the function succeeds, i.e. the actual request has been sent to the remote party.
- FALSE otherwise. No events will be generated.

3.11.4.35 bt\_bool bt\_avdtp\_set\_configuration ( bt\_avdtp\_mgr\_t \* mgr, bt\_byte strm\_handle, bt\_bdaddr\_t \* remote\_addr, bt\_byte seid\_int, bt\_byte seid\_acp, const bt\_avdtp\_sep\_capabilities\_t \* caps\_)

Set stream configuration.

This function tries to configure a stream before opening it. As a result of this operation the AVDTP\_EVT\_SET\_  $\leftarrow$  STREAM\_CONFIGURATION\_COMPLETED event will be generated. If configuration was a success the evt\_  $\leftarrow$  param.set\_stream\_configuration\_completed.err\_code == AVDTP\_ERROR\_SUCCESS. Otherwise the evt\_param.set\_stream\_configuration\_completed.err\_code == the error code sent by the remote and evt\_param.set\_stream\_configuration\_completed.svc\_category == the value of the first Service Category to fail.

### **Parameters**

mgr	AVDTP manager.
strm_handle	Stream handle.
remote_addr	The address of a remote device.
seid_int	Local SEP ID.
seid_acp	Remote SEP ID.
caps	Stream configuration.

#### Returns

- TRUE if the function succeeds.
- FALSE otherwise. No events will be generated.

3.11.4.36 bt\_bool bt\_avdtp\_start ( bt\_avdtp\_mgr\_t \* mgr )

## Start the AVDTP layer.

This function makes the AVDTP layer ready to accept connection requests from remote device. To make an outgoing connection calling this function is not required.

#### **Parameters**

mar	AVDTP manager.
9.	/ / / / managen

#### Returns

- TRUE if the function succeeds.
- FALSE otherwise.

3.11.4.37 bt\_bool bt\_avdtp\_start\_stream ( bt\_avdtp\_mgr\_t \* mgr, bt\_byte strm\_handle )

## Start a stream.

This function tries to start a stream by sending a request to the remote party. The stream has to be in AVDTP\_STR← EAM\_STATE\_OPEN state. The stream goes to this state as a result of successful configuration or suspension (both can be initiated by either party). As a result of this operation the AVDTP\_EVT\_START\_STREAM\_COMPLETED event will be generated. If the stream has been open the evt\_param.start\_stream\_requested.err← \_code == AVDTP\_ERROR\_SUCCESS. Otherwise, if the remote device for any reason cannot or does not wish to start the stream, the evt\_param.start\_stream\_requested.err\_code == the error code sent by the remote.

### **Parameters**

mgr	AVDTP manager.
strm_handle	Stream handle.

## Returns

- TRUE if the function succeeds, i.e. the actual request has been sent to the remote party.
- FALSE otherwise. No events will be generated.

3.11.4.38 bt\_bool bt\_avdtp\_suspend\_stream ( bt\_avdtp\_mgr\_t \* mgr, bt\_byte strm\_handle )

### Suspend a stream.

This function tries to suspend a stream by sending a request to the remote party. The stream has to be in AVDT P\_STREAM\_STATE\_STREAMING state. As a result of this operation the AVDTP\_EVT\_SUSPEND\_STREAM\_COMPLETED event will be generated. If the stream has been suspended the evt\_param.bt\_avdtp\_evtc\_suspend\_stream\_requested\_t.err\_code == AVDTP\_ERROR\_SUCCESS. Otherwise, if the remote device for any reason cannot or does not wish to suspend the stream, the evt\_param.bt\_avdtp\_evt\_code suspend\_stream\_requested\_t.err\_code == the error code sent by the remote.

#### **Parameters**

mgr	AVDTP manager.
strm_handle	Stream handle.

#### Returns

- TRUE if the function succeeds, i.e. the actual request has been sent to the remote party.
- FALSE otherwise. No events will be generated.

3.11.4.39 bt\_bool bt\_avdtp\_unregister\_codec ( bt avdtp\_mgr\_t \* mgr, bt\_byte codec\_type )

### Unregister a codec.

AVDTP in theory can support any type of codec. Each codec uses its own format for exchanging capabilities and configuration information. In order to make out implementation do not care about these formats we use a simple way of telling AVDTP how to parse and serialize codec's configuration. The consumer of AVDTP (e.g. A2DP) for each codec it wishes to support has to register a callback function (one per codec type). That callback has to perform two function. The first one is to read the configuration received from the remote device and store it in a structure defined by the consumer. The second one is to serialize the data from a structure to a format (in case of standard A2DP codecs the format is defined in A2DP specification, vendor specific codecs can define their own formats) suitable for sending as a part of a AVDTP request. This function removes a codec's callback function from an internal list.

### **Parameters**

mgr	AVDTP manager.
codec_type	Codec type. The codec_type can be one of the following values:
	AVDTP_CODEC_TYPE_SBC: SBC
	AVDTP_CODEC_TYPE_MPEG1_2_AUDIO: MPEG-1,2 (used in MP3 files)
	AVDTP_CODEC_TYPE_MPEG2_4_AAC: MPEG-2,4 AAC (used in Apple products)
	AVDTP_CODEC_TYPE_ATRAC: ATRAC (used in Sony products)
	AVDTP_CODEC_TYPE_NON_A2DP: Non-A2DP Codec

## Returns

- TRUE if the function succeeds.
- FALSE otherwise. The function fails if a callback for a codec type specified in the codec parameter
- has not been previously registered with bt\_avdtp\_register\_codec.

# 3.12 Configuration

This module describes parameters used to configure AVDTP layer.

#### **Macros**

• #define AVDTP MAX SEP

Maximum number of SEPs that can be exposed by a local device.

#define AVDTP\_MAX\_STREAMS

Maximum number of streams that can be exposed by a local device.

#define AVDTP MAX REMOTE DEVICES

Maximum number of remote devices a local device can be connected to.

#define AVDTP\_MAX\_CMD\_BUFFERS

Maximum number of command buffers.

• #define AVDTP MAX TRANSPORT CHANNELS

Maximum number of transport channels.

#define AVDTP\_MAX\_TX\_BUFFER\_LEN

Size of the transmit buffer.

#define AVDTP\_MAX\_CMD\_PARAM\_LEN

Maximum length of control command parameters.

• #define AVDTP\_CODEC\_CONFIG\_BUFFER\_LEN

Size of the buffer used to store codec specific configuration.

### 3.12.1 Detailed Description

This module describes parameters used to configure AVDTP layer.

dotstack is customized using a configuration file. The configuration file tailors the dotstack to the application being built. It has to have the structure shown below.

```
#include "cdbt/bt/bt_std.h"
// HCI, L2CAP and SDP must always be present
// HCI configuration parameters
#define HCI_MAX_CMD_BUFFERS
#define HCI_MAX_DATA_BUFFERS
                                        . . .
#define HCI_MAX_HCI_CONNECTIONS
#define HCI_RX_BUFFER_LEN
#define HCI_TX_BUFFER_LEN
#define HCI L2CAP BUFFER LEN
#define HCI_MAX_CMD_PARAM_LEN
// L2CAP configuration parameters
#define L2CAP_MAX_CMD_BUFFERS
#define L2CAP_MAX_FRAME_BUFFERS
#define L2CAP_MAX_PSMS
#define L2CAP_MAX_CHANNELS
// SDP configuration parameters
#define SDP_MAX_SEARCH_RESULT_LEN
#define SDP_MAX_ATTRIBUTE_RESULT_LEN
\ensuremath{//} Depending on protocols and profiles used below goes configuration parameters
// for each used module. E.g., to use and configure AVDTP & A2DP (this one does not need configuration),
// the following values must be defined:
#define BT_INCLUDE_AVDTP
                                        // tells dotstack to compile in AVDTP support
#define AVDTP_MAX_SEP
#define AVDTP MAX STREAMS
#define AVDTP_MAX_REMOTE_DEVICES
#define AVDTP_MAX_CMD_BUFFERS
#define AVDTP_MAX_TRANSPORT_CHANNELS ... #define AVDTP_MAX_TX_BUFFER_LEN ...
#define AVDTP_MAX_TX_BUFFER_LEN
#define AVDTP_MAX_CMD_PARAM_LEN
#define AVDTP_CODEC_CONFIG_BUFFER_LEN ...
#include "cdbt/bt/bt_oem_config.h"
```

3.12 Configuration 109

### 3.12.2 Macro Definition Documentation

### 3.12.2.1 #define AVDTP\_CODEC\_CONFIG\_BUFFER\_LEN

Size of the buffer used to store codec specific configuration.

Each codec uses unique configuration which can take different amount of memory. This parameter defines the size of the buffer for storing codec's configuration. The value of 16 is sufficient for SBC, AAC and MPEG1,2. If vendor specific codec is to be used this value may need to increased.

### 3.12.2.2 #define AVDTP\_MAX\_CMD\_BUFFERS

Maximum number of command buffers.

This parameter defines the number of buffers reserved for sending commands to a remote device over its control channel. Each channel uses its own buffers so the total number of buffers is AVDTP\_MAX\_REMOTE\_DEVICES \* AVDTP\_MAX\_CMD\_BUFFERS. The minimum value is 1. The maximum value is 255. 2 is usually sufficient.

### 3.12.2.3 #define AVDTP\_MAX\_CMD\_PARAM\_LEN

Maximum length of control command parameters.

This parameter defines the maximum length of all command parameters. The value should not exceed AVDTP\_← MAX\_TX\_BUFFER\_LEN - 2 (command header).

#### 3.12.2.4 #define AVDTP MAX REMOTE DEVICES

Maximum number of remote devices a local device can be connected to.

This parameter defines the number of remote devices a local device can have simultaneous connections to (i.e. control channels). This value should not exceed AVDTP\_MAX\_STREAMS. For each remote device AVDTP creates one control channel regardless of the number of streams between the local and the remote devices. Assuming that the local devices wants to have only one channel with each remote device and if AVDTP\_MAX\_REMOTE\_DEV ← ICES > AVDTP\_MAX\_STREAMS all memory reserved for devices in excess of AVDTP\_MAX\_STREAMS will be wasted. The minimum value is 1. The maximum value is 255.

### 3.12.2.5 #define AVDTP\_MAX\_SEP

Maximum number of SEPs that can be exposed by a local device.

This parameter defines the number of SEPs an application can expose to remote devices. The minimum value is 1. The maximum value is 255.

### 3.12.2.6 #define AVDTP\_MAX\_STREAMS

Maximum number of streams that can be exposed by a local device.

This parameter defines the number of streams an application can open between local and remote devices. This value can be different from AVDTP\_MAX\_SEP but should not exceed it. Since each SEP can only be used once the local device can only have as much streams as there are SEPs. If AVDTP\_MAX\_STREAMS > AVDTP\_MAX\_SEP all memory reserved for streams in excess of AVDTP\_MAX\_SEP will be wasted. The minimum value is 1. The maximum value is 255.

### 3.12.2.7 #define AVDTP\_MAX\_TRANSPORT\_CHANNELS

Maximum number of transport channels.

Depending on the SEP capabilities (multiplexing, recovery, reporting) each stream may need up to 3 transport channels. E.g., if multiplexing is not supported and recovery and reporting are supported a stream will use 3 transport channels - 1 for media transport session, 1 for recovery transport session and 1 for reporting transport session. If multiplexing is not supported and only reporting is supported the stream will use 2 transport channels - 1 for media transport session and 1 for reporting transport session. If multiplexing is supported the stream will need only 1 transport channel for all supported transport session. With multiplexing even different streams can share the same transport channel. dotstack currently does not support multiplexing, recovery and reporting. So each stream needs its own transport channel for it media transport session. Hence, AVDTP\_MAX\_TRANSPORT\_CHANNELS must be equal to AVDTP MAX STREAMS

3.12.2.8 #define AVDTP\_MAX\_TX\_BUFFER\_LEN

Size of the transmit buffer.

This parameter defines the size of the buffer used to send AVDTP control commands to L2CAP layer. Each control channel has it own buffer so the total amount of memory allocated for these buffers is AVDTP\_MAX\_TX\_BUFFE  $\leftarrow$  R\_LEN) \* AVDTP\_MAX\_REMOTE\_DEVICES. The minimum value is 1. The maximum value is 255. The value of 32 is usually sufficient.

# 3.13 Advanced Audio Distribution Profile (A2DP)

The Advanced Audio Distribution Profile (A2DP) defines the protocols and procedures that realize distribution of audio content of high-quality in mono or stereo on ACL channels.

### **Data Structures**

• struct bt a2dp evt open and start stream completed t

Parameter to A2DP\_EVT\_OPEN\_AND\_START\_STREAM\_COMPLETED event.

· union bt a2dp event t

Parameter to an application callback.

struct bt\_a2dp\_mgr\_t

A2DP manager.

### **Macros**

- #define bt\_a2dp\_find\_codec(mgr, codec\_type) bt\_avdtp\_find\_codec(mgr->avdtp\_mgr, codec\_type)
   Find a codec.
- #define bt\_a2dp\_connect(mgr, remote\_addr) bt\_avdtp\_connect(mgr->avdtp\_mgr, remote\_addr)
   Connect to a remote device.
- #define bt\_a2dp\_connect\_ex(mgr, remote\_addr, acl\_config) bt\_avdtp\_connect\_ex(mgr->avdtp\_mgr, remote\_addr, acl\_config)

Connect to a remote device.

- #define bt\_a2dp\_disconnect(mgr, remote\_addr) bt\_avdtp\_disconnect(mgr->avdtp\_mgr, remote\_addr)
   Disconnect from a remote device.
- #define bt\_a2dp\_register\_sink(mgr, caps) bt\_avdtp\_register\_sep(mgr->avdtp\_mgr, AVDTP\_SEP\_TYPE\_
   SINK, caps)

Register a Sink SEP with the local A2DP manager.

Register a Source SEP with the local A2DP manager.

- #define bt\_a2dp\_discover(mgr, remote\_addr) bt\_avdtp\_discover(mgr->avdtp\_mgr, remote\_addr)
  - Discover SEPs on a remote device.
- #define bt\_a2dp\_get\_capabilities(mgr, remote\_addr, seid\_acp) bt\_avdtp\_get\_capabilities(mgr->avdtp\_mgr, remote\_addr, seid\_acp)

Get remote SEP capabilities.

 #define bt\_a2dp\_get\_all\_capabilities(mgr, remote\_addr, seid\_acp) bt\_avdtp\_get\_all\_capabilities(mgr->avdtp\_mgr, remote\_addr, seid\_acp)

Get remote SEP capabilities.

- #define bt\_a2dp\_create\_stream(mgr) bt\_avdtp\_create\_stream(mgr->avdtp\_mgr)
  - Create a stream.
- #define bt\_a2dp\_destroy\_stream(mgr, strm\_handle) bt\_avdtp\_destroy\_stream(mgr->avdtp\_mgr, strm\_
  handle)

Destroy a stream.

- #define bt\_a2dp\_listen(mgr, strm\_handle, sep\_id) bt\_avdtp\_listen(mgr->avdtp\_mgr, strm\_handle, sep\_id)

  Listen for incoming connections.

Cancel listening for incoming connections.

#define bt\_a2dp\_reconfigure\_stream(mgr, strm\_handle, caps) bt\_avdtp\_reconfigure\_stream(mgr->avdtp\_

mgr, strm\_handle, caps)

Reconfigure stream.

#define bt\_a2dp\_get\_stream\_state(mgr, strm\_handle) bt\_avdtp\_get\_stream\_state(mgr->avdtp\_mgr, strm
 —handle)

Get local stream state.

 #define bt\_a2dp\_get\_stream\_local\_sep\_id(mgr, strm\_handle) bt\_avdtp\_get\_stream\_local\_sep\_id(mgr->avdtp\_mgr, strm\_handle)

Get stream's local SEP ID.

 #define bt\_a2dp\_get\_stream\_remote\_sep\_id(mgr, strm\_handle) bt\_avdtp\_get\_stream\_remote\_sep\_id(mgr->avdtp\_mgr, strm\_handle)

Get stream's remote SEP ID.

#define bt\_a2dp\_get\_stream\_remote\_address(mgr, strm\_handle) bt\_avdtp\_get\_stream\_remote\_
 address(mgr->avdtp\_mgr, strm\_handle)

Get stream's remote BT address.

 #define bt\_a2dp\_get\_stream\_codec\_type(mgr, strm\_handle) bt\_avdtp\_get\_stream\_codec\_type(mgr->avdtp\_mgr, strm\_handle)

Get the type of the codec currently used with the stream.

 #define bt\_a2dp\_get\_stream\_codec\_config(mgr, strm\_handle) bt\_avdtp\_get\_stream\_codec\_config(mgr->avdtp\_mgr, strm\_handle)

Get the configuration of the codec currently used with the stream.

• #define bt\_avdtp\_get\_stream\_config(mgr, strm\_handle) bt\_avdtp\_get\_stream\_config(mgr->avdtp\_mgr, strm\_handle)

Get stream's configuration.

- #define bt\_a2dp\_start\_stream(mgr, strm\_handle) bt\_avdtp\_start\_stream(mgr->avdtp\_mgr, strm\_handle)
   Start a stream
- #define bt\_a2dp\_close\_stream(mgr, strm\_handle) bt\_avdtp\_close\_stream(mgr->avdtp\_mgr, strm\_handle)
   Close a stream.

Suspend a stream.

- #define bt\_a2dp\_abort\_stream(mgr, strm\_handle) bt\_avdtp\_abort\_stream(mgr->avdtp\_mgr, strm\_handle) Suspend a stream.
- #define bt\_a2dp\_get\_hci\_connection(mgr, strm\_handle) bt\_avdtp\_get\_hci\_connection(mgr->avdtp\_mgr, strm\_handle)

Get HCI connection for a stream.

 #define bt\_a2dp\_add\_media\_rx\_buffer(mgr, strm\_handle, buffer) bt\_avdtp\_add\_media\_rx\_buffer(mgr->avdtp\_mgr, strm\_handle, buffer)

Add a media packet buffer to a receive queue.

#define bt\_a2dp\_remove\_media\_rx\_buffer(mgr, strm\_handle, buffer) bt\_avdtp\_remove\_media\_rx\_
 buffer(mgr->avdtp\_mgr, strm\_handle, buffer)

Remove a media packet buffer from a receive queue.

 #define bt\_a2dp\_add\_media\_tx\_buffer(mgr, strm\_handle, buffer) bt\_avdtp\_add\_media\_tx\_buffer(mgr->avdtp\_mgr, strm\_handle, buffer)

Add a media packet buffer to a send queue.

#define bt\_a2dp\_remove\_media\_tx\_buffer(mgr, strm\_handle, buffer) bt\_avdtp\_remove\_media\_tx\_
 buffer(mgr->avdtp\_mgr, strm\_handle, buffer)

Remove a media packet buffer from a send queue.

## **Typedefs**

- typedef void(\* bt\_a2dp\_find\_server\_callback\_fp) (bt\_uint supported\_features, bt\_bool found, void \*param)

  Notify the application of the result of searching for a remote A2DP entity (sourse or sink)
- typedef void(\* bt\_a2dp\_mgr\_callback\_fp) (bt\_a2dp\_mgr\_t \*mgr, bt\_byte evt, bt\_a2dp\_event\_t \*evt\_param, void \*callback\_param)

A2DP application callback.

# **Functions**

bt\_a2dp\_mgr\_t \* bt\_a2dp\_get\_mgr (void)

Return a pointer to an instance of the A2DP manager.

void bt\_a2dp\_init (void)

Initialize the A2DP layer.

bt\_bool bt\_a2dp\_start (bt\_a2dp\_mgr\_t \*mgr)

Start the A2DP layer.

void bt\_a2dp\_register\_callback (bt\_a2dp\_mgr\_t \*mgr, bt\_a2dp\_mgr\_callback\_fp callback, void \*callback\_ 
 param)

Register a A2DP application callback.

bt\_bool bt\_a2dp\_open\_and\_start\_stream (bt\_a2dp\_mgr\_t \*mgr, bt\_byte strm\_handle, bt\_bdaddr\_
 t \*remote\_addr, bt\_byte seid\_int, bt\_byte seid\_acp, const bt\_avdtp\_sep\_capabilities\_t \*caps)

Open & start a stream.

bt\_bool bt\_a2dp\_find\_source (bt\_bdaddr\_t \*deviceAddress, bt\_a2dp\_find\_server\_callback\_fp callback, bt
 \_sdp\_client\_callback\_fp client\_callback, void \*callback\_param)

Find source

bt\_bool bt\_a2dp\_find\_sink (bt\_bdaddr\_t \*deviceAddress, bt\_a2dp\_find\_server\_callback\_fp callback, bt\_
 sdp\_client\_callback\_fp client\_callback, void \*callback\_param)

Find sink

• void bt\_a2dp\_register\_mpeg\_codec (bt\_a2dp\_mgr\_t \*mgr)

Register default MPEG codec.

void bt\_a2dp\_register\_aac\_codec (bt\_a2dp\_mgr\_t \*mgr)

Register default AAC codec.

## Source features

• #define A2DP\_SOURCE\_FEATURE\_PLAYER 1

Player.

#define A2DP\_SOURCE\_FEATURE\_MICROPHONE 2

Mic

#define A2DP\_SOURCE\_FEATURE\_TUNER 4

Tuner.

#define A2DP\_SOURCE\_FEATURE\_MIXER 8

Mixer.

### Sink features

The following is a list of features a sink can support. The supported features are specified in the corresponding SDP service record and can be retrieved with bt\_a2dp\_find\_sink.

#define A2DP\_SINK\_FEATURE\_HEADPHONE 1

Headphone.

• #define A2DP SINK FEATURE SPEAKER 2

Speaker.

#define A2DP\_SINK\_FEATURE\_RECORDER 4

Recorder

• #define A2DP\_SINK\_FEATURE\_AMPLIFIER 8

Amplifier.

### **Events**

The following is a list of events A2DP layer generates and can report to the upper layer when it completes executing an operation initiated by either local or remote device.

• #define A2DP\_EVT\_CTRL\_CHANNEL\_CONNECTED AVDTP\_EVT\_CTRL\_CHANNEL\_CONNECTED

This event is generated when a control channel between two AVDTP entities has been established.

• #define A2DP\_EVT\_CTRL\_CHANNEL\_DISCONNECTED AVDTP\_EVT\_CTRL\_CHANNEL\_DISCONNEC

TED

This event is generated when a control channel between two AVDTP entities has been terminated.

#define A2DP EVT CTRL CONNECTION FAILED AVDTP EVT CTRL CONNECTION FAILED

This event is generated when a local device failed to create a control channel between two AVDTP entities.

#define A2DP\_EVT\_DISCOVER\_SEP\_COMPLETED AVDTP\_EVT\_DISCOVER\_COMPLETED

This event is generated when a local device received a response (either positive or negative) to a "discover" request.

#define A2DP\_EVT\_GET\_SEP\_CAPABILITIES\_COMPLETED AVDTP\_EVT\_GET\_SEP\_CAPABILITIES

 COMPLETED

This event is generated when a local device received a response (either positive or negative) to a "get SEP capabilities" request.

#define A2DP\_EVT\_SET\_STREAM\_CONFIGURATION\_COMPLETED AVDTP\_EVT\_SET\_STREAM\_C
 ONFIGURATION COMPLETED

This event is generated when a local device received a response (either positive or negative) to a "set stream configuration" request.

#define A2DP\_EVT\_GET\_STREAM\_CONFIGURATION\_COMPLETED AVDTP\_EVT\_GET\_STREAM\_C
 ONFIGURATION\_COMPLETED

This event is generated when a local device received a response (either positive or negative) to a "get stream configuration" request.

#define A2DP\_EVT\_RECONFIGURE\_STREAM\_COMPLETED AVDTP\_EVT\_STREAM\_RECONFIGURE 
 COMPLETED

This event is generated when a local device received a response (either positive or negative) to a "change stream configuration" request.

• #define A2DP\_EVT\_OPEN\_STREAM\_COMPLETED AVDTP\_EVT\_OPEN\_STREAM\_COMPLETED

This event is generated when a local device received a response (either positive or negative) to a "open stream" request.

#define A2DP\_EVT\_START\_STREAM\_COMPLETED AVDTP\_EVT\_START\_STREAM\_COMPLETED

This event is generated when a local device received a response (either positive or negative) to a "start stream" request.

• #define A2DP\_EVT\_CLOSE\_STREAM\_COMPLETED AVDTP\_EVT\_CLOSE\_STREAM\_COMPLETED

This event is generated when a local device received a response (either positive or negative) to a "close stream" request.

This event is generated when a local device received a response (either positive or negative) to a "suspend stream" request.

#define A2DP\_EVT\_STREAM\_SECURITY\_CONTROL\_COMPLETED AVDTP\_EVT\_STREAM\_SECURI

 TY CONTROL COMPLETED

This event is generated when a local device received a response (either positive or negative) to a "exchange content protection control data" request.

#define A2DP EVT ABORT STREAM COMPLETED AVDTP EVT ABORT STREAM COMPLETED

This event is generated when a local device received a response (either positive or negative) to a "abort stream" request.

• #define A2DP\_EVT\_SEP\_INFO\_RECEIVED AVDTP\_EVT\_SEP\_INFO\_RECEIVED

This event is generated for each SEP contained in a positive response to a "discover" request.

• #define A2DP EVT SEP CAPABILITIES RECEIVED AVDTP EVT SEP CAPABILITIES RECEIVED

This event is generated when a local device received a positive response to a "get SEP capabilities" request.

#define A2DP\_EVT\_STREAM\_CONFIGURATION\_RECEIVED AVDTP\_EVT\_STREAM\_CONFIGURATI
 ON RECEIVED

This event is generated when a local device received a positive response to a "get stream configuration" request.

#define A2DP\_EVT\_SET\_STREAM\_CONFIGURATION\_REQUESTED AVDTP\_EVT\_SET\_STREAM\_C
 ONFIGURATION\_REQUESTED

This event is generated when a local device received "set stream configuration" request.

• #define A2DP EVT OPEN STREAM REQUESTED AVDTP EVT OPEN STREAM REQUESTED

This event is generated when a local device received "open stream" request.

#define A2DP EVT START STREAM REQUESTED AVDTP EVT START STREAM REQUESTED

This event is generated when a local device received "start stream" request.

#define A2DP\_EVT\_CLOSE\_STREAM\_REQUESTED AVDTP\_EVT\_CLOSE\_STREAM\_REQUESTED

This event is generated when a local device received "close stream" request.

• #define A2DP\_EVT\_SUSPEND\_STREAM\_REQUESTED AVDTP\_EVT\_SUSPEND\_STREAM\_REQUES↔ TED

This event is generated when a local device received "suspend stream" request.

• #define A2DP EVT ABORT STREAM REQUESTED AVDTP EVT ABORT STREAM REQUESTED

This event is generated when a local device received "abort stream" request.

#define A2DP\_EVT\_RECONFIGURE\_STREAM\_REQUESTED AVDTP\_EVT\_RECONFIGURE\_STREAM 
 REQUESTED

This event is generated when a local device received "change stream configuration" request.

#define A2DP EVT MEDIA PACKET RECEIVED AVDTP EVT MEDIA PACKET RECEIVED

This event is generated when a local device received a media packet.

#define A2DP EVT STREAM CONFIGURED AVDTP EVT STREAM CONFIGURED

This event is generated when a local device has successfully configured a stream.

#define A2DP\_EVT\_STREAM\_RECONFIGURED AVDTP\_EVT\_STREAM\_RECONFIGURED

This event is generated when a local device has successfully reconfigured a stream.

#define A2DP\_EVT\_STREAM\_OPENED AVDTP\_EVT\_STREAM\_OPENED

This event is generated when a local device has successfully opened a stream.

• #define A2DP\_EVT\_STREAM\_STARTED AVDTP\_EVT\_STREAM\_STARTED

This event is generated when a local device has successfully started a stream.

#define A2DP\_EVT\_STREAM\_CLOSED AVDTP\_EVT\_STREAM\_CLOSED

This event is generated when a local device has successfully closed a stream.

• #define A2DP\_EVT\_STREAM\_SUSPENDED AVDTP\_EVT\_STREAM\_SUSPENDED

This event is generated when a local device has successfully suspended a stream.

• #define A2DP\_EVT\_STREAM\_ABORTED AVDTP\_EVT\_STREAM\_ABORTED

This event is generated when a local device has successfully aborted a stream.

#define A2DP\_EVT\_MEDIA\_PACKET\_SENT AVDTP\_EVT\_MEDIA\_PACKET\_SENT

This event is generated when a local device sent a media packet.

#define A2DP\_EVT\_MEDIA\_PACKET\_SEND\_FAILED AVDTP\_EVT\_MEDIA\_PACKET\_SEND\_FAILED

This event is generated when a local device failed to send a media packet.

• #define A2DP\_EVT\_OPEN\_AND\_START\_STREAM\_COMPLETED (AVDTP\_EVT\_LAST + 1)

This event is generated when a local device completed "open and start" request.

# 3.13.1 Detailed Description

The Advanced Audio Distribution Profile (A2DP) defines the protocols and procedures that realize distribution of audio content of high-quality in mono or stereo on ACL channels.

# 3.13.2 Macro Definition Documentation

# 3.13.2.1 #define A2DP\_EVT\_STREAM\_ABORTED AVDTP\_EVT\_STREAM\_ABORTED

This event is generated when a local device has successfully aborted a stream.

This event follows the A2DP\_EVT\_ABORT\_STREAM\_REQUESTED if the upper layer has accepted it. This event is not generated if stream abortion was initiated by the local device.

# 3.13.2.2 #define A2DP\_EVT\_STREAM\_CLOSED AVDTP\_EVT\_STREAM\_CLOSED

This event is generated when a local device has successfully closed a stream.

This event follows the A2DP\_EVT\_CLOSE\_STREAM\_REQUESTED if the upper layer has accepted it. This event is not generated if stream closing was initiated by the local device.

### 3.13.2.3 #define A2DP\_EVT\_STREAM\_CONFIGURED AVDTP\_EVT\_STREAM\_CONFIGURED

This event is generated when a local device has successfully configured a stream.

This event follows the A2DP\_EVT\_SET\_STREAM\_CONFIGURATION\_REQUESTED if the upper layer has accepted it. This event is not generated if stream configuration was initiated by the local device.

# 3.13.2.4 #define A2DP\_EVT\_STREAM\_OPENED AVDTP\_EVT\_STREAM\_OPENED

This event is generated when a local device has successfully opened a stream.

This event follows the A2DP\_EVT\_OPEN\_STREAM\_REQUESTED if the upper layer has accepted it. This event is not generated if stream opening was initiated by the local device.

# 3.13.2.5 #define A2DP\_EVT\_STREAM\_RECONFIGURED AVDTP\_EVT\_STREAM\_RECONFIGURED

This event is generated when a local device has successfully reconfigured a stream.

This event follows the A2DP\_EVT\_RECONFIGURE\_STREAM\_REQUESTED if the upper layer has accepted it. This event is not generated if stream reconfiguration was initiated by the local device.

# 3.13.2.6 #define A2DP\_EVT\_STREAM\_STARTED AVDTP\_EVT\_STREAM\_STARTED

This event is generated when a local device has successfully started a stream.

This event follows the A2DP\_EVT\_START\_STREAM\_REQUESTED if the upper layer has accepted it. This event is not generated if stream starting was initiated by the local device.

# 3.13.2.7 #define A2DP\_EVT\_STREAM\_SUSPENDED AVDTP\_EVT\_STREAM\_SUSPENDED

This event is generated when a local device has successfully suspended a stream.

This event follows the A2DP\_EVT\_SUSPEND\_STREAM\_REQUESTED if the upper layer has accepted it. This event is not generated if stream suspension was initiated by the local device.

3.13.2.8 #define bt\_a2dp\_abort\_stream( mgr, strm\_handle ) bt\_avdtp\_abort\_stream(mgr->avdtp\_mgr, strm\_handle)

Suspend a stream.

This function tries to suspend a stream by sending a request to the remote party. The stream can be in any state state except AVDTP\_STREAM\_STATE\_IDLE. As a result of this operation the A2DP\_EVT\_ABORT\_STREAM\_ COMPLETED event will be generated. This operation cannot be rejected. The evt\_param.abort\_stream requested.err\_code is always == AVDTP\_ERROR\_SUCCESS.

#### **Parameters**

mgr	A2DP manager.
strm_handle	Stream handle.

#### Returns

- TRUE if the function succeeds, i.e. the actual request has been sent to the remote party.
- FALSE otherwise. No events will be generated.

3.13.2.9 #define bt\_a2dp\_add\_media\_rx\_buffer( *mgr*, *strm\_handle*, *buffer* ) bt\_avdtp\_add\_media\_rx\_buffer(mgr->avdtp\_mgr, strm\_handle, buffer)

Add a media packet buffer to a receive queue.

The consumer of A2DP is responsible for allocating and supplying A2DP with buffers used to store received packets. A2DP itself only has a queue for storing pointers to buffers supplied by the consumer. When a packet comes in A2DP finds the first buffer large enough to hold the received packet, copies the packet to the buffer and generates a A2DP\_EVT\_MEDIA\_PACKET\_RECEIVED event. The consumer then has to process the data in the buffer and return it back to the queue. If there is no buffers in the queue or none of the buffers is large enough the received packets is dropped. Each buffer has a field (data\_len) that holds the length of the received buffer. This field is never 0 if the buffer contains a packet. If a channel closed regardless of what has caused that and there are still buffers in the queue A2DP generates a A2DP\_EVT\_MEDIA\_PACKET\_RECEIVED event for each buffer and sets the data\_len to 0. This is to inform the A2DP consumer that the buffer has not been used and can be, for example, deallocated. This function adds a buffer to the receive queue.

### **Parameters**

mgr	A2DP manager.
strm_handle	Stream handle.
buffer	Pointer to a structure that holds the buffer and its parameters.

### Returns

- TRUE if the function succeeds.
- FALSE otherwise. The function fails only if a stream specified by the strm\_handle parameter
- does not exist. The stream can be in any state to call this function.

3.13.2.10 #define bt\_a2dp\_add\_media\_tx\_buffer( *mgr, strm\_handle, buffer* ) bt\_avdtp\_add\_media\_tx\_buffer(mgr->avdtp\_mgr, strm\_handle, buffer)

Add a media packet buffer to a send queue.

When the consumer of A2DP wants to send a packet to a remote device it calls this function. The function adds the packet to a queue and tells A2DP that it has something to send. The packet will be send as soon as the stream goes to A2DP\_STREAM\_STATE\_STREAMING state. When the packet has been successfully sent a A2DP\_EVT — \_\_MEDIA\_PACKET\_SENT is generated. Otherwise a A2DP\_EVT\_MEDIA\_PACKET\_SEND\_FAILED is generated. Regardless of the event generated the consumer can re-use the buffer as A2DP has removed it from the queue and gave up control over it. As in the case of received buffers, if a channel closed regardless of what has caused that and there are still buffers in the queue A2DP generates a A2DP\_EVT\_MEDIA\_PACKET\_SENT event for each buffer and sets the data\_len field to 0. This is to inform the AVDTP consumer that the buffer has not been used and can be, for example, deallocated.

### **Parameters**

mgr	A2DP manager.
strm_handle	Stream handle.
buffer	Pointer to a structure that holds the buffer and its parameters.

### Returns

- TRUE if the function succeeds.
- FALSE otherwise. The function fails only if a stream specified by the strm\_handle parameter
- · does not exist. The stream can be in any state to call this function.

3.13.2.11 #define bt\_a2dp\_cancel\_listen( *mgr*, *strm\_handle*, *sep\_id* ) bt\_avdtp\_cancel\_listen(mgr->avdtp\_mgr, strm\_handle, sep\_id)

Cancel listening for incoming connections.

This function removes a SEP from a list of SEPS which a stream can use for incoming requests.

### **Parameters**

mgr	A2DP manager.
strm_handle	Stream handle.
sep_id	Local SEP ID.

### Returns

- TRUE if the function succeeds.
- FALSE otherwise.

3.13.2.12 #define bt\_a2dp\_close\_stream( mgr, strm\_handle ) bt\_avdtp\_close\_stream(mgr->avdtp\_mgr, strm\_handle)

Close a stream.

This function tries to close a stream by sending a request to the remote party. The stream has to be in AVD $\leftarrow$  TP\_STREAM\_STATE\_OPEN or AVDTP\_STREAM\_STATE\_STREAMING state. As a result of this operation the A2DP\_EVT\_CLOSE\_STREAM\_COMPLETED event will be generated. If the stream has been closed the evt $\leftarrow$  \_param.bt\_avdtp\_evt\_close\_stream\_completed\_t.err\_code == AVDTP\_ERROR\_SUCCES  $\leftarrow$  S. Otherwise, if the remote device for any reason cannot or does not wish to close the stream, the evt\_param.  $\leftarrow$  bt\_avdtp\_evt\_close\_stream\_completed\_t.err\_code == the error code sent by the remote.

# **Parameters**

mgr	A2DP manager.
strm_handle	Stream handle.

# Returns

- TRUE if the function succeeds, i.e. the actual request has been sent to the remote party.
- $\bullet$  FALSE otherwise. No events will be generated.

3.13.2.13 #define bt\_a2dp\_connect( mgr, remote\_addr) bt\_avdtp\_connect(mgr->avdtp\_mgr, remote\_addr)

Connect to a remote device.

This function opens a control channel connection to a remote device specified by the remote\_addr. If connection cannot be initiated for some reason, for example, there is not enough resources, it returns FALSE and not events are

generated. Otherwise the result of an attempt to connect to the remote device is reported via the AVDTP callback. The events generated will either be A2DP\_EVT\_CTRL\_CHANNEL\_CONNECTED or A2DP\_EVT\_CTRL\_CHAN← NEL\_CONNECTION\_FAILED.

### **Parameters**

	mgr	A2DP manager.
re	mote_addr	The address of a remote device.

### Returns

- TRUE if connection establishment has been started.
- FALSE otherwise.

3.13.2.14 #define bt\_a2dp\_connect\_ex( mgr, remote\_addr, acl\_config ) bt\_avdtp\_connect\_ex(mgr->avdtp\_mgr, remote\_addr, acl\_config)

Connect to a remote device.

This function opens a control channel connection to a remote device specified by the remote\_addr. If connection cannot be initiated for some reason, for example, there is not enough resources, it returns FALSE and not events are generated. Otherwise the result of an attempt to connect to the remote device is reported via the AVDTP callback. The events generated will either be A2DP\_EVT\_CTRL\_CHANNEL\_CONNECTED or A2DP\_EVT\_CTRL\_CHAN⊷ NEL\_CONNECTION\_FAILED.

### **Parameters**

A2DP manager.
The address of a remote device.
ACL link configuration. This can be a combination of the following values:
HCI_CONFIG_ENABLE_AUTHENTICATION
HCI_CONFIG_ENABLE_ENCRYPTION
HCI_CONFIG_BECOME_MASTER

# Returns

- TRUE if connection establishment has been started.
- FALSE otherwise.

3.13.2.15 #define bt\_a2dp\_create\_stream( mgr ) bt\_avdtp\_create\_stream(mgr->avdtp\_mgr)

# Create a stream.

This function allocates memory for storing stream's data and assigns a stream handle. The stream handle is used to manipulate the stream - open, close, configure, suspend, abort.

# **Parameters**

mgr A2DP manager.	
-------------------	--

# Returns

- Stream handle if the function succeeds.
- 0 otherwise.

3.13.2.16 #define bt\_a2dp\_destroy\_stream( *mgr, strm\_handle* ) bt\_avdtp\_destroy\_stream(mgr->avdtp\_mgr, strm\_handle)

# Destroy a stream.

This function frees memory used by the stream. The stream has to exist and be in the "idle" state for this function to succeed. I.e. the stream has to be closed or aborted before this function can be called.

#### **Parameters**

mgr	A2DP manager.
strm_handle	Stream handle.

#### Returns

- · TRUE if the function succeeds.
- FALSE otherwise.

3.13.2.17 #define bt\_a2dp\_disconnect( mgr, remote\_addr) bt\_avdtp\_disconnect(mgr->avdtp\_mgr, remote\_addr)

Disconnect from a remote device.

This function closes a control and transport channels on all streams associated with the remote device specified by the remote\_addr. As a result of this operation the following events will be generated:

- A2DP\_EVT\_MEDIA\_PACKET\_RECEIVED: if a stream's receive queue is not empty this event is generated for each buffer with bt\_media\_packet\_t::data\_len set to 0
- A2DP\_EVT\_MEDIA\_PACKET\_SENT: if a stream's send queue is not empty this event is generated for each buffer with bt\_media\_packet\_t::data\_len set to 0
- A2DP\_EVT\_STREAM\_CLOSED: this event is generate if a stream is in "closing" state as a result of a request from the remote device or bt a2dp close stream call before bt a2dp disconnect call.
- A2DP\_EVT\_STREAM\_ABORTED: this event is generated if a stream is in "active" state at the time of bt\_

   avdtp\_disconnect call.

## Parameters

mgr	A2DP manager.
remote_addr	The address of a remote device.

## Returns

- · TRUE if disconnection has been started.
- FALSE otherwise. No events will be generated.

3.13.2.18 #define bt\_a2dp\_discover( mgr, remote\_addr ) bt\_avdtp\_discover(mgr->avdtp\_mgr, remote\_addr)

Discover SEPs on a remote device.

This function asks the remote device to send a list of all available SEPs. As a result of this operation the following events will be generated:

- A2DP\_EVT\_SEP\_INFO\_RECEIVED: this event is generated for every SEP received from the remote device. the evt\_param.sep\_info\_received contains SEP information.
- A2DP\_EVT\_DISCOVER\_COMPLETED: this event is generated after the last A2DP\_EVT\_SEP\_INFO\_R 
  ECEIVED if the remote accepted the request and the evt\_param.discover\_completed.err\_←

  code == AVDTP\_ERROR\_SUCCESS. if the remote rejected the request the evt\_param.discover←

  \_completed.err\_code == the error code sent by the remote.

### **Parameters**

	mgr	A2DP manager.
remo	te_addr	The address of a remote device.

### Returns

- TRUE if discover request has been sent.
- FALSE otherwise. No events will be generated.

3.13.2.19 #define bt\_a2dp\_find\_codec( mgr, codec\_type ) bt\_avdtp\_find\_codec(mgr->avdtp\_mgr, codec\_type)

# Find a codec.

A2DP in theory can support any type of codec. Each codec uses its own format for exchanging capabilities and configuration information. In order to make our implementation do not care about these formats we use a simple way of telling AVDTP how to parse and serialize codec's configuration. The consumer of A2DP has to register a callback function (one per codec type) for each codec it wishes to support. That callback has to perform two function. The first one is to read the configuration received from the remote device and store it in a structure defined by the consumer. The second one is to serialize the data from a structure to a format (in case of standard A2DP codecs the format is defined in A2DP specification, vendor specific codecs can define their own formats) suitable for sending as a part of a AVDTP request. This function returns a pointer to a structure that holds a pointer to a codec's callback function.

#### **Parameters**

mgr	A2DP manager.
codec_type	Codec type. The codec_type can be one of the following values:
	AVDTP_CODEC_TYPE_SBC: SBC
	<ul> <li>AVDTP_CODEC_TYPE_MPEG1_2_AUDIO: MPEG-1,2 (used in MP3 files)</li> </ul>
	<ul> <li>AVDTP_CODEC_TYPE_MPEG2_4_AAC: MPEG-2,4 AAC (used in Apple products)</li> </ul>
	<ul> <li>AVDTP_CODEC_TYPE_ATRAC: ATRAC (used in Sony products)</li> </ul>
	<ul> <li>AVDTP_CODEC_TYPE_NON_A2DP: Non-A2DP Codec</li> </ul>

### Returns

- TRUE if the function succeeds.
- FALSE otherwise. The function fails if a callback for a codec type specified in the <code>codec</code> parameter
- has not been previously registered with bt\_avdtp\_register\_codec.

3.13.2.20 #define bt\_a2dp\_get\_all\_capabilities( mgr, remote\_addr, seid\_acp ) bt\_avdtp\_get\_all\_capabilities(mgr->avdtp\_mgr, remote\_addr, seid\_acp)

Get remote SEP capabilities.

This function asks the remote device to send capabilities of a SEP specified by the <code>seid\_acp</code>. As a result of this operation the following events will be generated:

- A2DP\_EVT\_SEP\_CAPABILITIES\_RECEIVED: this event is generated if the remote device accepted the request. the evt\_param.sep\_capabilities\_received contains SEP capabilities.
- A2DP\_EVT\_GET\_SEP\_CAPABILITIES\_COMPLETED: this event is generated right after A2DP\_EVT

  \_SEP\_CAPABILITIES\_RECEIVED if the remote accepted the request the evt\_param.get\_sep\_←
  capabilities\_completed.err\_code == AVDTP\_ERROR\_SUCCESS. if the remote rejected the

request the evt\_param.get\_sep\_capabilities\_completed.err\_code == the error code sent
by the remote.

### **Parameters**

mgr	A2DP manager.
remote_addr	The address of a remote device.
seid_acp	The ID of a remote SEP.

## Returns

- TRUE if discover request has been sent.
- FALSE otherwise. No events will be generated.

3.13.2.21 #define bt\_a2dp\_get\_capabilities( mgr, remote\_addr, seid\_acp ) bt\_avdtp\_get\_capabilities(mgr->avdtp\_mgr, remote\_addr, seid\_acp)

Get remote SEP capabilities.

This function asks the remote device to send capabilities of a SEP specified by the seid\_acp. As a result of this operation the following events will be generated:

- A2DP\_EVT\_SEP\_CAPABILITIES\_RECEIVED: this event is generated if the remote device accepted the request. the evt\_param.sep\_capabilities\_received contains SEP capabilities.
- A2DP\_EVT\_GET\_SEP\_CAPABILITIES\_COMPLETED: this event is generated right after A2DP\_EVT 
  \_\_SEP\_CAPABILITIES\_RECEIVED if the remote accepted the request the evt\_param.get\_sep\_←
  capabilities\_completed.err\_code == AVDTP\_ERROR\_SUCCESS. if the remote rejected the
  request the evt\_param.get\_sep\_capabilities\_completed.err\_code == the error code sent
  by the remote.

# **Parameters**

mgr	A2DP manager.
remote_addr	The address of a remote device.
seid_acp	The ID of a remote SEP.

# Returns

- · TRUE if discover request has been sent.
- FALSE otherwise. No events will be generated.

3.13.2.22 #define bt\_a2dp\_get\_hci\_connection( *mgr*, *strm\_handle* ) bt\_avdtp\_get\_hci\_connection(mgr->avdtp\_mgr, strm\_handle)

Get HCI connection for a stream.

This function returns a pointer to a structure that describes an HCl connection a stream is open on. The return value can be used to call various function from the HCl layer. For example, if an app wants to force disconnection from a remote device it can call bt hci disconnect.

mgr	A2DP manager.
strm_handle	Stream handle.

### Returns

- Pointer to a structure that describes an HCI connection if the function succeeds.
- NULL otherwise. The function fails only if a stream specified by the strm\_handle parameter
- does not exist or there is no HCI connection between local and remote devices associated with the stream.

#### Note

This function has not been implemented.

3.13.2.23 #define bt\_a2dp\_get\_stream\_codec\_config( *mgr, strm\_handle* ) bt\_avdtp\_get\_stream\_codec\_config(mgr->avdtp\_mgr, strm\_handle)

Get the configuration of the codec currently used with the stream.

This function returns a pointer to a structure that contains configuration of the codec currently used with the stream. The structure returned depends on the codec. The dotstack defines structures only for SBC, MPEG-1,2 and MP← EG-2,4 AAC codecs:

- SBC: bt\_a2dp\_sbc\_config\_t (defined in a2dp\_sbc\_codec.h)
- MPEG-1,2: bt\_a2dp\_mpeg\_config\_t (defined in a2dp\_mpeg\_codec.h)
- MPEG-2,4 AAC: bt a2dp aac config t (defined in a2dp aac codec.h)

# Parameters

mgr	A2DP manager.
strm_handle	Stream handle.

# Returns

 The codec's configuration if strm\_handle specifies a valid stream and the stream is in one of the following state:

```
AVDTP_STREAM_STATE_CONFIGURED
AVDTP_STREAM_STATE_OPEN
AVDTP_STREAM_STATE_STREAMING
```

\li NULL otherwise.

3.13.2.24 #define bt\_a2dp\_get\_stream\_codec\_type( mgr, strm\_handle ) bt\_avdtp\_get\_stream\_codec\_type(mgr->avdtp\_mgr, strm\_handle)

Get the type of the codec currently used with the stream.

This function returns the type of the codec currently used with the stream.

mgr	A2DP manager.
strm_handle	Stream handle.

#### Returns

 The type of the codec if strm\_handle specifies a valid stream and the stream is in one of the following states:

```
AVDTP_STREAM_STATE_CONFIGURED
AVDTP_STREAM_STATE_OPEN
AVDTP_STREAM_STATE_STREAMING

@arg The result will be one of the following values:

AVDTP_CODEC_TYPE_SBC: SBC

AVDTP_CODEC_TYPE_MPEG1_2_AUDIO: MPEG-1,2 (used in MP3 files)
AVDTP_CODEC_TYPE_MPEG2_4_AAC: MPEG-2,4 AAC (used in Apple products)
AVDTP_CODEC_TYPE_ATRAC: ATRAC (used in Sony products)
AVDTP_CODEC_TYPE_NON_A2DP: Non-A2DP Codec

@arg OxFF otherwise.
```

3.13.2.25 #define bt\_a2dp\_get\_stream\_local\_sep\_id( *mgr, strm\_handle* ) bt\_avdtp\_get\_stream\_local\_sep\_id(mgr->avdtp\_mgr, strm\_handle)

Get stream's local SEP ID.

This function returns the ID of the local SEP associated with the stream.

#### **Parameters**

mgr	A2DP manager.
strm_handle	Stream handle.

### Returns

- The ID of the local SEP if strm\_handle specifies a valid stream.
- · 0 otherwise.
- 3.13.2.26 #define bt\_a2dp\_get\_stream\_remote\_address( *mgr*, *strm\_handle* ) bt\_avdtp\_get\_stream\_remote\_← address(mgr->avdtp\_mgr, strm\_handle)

Get stream's remote BT address.

This function returns the address of the remote device associated with the stream.

### **Parameters**

mgr	A2DP manager.
strm_handle	Stream handle.

### Returns

- The address of the remote device if strm\_handle specifies a valid stream.
- · NULL otherwise.
- 3.13.2.27 #define bt\_a2dp\_get\_stream\_remote\_sep\_id( *mgr, strm\_handle* ) bt\_avdtp\_get\_stream\_remote\_sep\_← id(mgr->avdtp\_mgr, strm\_handle)

Get stream's remote SEP ID.

This function returns the ID of the remote SEP associated with the stream.

### **Parameters**

mgr	A2DP manager.
strm_handle	Stream handle.

### Returns

- The ID of the remote SEP if strm handle specifies a valid stream.
- · 0 otherwise.

3.13.2.28 #define bt\_a2dp\_get\_stream\_state( mgr, strm\_handle ) bt\_avdtp\_get\_stream\_state(mgr->avdtp\_mgr, strm\_handle)

Get local stream state.

This function returns local state of a stream specified by the strm\_handle. No request is sent to the remote party.

#### **Parameters**

mgr	A2DP manager.
strm_handle	Stream handle.

#### Returns

The state of the stream. The result will be one of the following values:

- AVDTP\_STREAM\_STATE\_IDLE: The stream is idle. This can mean two things. The stream specified by strm\_handle does not exist or the stream is closed.
- AVDTP\_STREAM\_OPENING\_TRANSPORT\_CHANNELS: The stream is opening transport channels.
- AVDTP\_STREAM\_CLOSING\_TRANSPORT\_CHANNELS: The stream is closing transport channels.
- AVDTP STREAM STATE CONFIGURED: The stream has been configured.
- · AVDTP STREAM STATE OPEN: The stream has been opened.
- AVDTP\_STREAM\_STATE\_STREAMING: The stream has been started. Depending on the local SEP type (source or sink) it means that the stream is can send or receive media packets.
- AVDTP\_STREAM\_STATE\_CLOSING: The stream is closing. This means that all transport channels
  associated with the stream are being closed. After they have been closed the stream goes to AVDTP

  \_STREAM\_STATE\_IDLE state.
- AVDTP\_STREAM\_STATE\_ABORTING: The stream is aborting. This means that all transport channels
  associated with the stream are being closed. After they have been closed the stream goes to AVDTP

  \_STREAM\_STATE\_IDLE state.

3.13.2.29 #define bt\_a2dp\_listen( mgr, strm\_handle, sep\_id) bt avdtp\_listen(mgr->avdtp\_mgr, strm\_handle, sep\_id)

Listen for incoming connections.

This function tells a stream that it can use a particular SEP to accept incoming requests to open it. The SEP can be associated with multiple streams but used with only one. The stream has to be closed before the SEP can be used with another stream. For outgoing connections this is not needed. Any SEP can be used with any stream given that the SEP is not already in use by another stream.

Para	meter	S
------	-------	---

mgr	A2DP manager.
strm_handle	Stream handle.
sep_id	Local SEP ID.

### Returns

- TRUE if the function succeeds.
- FALSE otherwise.

3.13.2.30 #define bt\_a2dp\_reconfigure\_stream( *mgr, strm\_handle, caps* ) bt\_avdtp\_reconfigure\_stream(mgr->avdtp\_mgr, strm\_handle, caps)

# Reconfigure stream.

This function tries to change the stream's configuration. For this function to succeed the stream has to be open. As a result of this operation the A2DP\_EVT\_STREAM\_RECONFIGURE\_COMPLETED event will be generated. If reconfiguration was a success the evt\_param.stream\_reconfigure\_completed.err\_code ==  $A \leftarrow VDTP_ERROR_SUCCESS$ . Otherwise the evt\_param.stream\_reconfigure\_completed.err\_code == the error code sent by the remote.

#### **Parameters**

mgr	A2DP manager.
strm_handle	Stream handle.
caps	New stream configuration.

# Returns

- TRUE if the function succeeds, i.e. the actual request has been sent to the remote party.
- FALSE otherwise. No events will be generated.

3.13.2.31 #define bt\_a2dp\_register\_sink( *mgr, caps* ) bt\_avdtp\_register\_sep(mgr->avdtp\_mgr, AVDTP\_SEP\_TYPE\_SINK, caps)

Register a Sink SEP with the local A2DP manager.

This function is used to add a sink SEP to a list of SEPs supported by the local A2DP entity.

### **Parameters**

mgr	A2DP manager.
caps	The capabilities of a SEP.

### Returns

- ID of a SEP if the function succeeds.
- FALSE otherwise.

3.13.2.32 #define bt\_a2dp\_register\_source( mgr, caps) bt\_avdtp\_register\_sep(mgr->avdtp\_mgr, AVDTP\_SEP\_TYPE\_SOURCE, caps)

Register a Source SEP with the local A2DP manager.

This function is used to add a source SEP to a list of SEPs supported by the local A2DP entity.

#### **Parameters**

mgr	A2DP manager.
caps	The capabilities of a SEP.

#### Returns

- ID of a SEP if the function succeeds.
- FALSE otherwise.

3.13.2.33 #define bt\_a2dp\_remove\_media\_rx\_buffer( mgr, strm\_handle, buffer) bt\_avdtp\_remove\_media\_rx\_
buffer(mgr->avdtp\_mgr, strm\_handle, buffer)

Remove a media packet buffer from a receive queue.

The consumer of A2DP is responsible for allocating and supplying A2DP with buffers used to store received packets. A2DP itself only has a queue for storing pointers to buffers supplied by the consumer. When a packet comes in A2DP finds the first buffer large enough to hold the received packet, copies the packet to the buffer and generates a A2DP\_EVT\_MEDIA\_PACKET\_RECEIVED event. The consumer then has to process the data in the buffer and return it back to the queue. If there is no buffers in the queue or none of the buffers is large enough the received packets is dropped. Each buffer has a field (data\_len) that holds the length of the received buffer. This field is never 0 if the buffer contains a packet. If a channel closed regardless of what has caused that and there are still buffers in the queue A2DP generates a A2DP\_EVT\_MEDIA\_PACKET\_RECEIVED event for each buffer and sets the data\_len to 0. This is to inform the A2DP consumer that the buffer has not been used and can be, for example, deallocated. This function removes a buffer from the receive queue.

#### **Parameters**

mgr	A2DP manager.
strm_handle	Stream handle.
buffer	Pointer to a structure that holds the buffer and its parameters.

# Returns

- TRUE if the function succeeds.
- FALSE otherwise. The function fails only if a stream specified by the strm\_handle parameter
- does not exist. The stream can be in any state to call this function.

3.13.2.34 #define bt\_a2dp\_remove\_media\_tx\_buffer( *mgr, strm\_handle, buffer* ) bt\_avdtp\_remove\_media\_tx\_
buffer(mgr->avdtp\_mgr, strm\_handle, buffer)

Remove a media packet buffer from a send queue.

When the consumer of A2DP wants to send a packet to a remote device it calls bt\_avdtp\_add\_media\_tx\_buffer function. The function adds the packet to a queue and tells A2DP that it has something to send. The packet will be send as soon as the stream goes to A2DP\_STREAM\_STATE\_STREAMING state. The consumer has a chance to remove a packet from the queue before it has been sent to a remote device by calling bt\_a2dp\_remove\_media\_ct buffer.

mgr	A2DP manager.
strm_handle	Stream handle.

1 " B': : : : : : : : : : : : : : : : : : :	
buffer   Pointer to a structure that holds the buffer and its parameters.	
banci   i dilita to a structure that holds the baller and its parameters.	

#### Returns

- TRUE if the function succeeds.
- FALSE otherwise. The function fails only if a stream specified by the strm\_handle parameter
- does not exist. The stream can be in any state to call this function.

### Note

This function has not been implemented.

3.13.2.35 #define bt\_a2dp\_start\_stream( mgr, strm\_handle ) bt\_avdtp\_start\_stream(mgr->avdtp\_mgr, strm\_handle)

#### Start a stream.

This function tries to start a stream by sending a request to the remote party. The stream has to be in AVDTP\_S $\leftarrow$  TREAM\_STATE\_OPEN state. The stream goes to this state as a result of successful configuration or suspension (both can be initiated by either party). As a result of this operation the A2DP\_EVT\_START\_STREAM\_COMPLE $\leftarrow$  TED event will be generated. If the stream has been open the evt\_param.start\_stream\_requested.  $\leftarrow$  err\_code == AVDTP\_ERROR\_SUCCESS. Otherwise, if the remote device for any reason cannot or does not wish to start the stream, the evt\_param.start\_stream\_requested.err\_code == the error code sent by the remote.

### **Parameters**

mgr	A2DP manager.
strm_handle	Stream handle.

### Returns

- TRUE if the function succeeds, i.e. the actual request has been sent to the remote party.
- FALSE otherwise. No events will be generated.

3.13.2.36 #define bt\_a2dp\_suspend\_stream( *mgr, strm\_handle* ) bt\_avdtp\_suspend\_stream(mgr->avdtp\_mgr, strm\_handle)

# Suspend a stream.

This function tries to suspend a stream by sending a request to the remote party. The stream has to be in AVD  $\leftarrow$  TP\_STREAM\_STATE\_STREAMING state. As a result of this operation the A2DP\_EVT\_SUSPEND\_STREAM\_  $\leftarrow$  COMPLETED event will be generated. If the stream has been suspended the evt\_param.bt\_avdtp\_evt  $\leftarrow$  \_suspend\_stream\_requested\_t.err\_code == AVDTP\_ERROR\_SUCCESS. Otherwise, if the remote device for any reason cannot or does not wish to suspend the stream, the evt\_param.bt\_avdtp\_evt\_  $\leftarrow$  suspend\_stream\_requested\_t.err\_code == the error code sent by the remote.

### **Parameters**

mgr	A2DP manager.
strm_handle	Stream handle.

### Returns

- TRUE if the function succeeds, i.e. the actual request has been sent to the remote party.
- FALSE otherwise. No events will be generated.

3.13.2.37 #define bt\_avdtp\_get\_stream\_config( mgr, strm\_handle ) bt\_avdtp\_get\_stream\_config(mgr->avdtp\_mgr, strm\_handle)

Get stream's configuration.

This function returns a pointer to a structure holding current configuration of the stream.

### **Parameters**

mgr	A2DP manager.
strm_handle	Stream handle.

#### Returns

The stream's configuration if strm\_handle specifies a valid stream and the stream is in one of the following state:

AVDTP\_STREAM\_STATE\_CONFIGURED AVDTP\_STREAM\_STATE\_OPEN AVDTP\_STREAM\_STATE\_STREAMING

\li NULL otherwise.

# 3.13.3 Typedef Documentation

3.13.3.1 typedef void(\* bt\_a2dp\_find\_server\_callback\_fp) (bt\_uint supported\_features, bt\_bool found, void \*param)

Notify the application of the result of searching for a remote A2DP entity (sourse or sink)

This function is called by the A2DP layer when searching for an A2DP entity on a remote device has completed.

# **Parameters**

supported_←	Features supported by a remote A2DP entity.
features	
found	TRUE if an A2DP entity has been found on the remote device. FALSE otherwise.
param	pointer to arbitrary data passed to the bt_a2dp_find_source() or bt_a2dp_find_sink function
	through its callback_param parameter.

3.13.3.2 typedef void(\* bt\_a2dp\_mgr\_callback\_fp) (bt\_a2dp\_mgr\_t \*mgr, bt\_byte evt, bt\_a2dp\_event\_t \*evt\_param, void \*callback\_param)

# A2DP application callback.

In order to be notified of various events a consumer of the A2DP layer has to register a callback function. The stack will call that function whenever a new event has been generated.

# **Parameters**

mgr	A2DP manager.
evt	A2DP event. The event can be one of the following values:
	A2DP_EVT_CTRL_CHANNEL_CONNECTED: Control channel connected.
	A2DP_EVT_CTRL_CHANNEL_DISCONNECTED: Control channel disconnected.
	A2DP_EVT_CTRL_CONNECTION_FAILED: Control channel connection failed (generated only if control connection has been initiated by the local device).
	A2DP_EVT_DISCOVER_COMPLETED: Local device completed discovering remote SEPs.
	<ul> <li>A2DP_EVT_GET_SEP_CAPABILITIES_COMPLETED: Local device received a response to Get SEP capabilities operation.</li> </ul>
	A2DP_EVT_SET_STREAM_CONFIGURATION_COMPLETED: Local device received a response to Set stream configuration operation.
	A2DP_EVT_GET_STREAM_CONFIGURATION_COMPLETED: Local device received a response to Get stream configuration operation.
	A2DP_EVT_STREAM_RECONFIGURE_COMPLETED: Local device received a response to Reconfigure stream operation.
	A2DP_EVT_OPEN_STREAM_COMPLETED: Local device received a response to Open stream operation.
	A2DP_EVT_START_STREAM_COMPLETED: Local device received a response to Start stream operation.
	A2DP_EVT_CLOSE_STREAM_COMPLETED: Local device received a response to Close stream operation.
	A2DP_EVT_SUSPEND_STREAM_COMPLETED: Local device received a response to Suspend stream operation.
	<ul> <li>A2DP_EVT_STREAM_SECURITY_CONTROL_COMPLETED: Local device received a response to Stream security control operation.</li> </ul>
	A2DP_EVT_ABORT_STREAM_COMPLETED: Local device received a response to Abort stream operation.
	A2DP_EVT_SEP_INFO_RECEIVED: SEP information received.
	A2DP_EVT_SEP_CAPABILITIES_RECEIVED: SEP capabilities received.
	A2DP_EVT_STREAM_CONFIGURATION_RECEIVED: Stream configuration received.
	A2DP_EVT_SET_STREAM_CONFIGURATION_REQUESTED: Remote device requested stream configuration.
	A2DP_EVT_OPEN_STREAM_REQUESTED: Remote device requested to open a stream.
	A2DP_EVT_START_STREAM_REQUESTED: Remote device requested to start a stream.
	A2DP_EVT_CLOSE_STREAM_REQUESTED: Remote device requested to close a stream.
	A2DP_EVT_SUSPEND_STREAM_REQUESTED: Remote device requested to suspend a stream.
	A2DP_EVT_ABORT_STREAM_REQUESTED: Remote device requested to abort a
Copyright © 2015 SEARAN	LLC. All right@asmved.
	A2DP_EVT_RECONFIGURE_STREAM_REQUESTED: Remote device requested to reconfigure a stream.

reconfigure a stream.

# 3.13.4 Function Documentation

3.13.4.1 bt\_bool bt\_a2dp\_find\_sink ( bt\_bdaddr\_t \* deviceAddress, bt\_a2dp\_find\_server\_callback\_fp callback, bt\_sdp\_client\_callback\_fp client\_callback, void \* callback\_param )

# Find sink.

This function looks for a sink on a remote device specified by deviceAddress and, if found, returns features supported by the sink.

### **Parameters**

deviceAddress	The address of a remote device.
callback	The callback function that will be called when search has completed.
client_callback	The optional callback function that an application can set if it wants to be notified of state changes of the SDP client. The evt parameter of the callback can be one of the following values:
	SDP_CLIENT_STATE_IDLE
	SDP_CLIENT_STATE_CONNECTING
	SDP_CLIENT_STATE_DISCONNECTING
	SDP_CLIENT_STATE_CONNECTED
callback_param	A pointer to arbitrary data to be passed to the callback and client_callback call-
	backs.

# Returns

- TRUE if the function succeeds.
- FALSE otherwise.

3.13.4.2 bt\_bool bt\_a2dp\_find\_source ( bt\_bdaddr\_t \* deviceAddress, bt\_a2dp\_find\_server\_callback\_fp callback, bt\_sdp\_client\_callback\_fp client\_callback, void \* callback\_param )

# Find source.

This function looks for a source on a remote device specified by deviceAddress and, if found, returns features supported by the source.

### **Parameters**

deviceAddress	The address of a remote device.
callback	The callback function that will be called when search has completed.
client_callback	The optional callback function that an application can set if it wants to be notified of state changes of the SDP client. The evt parameter of the callback can be one of the following values:  • SDP_CLIENT_STATE_IDLE  • SDP_CLIENT_STATE_CONNECTING  • SDP_CLIENT_STATE_DISCONNECTING
	SDP_CLIENT_STATE_CONNECTED
callback_param	A pointer to arbitrary data to be passed to the callback and client_callback call-
	backs.

### Returns

- TRUE if the function succeeds.
- · FALSE otherwise.

# 3.13.4.3 bt\_a2dp\_mgr\_t\* bt\_a2dp\_get\_mgr ( void )

Return a pointer to an instance of the A2DP manager.

This function returns a pointer to an instance of the A manager. There is only one instance of the manager allocated by the stack. The pointer is passed as the first parameter to all A2DP functions.

```
3.13.4.4 void bt_a2dp_init (void)
```

Initialize the A2DP layer.

This function initializes the A2DP layer of the stack. It must be called prior to any other A2DP function can be called.

3.13.4.5 bt\_bool bt\_a2dp\_open\_and\_start\_stream ( bt\_a2dp\_mgr\_t \* mgr, bt\_byte  $strm_handle$ , bt\_bdaddr\_t \* remote\_addr, bt\_byte  $seid_int$ , bt\_byte  $seid_acp$ , const bt\_avdtp\_sep\_capabilities\_t \* caps )

Open & start a stream.

Opening a stream involves sending 3 requests to a remote device - "set configuration", "open stream" and "start stream". Each event generates its own event which must be handled and acted accordingly by the application. To make the use of API easier dotstack combines all these requests in one request called "open & start stream". dotstack sends necessary requests in a proper sequence, handles responses and generates only one event (A2  $\leftarrow$  DP\_EVT\_OPEN\_AND\_START\_STREAM\_COMPLETED) at the end. If any of the individual requests has failed the event's parameter bt\_a2dp\_event\_t::open\_and\_start\_stream\_completed is populated with the error code and request id which caused it.

mgr	A2DP manager.
strm_handle	Stream handle.

remote_addr	The address of a remote device.
seid_int	Local SEP ID.
seid_acp	Remote SEP ID.
caps	Stream configuration.

### Returns

- TRUE if the function succeeds.
- FALSE otherwise. No events will be generated.

3.13.4.6 void bt\_a2dp\_register\_aac\_codec ( bt\_a2dp\_mgr\_t \* mgr )

# Register default AAC codec.

This function adds AAC codec implemented by dotstack to the list of known codecs. For more information about codecs see description of <a href="mailto:bt\_avdtp\_register\_codec">bt\_avdtp\_register\_codec</a>. The only codec A2DP is mandatory to support is SBC. All other codecs are optional. If an application wants to use AAC codec it must call this function when it is initializing.

#### Note

dotstack codecs do not do actual encoding/decoding. their function is to parse and serialize codec's configuration.

#### **Parameters**

mgr	A2DP manager.
-----	---------------

3.13.4.7 void bt\_a2dp\_register\_callback ( bt\_a2dp\_mgr\_t \* mgr, bt\_a2dp\_mgr\_callback\_fp callback, void \* callback\_param )

Register a A2DP application callback.

In order to be notified of various events a consumer of the A2DP layer has to register a callback function. The stack will call this function whenever a new event has been generated passing the code of the event as the second parameter. The event can be one of the following values:

- A2DP EVT CTRL CHANNEL CONNECTED: Control channel connected.
- A2DP\_EVT\_CTRL\_CHANNEL\_DISCONNECTED: Control channel disconnected.
- A2DP\_EVT\_CTRL\_CONNECTION\_FAILED: Control channel connection failed (generated only if control connection has been initiated by the local device).
- A2DP\_EVT\_DISCOVER\_COMPLETED: Local device completed discovering remote SEPs.
- A2DP\_EVT\_GET\_SEP\_CAPABILITIES\_COMPLETED: Local device received a response to Get SEP capabilities operation.
- A2DP\_EVT\_SET\_STREAM\_CONFIGURATION\_COMPLETED: Local device received a response to Set stream configuration operation.
- A2DP\_EVT\_GET\_STREAM\_CONFIGURATION\_COMPLETED: Local device received a response to Get stream configuration operation.
- A2DP\_EVT\_STREAM\_RECONFIGURE\_COMPLETED: Local device received a response to Reconfigure stream operation.
- A2DP\_EVT\_OPEN\_STREAM\_COMPLETED: Local device received a response to Open stream operation.
- A2DP\_EVT\_START\_STREAM\_COMPLETED: Local device received a response to Start stream operation.

- A2DP\_EVT\_CLOSE\_STREAM\_COMPLETED: Local device received a response to Close stream operation.
- A2DP\_EVT\_SUSPEND\_STREAM\_COMPLETED: Local device received a response to Suspend stream operation.
- A2DP\_EVT\_STREAM\_SECURITY\_CONTROL\_COMPLETED: Local device received a response to Stream security control operation.
- A2DP\_EVT\_ABORT\_STREAM\_COMPLETED: Local device received a response to Abort stream operation.
- · A2DP EVT SEP INFO RECEIVED: SEP information received.
- A2DP\_EVT\_SEP\_CAPABILITIES\_RECEIVED: SEP capabilities received.
- A2DP\_EVT\_STREAM\_CONFIGURATION\_RECEIVED: Stream configuration received.
- A2DP\_EVT\_SET\_STREAM\_CONFIGURATION\_REQUESTED: Remote device requested stream configuration
- A2DP EVT OPEN STREAM REQUESTED: Remote device requested to open a stream.
- A2DP\_EVT\_START\_STREAM\_REQUESTED: Remote device requested to start a stream.
- · A2DP EVT CLOSE STREAM REQUESTED: Remote device requested to close a stream.
- A2DP\_EVT\_SUSPEND\_STREAM\_REQUESTED: Remote device requested to suspend a stream.
- A2DP\_EVT\_ABORT\_STREAM\_REQUESTED: Remote device requested to abort a stream.
- · A2DP EVT RECONFIGURE STREAM REQUESTED: Remote device requested to reconfigure a stream.
- A2DP EVT MEDIA PACKET RECEIVED: Remote device sent a media packet.
- A2DP\_EVT\_STREAM\_CONFIGURED: A stream has been configured (This event is generated right after A2DP EVT SET STREAM CONFIGURATION REQUESTED if the local devices accepted the request).
- A2DP\_EVT\_STREAM\_RECONFIGURED: A stream has been re-configured (This event is generated right after A2DP\_EVT\_RECONFIGURE\_STREAM\_REQUESTED if the local devices accepted the request).
- A2DP\_EVT\_STREAM\_OPENED: A stream has been opened (This event is generated as a result of local or remote stream opening request).
- A2DP\_EVT\_STREAM\_CLOSED: A stream has been close (This event is generated right after A2DP\_EVT

   CLOSE STREAM REQUESTED if the local devices accepted the request).
- A2DP\_EVT\_STREAM\_SUSPENDED: A stream has been suspended (This event is generated right after A2DP\_EVT\_SUSPEND\_STREAM\_REQUESTED if the local devices accepted the request).
- A2DP\_EVT\_STREAM\_ABORTED: A stream has been aborted (This event is generated right after A2DP
   \_EVT\_SUSPEND\_STREAM\_REQUESTED if the local devices accepted the request. It is also generated if
   connection between devices has been terminated by means other than A2DP signaling, e.g. devices going
   out of range).
- A2DP\_EVT\_MEDIA\_PACKET\_SENT: The local device has successfully sent a media packet to the remote device.
- A2DP\_EVT\_MEDIA\_PACKET\_SEND\_FAILED: The local device was not able to send a media packet to the remote device.
- A2DP\_EVT\_OPEN\_AND\_START\_STREAM\_COMPLETED This event is generated when a local device completed "open and start" request.

### **Parameters**

ſ	mgr	AVDTP manager.
Ī	callback	The callback function that will be called when the AVDTP generates an event.
	callback_param	A pointer to arbitrary data to be passed to the callback callback.

3.13.4.8 void bt\_a2dp\_register\_mpeg\_codec ( bt\_a2dp\_mgr\_t \* mgr )

# Register default MPEG codec.

This function adds MPEG codec implemented by dotstack to the list of known codecs. For more information about codecs see description of <a href="mailto:bt\_avdtp\_register\_codec">bt\_avdtp\_register\_codec</a>. The only codec A2DP is mandatory to support is SBC. All other codecs are optional. If an application wants to use MPEG-1,2 codec it must call this function when it is initializing.

# Note

dotstack codecs do not do actual encoding/decoding. their function is to parse and serialize codec's configuration.

### **Parameters**

mgr	A2DP manager.

3.13.4.9 bt\_bool bt\_a2dp\_start ( bt\_a2dp\_mgr\_t \* mgr )

# Start the A2DP layer.

This function makes the A2DP layer ready to accept connection requests from remote device. To make an outgoing connection calling this function is not required.

## **Parameters**

mgr	AVDTP manager.

### Returns

- TRUE if the function succeeds.
- FALSE otherwise.

# 3.14 Audio/Video Control Transport Protocol (AVCTP)

AVCTP is the transport mechanisms used to exchange messages for controlling Audio and/or Video devices.

# **Modules**

Configuration

This module describes parameters used to configure AVCTP layer.

# **Data Structures**

```
    struct bt avctp evt channel connected t
```

Parameter to AVCTP\_EVT\_CHANNEL\_CONNECTED event.

struct bt\_avctp\_evt\_channel\_disconnected\_t

Parameter to AVCTP\_EVT\_CHANNEL\_DISCONNECTED event.

• struct bt\_avctp\_evt\_connection\_failed\_t

Parameter to AVCTP\_EVT\_CONNECTION\_FAILED event.

• struct bt\_avctp\_evt\_command\_received\_t

Parameter to AVCTP\_EVT\_COMMAND\_RECEIVED event.

• struct bt\_avctp\_evt\_response\_received\_t

Parameter to AVCTP\_EVT\_RESPONSE\_RECEIVED event.

struct bt\_avctp\_evt\_command\_sent\_t

Parameter to AVCTP\_EVT\_COMMAND\_SENT event.

struct bt\_avctp\_evt\_command\_cancelled\_t

Parameter to AVCTP\_EVT\_COMMAND\_CANCELLED event.

• struct bt\_avctp\_evt\_response\_sent\_t

Parameter to AVCTP\_EVT\_RESPONSE\_SENT event.

• struct bt\_avctp\_evt\_response\_cancelled\_t

Parameter to AVCTP\_EVT\_RESPONSE\_CANCELLED event.

• union bt\_avctp\_event\_t

Parameter to an application callback.

struct bt\_avctp\_message\_t

AVCTP message description.

struct bt\_avctp\_transport\_t

AVCTP transport description.

struct bt\_avctp\_channel\_t

AVCTP channel description.

struct bt\_avctp\_mgr\_t

AVCTP manager.

# **Typedefs**

typedef void(\* bt\_avctp\_mgr\_callback\_fp) (struct\_bt\_avctp\_mgr\_t \*mgr, bt\_byte evt, bt\_avctp\_event\_t \*evt
 —param, void \*callback\_param)

AVCTP application callback.

# **Functions**

• bt\_avctp\_mgr\_t \* bt\_avctp\_get\_mgr (void)

Return a pointer to an instance of the AVCTP manager.

void bt\_avctp\_init (void)

Initialize the AVCTP layer.

void bt\_avctp\_set\_callback (bt\_avctp\_mgr\_t \*mgr, bt\_avctp\_mgr\_callback\_fp callback, void \*callback\_
 param)

Register a AVCTP application callback.

• bt\_avctp\_channel\_t \* bt\_avctp\_create\_channel (bt\_avctp\_mgr\_t \*mgr, bt\_uint profile\_id, bt\_int psm, bt\_byte l2cap\_mode)

Allocate AVCTP channel.

• bt\_avctp\_channel\_t \* bt\_avctp\_create\_outgoing\_channel (bt\_avctp\_mgr\_t \*mgr, bt\_uint profile\_id, bt\_int psm, bt\_byte l2cap\_mode)

Allocate AVCTP channel.

• bt bool bt avctp destroy channel (bt avctp channel t \*channel)

Destroy AVCTP channel.

• bt\_bool bt\_avctp\_listen (bt\_avctp\_channel\_t \*channel)

Listen for incoming connections.

void bt\_avctp\_cancel\_listen (bt\_avctp\_channel\_t \*channel)

Cancel listening for incoming connections.

• bt\_byte bt\_avctp\_get\_channel\_state (bt\_avctp\_channel\_t \*channel)

Get channel state.

• bt\_bdaddr\_t \* bt\_avctp\_get\_channel\_remote\_address (bt\_avctp\_channel\_t \*channel)

Get channel's remote BT address.

• bt\_bool bt\_avctp\_connect (bt\_avctp\_channel\_t \*channel, bt\_bdaddr\_t \*remote\_address, bt\_uint acl\_config)

Connect to a remote device.

bt\_bool bt\_avctp\_disconnect (bt\_avctp\_channel\_t \*channel)

Disconnect from a remote device.

• bt\_bool bt\_avctp\_send\_command (bt\_avctp\_channel\_t \*channel, bt\_byte \*data, bt\_uint data\_len, bt\_byte \*tran id)

Send a command message to a remote device.

• bt\_bool bt\_avctp\_send\_response (bt\_avctp\_channel\_t \*channel, bt\_byte tran\_id, bt\_byte \*data, bt\_uint data\_len)

Send a response message to a remote device.

void bt\_avctp\_cancel\_command (bt\_avctp\_channel\_t \*channel, bt\_byte tran\_id)

Cancel a command message.

• void bt\_avctp\_cancel\_response (bt\_avctp\_channel\_t \*channel, bt\_byte tran\_id)

Cancel a response message.

• bt\_hci\_conn\_state\_t \* bt\_avctp\_get\_hci\_connection (bt\_avctp\_channel\_t \*channel)

Get HCI connection for a channel.

# 3.14.1 Detailed Description

AVCTP is the transport mechanisms used to exchange messages for controlling Audio and/or Video devices.

The actual message contents are defined in the applicable A/V control profiles.

# 3.14.2 Typedef Documentation

3.14.2.1 typedef void(\* bt\_avctp\_mgr\_callback\_fp) (struct \_bt\_avctp\_mgr\_t \*mgr, bt\_byte evt, bt\_avctp\_event\_t \*evt\_param, void \*callback\_param)

# AVCTP application callback.

In order to be notified of various events a consumer of the AVCTP layer has to register a callback function (done with <a href="mailto:bt\_avctp\_set\_callback">bt\_avctp\_set\_callback</a>()). The stack will call that function whenever a new event has been generated.

mgr	AVCTP manager.
evt	AVCTP event. The event can be one of the following values:
	<ul> <li>AVCTP_EVT_CHANNEL_CONNECTED Channel connected.</li> </ul>
	<ul> <li>AVCTP_EVT_CHANNEL_DISCONNECTED Channel disconnected.</li> </ul>
	<ul> <li>AVCTP_EVT_CONNECTION_FAILED Channel connection failed (generated only if connection has been initiated by the local device).</li> </ul>
	<ul> <li>AVCTP_EVT_COMMAND_RECEIVED Command received.</li> </ul>
	<ul> <li>AVCTP_EVT_RESPONE_RECEIVED Response received.</li> </ul>
	<ul> <li>AVCTP_EVT_COMMAND_SENT Command sent.</li> </ul>
	AVCTP_EVT_RESPONSE_SENT Response sent.
	<ul> <li>AVCTP_EVT_COMMAND_CANCELLED Command canceled.</li> </ul>
	AVCTP_EVT_RESPONSE_CANCELLED Response canceled.
evt_param	Event parameter. Which member of the bt_avctp_event_t union is valid depends on the event:
	• AVCTP_EVT_CHANNEL_CONNECTED bt_avctp_evt_channel_ ⇔ connected_t channel_connected
	• AVCTP_EVT_CHANNEL_DISCONNECTED bt_avctp_evt_channel_⇔ disconnected_t channel_disconnected
	• AVCTP_EVT_CONNECTION_FAILED bt_avctp_evt_connection_← failed_t connection_failed
	• AVCTP_EVT_COMMAND_RECEIVED bt_avctp_evt_command_received  _t command_received
	• AVCTP_EVT_RESPONE_RECEIVED bt_avctp_evt_response_← received_t response_received
	• AVCTP_EVT_COMMAND_SENT bt_avctp_evt_command_sent_← t command_sent
	• AVCTP_EVT_RESPONSE_SENT bt_avctp_evt_response_sent_ ← t response_sent
	• AVCTP_EVT_COMMAND_CANCELLED bt_avctp_evt_command_← cancelled_t command_cancelled
	• AVCTP_EVT_RESPONSE_CANCELLED bt_avctp_evt_response_← cancelled_t response_cancelled
callback_param	A pointer to an arbitrary data set by a call to bt_avctp_set_callback.

# 3.14.3 Function Documentation

3.14.3.1 void bt\_avctp\_cancel\_command ( bt\_avctp\_channel\_t \* channel, bt\_byte tran\_id )

Cancel a command message.

If a message has not yet been sent to the remote device, it can be canceled (i.e. removed from send queue) by calling this function.

#### **Parameters**

channel	AVCTP channel.
tran_id	Transaction Id. This value is obtained by calling bt_avctp_send_command.

## Returns

- TRUE if the function succeeds.
- FALSE otherwise. No events will be generated.

3.14.3.2 void bt\_avctp\_cancel\_listen ( bt\_avctp\_channel\_t \* channel )

Cancel listening for incoming connections.

This function stops listening for incoming connections on the specified channel.

#### **Parameters**

channel	AVCTP channel.
---------	----------------

#### Returns

- TRUE if the function succeeds.
- FALSE otherwise.

3.14.3.3 void bt\_avctp\_cancel\_response ( bt\_avctp\_channel\_t \* channel, bt\_byte tran\_id )

Cancel a response message.

If a message has not yet been sent to the remote device, it can be canceled (i.e. removed from send queue) by calling this function.

### **Parameters**

channel	AVCTP channel.
tran_id	Transaction Id. This value is obtained by calling bt_avctp_send_command.

## Returns

- TRUE if the function succeeds.
- FALSE otherwise. No events will be generated.

3.14.3.4 bt\_bool bt\_avctp\_connect ( bt\_avctp\_channel\_t \* channel, bt\_bdaddr\_t \* remote\_address, bt\_uint acl\_config )

Connect to a remote device.

This function establishes a connection to a remote device specified by the remote\_address. If connection cannot be initiated for some reason, for example, there is not enough resources, it returns FALSE and no events are generated. Otherwise the result of an attempt to connect to the remote device is reported via the AVCTP callback. The events generated will either be AVCTP\_EVT\_CHANNEL\_CONNECTED or AVCTP\_EVT\_CONNE CTION\_FAILED.

### **Parameters**

channel	AVCTP channel.
remote_address	The address of a remote device.
acl_config	ACL link configuration. This can be a combination of the following values:
	HCI_CONFIG_ENABLE_AUTHENTICATION
	HCI_CONFIG_ENABLE_ENCRYPTION
	HCI_CONFIG_BECOME_MASTER

### Returns

- · TRUE if connection establishment has been started.
- FALSE otherwise.
- 3.14.3.5 bt\_avctp\_channel\_t\* bt\_avctp\_create\_channel ( bt\_avctp\_mgr\_t \* mgr, bt\_uint  $profile_id$ , bt\_int psm, bt\_byte  $l2cap\_mode$  )

# Allocate AVCTP channel.

This function allocates a new incoming AVCTP channel. The channel is intended to be used to accept a connection from a remote device. There can be only one channel for each combination of profile\_id and psm.

### **Parameters**

mgr	AVCTP manager.
profile_id	Profile Id
psm	The PSM on which the underlying L2CAP channel will listen and accept incoming connec-
	tions.
l2cap_mode	Underlying L2CAP channel mode. This currently can only be CMODE_BASIC.

# Returns

- A pointer to the new AVCTP channel if the function succeeds.
- NULL otherwise.
- 3.14.3.6 bt\_avctp\_channel\_t\* bt\_avctp\_create\_outgoing\_channel ( bt\_avctp\_mgr\_t \* mgr, bt\_uint profile\_id, bt\_int psm, bt\_byte l2cap\_mode )

# Allocate AVCTP channel.

This function allocates a new outgoing AVCTP channel. The channel is intended to be used to create a connection to a remote device. There can be multiple channels with the same  $profile\_id$  and psm.

mgr	AVCTP manager.
profile_id	Profile Id
psm	The PSM on which the underlying L2CAP channel will listen and accept incoming connec-
	tions.

*I2cap\_mode* Underlying L2CAP channel mode. This currently can only be CMODE\_BASIC.

### Returns

- A pointer to the new AVCTP channel if the function succeeds.
- NULL otherwise.

3.14.3.7 bt\_bool bt\_avctp\_destroy\_channel ( bt\_avctp\_channel\_t \* channel )

Destroy AVCTP channel.

This function frees memory used by the channel. The channel has to exist and be in the "idle" state for this function to succeed. I.e. the channel has to be disconnected before this function can be called.

#### **Parameters**

channel AVCTP channel.

#### Returns

- TRUE if the function succeeds.
- FALSE otherwise.

3.14.3.8 bt\_bool bt\_avctp\_disconnect ( bt\_avctp\_channel\_t \* channel )

Disconnect from a remote device.

This function closes a connection to a remote device.

# Parameters

channel AVCTP channel.

### Returns

- TRUE if disconnection has been started.
- FALSE otherwise. No events will be generated.

3.14.3.9 bt\_bdaddr\_t\* bt\_avctp\_get\_channel\_remote\_address ( bt\_avctp\_channel\_t \* channel )

Get channel's remote BT address.

This function returns the address of the remote device associated with the channel.

# **Parameters**

channel AVCTP channel.

# Returns

- The address of the remote device if channel is connected.
- · NULL otherwise.

3.14.3.10 bt\_byte bt\_avctp\_get\_channel\_state ( bt\_avctp\_channel\_t \* channel )

Get channel state.

This function return current state of the specified channel

### **Parameters**

channel	AVCTP channel.

# Returns

- AVCTP\_CHANNEL\_STATE\_FREE
- · AVCTP CHANNEL STATE IDLE
- · AVCTP CHANNEL STATE CONNECTING
- · AVCTP CHANNEL STATE CONNECTED
- AVCTP\_CHANNEL\_STATE\_DISCONNECTING

3.14.3.11 bt\_hci\_conn\_state\_t\* bt\_avctp\_get\_hci\_connection ( bt\_avctp\_channel\_t \* channel )

Get HCI connection for a channel.

This function returns a pointer to a structure that describes an HCI connection a channel is open on. The return value can be used to call various function from the HCI layer. For example, if an app wants to force disconnection from a remote device it can call bt\_hci\_disconnect.

# **Parameters**

channel	AVCTP channel.

#### Returns

- Pointer to a structure that describes an HCI connection if the function succeeds.
- NULL otherwise. The function fails only if a channel specified by the channel parameter
- does not exist or there is no HCI connection between local and remote devices associated with the channel.

3.14.3.12 bt\_avctp\_mgr\_t\* bt\_avctp\_get\_mgr ( void )

Return a pointer to an instance of the AVCTP manager.

This function returns a pointer to an instance of the AVCTP manager. There is only one instance of the manager allocated by the stack.

3.14.3.13 void bt\_avctp\_init (void)

Initialize the AVCTP layer.

This function initializes the AVCTP layer of the stack. It must be called prior to any other AVCTP function can be called.

3.14.3.14 bt bool bt avctp\_listen ( bt\_avctp\_channel\_t \* channel )

Listen for incoming connections.

This function enables incoming connections on the specified AVCTP channel.

### **Parameters**

channel	AVCTP channel.
---------	----------------

# Returns

- TRUE if the function succeeds.
- FALSE otherwise.

3.14.3.15 bt\_bool bt\_avctp\_send\_command ( bt\_avctp\_channel\_t \* channel, bt\_byte \* data, bt\_uint data\_len, bt\_byte \* tran\_id )

Send a command message to a remote device.

This function sends a command message to a remote device.

### **Parameters**

channel	AVCTP channel.
data	Message body.
data_len	Message body length.
tran_id	Pointer to a bt_byte where AVRCP will write transaction id assigned to the message.

#### Returns

- TRUE if the function succeeds.
- FALSE otherwise. No events will be generated.

3.14.3.16 bt\_bool bt\_avctp\_send\_response ( bt\_avctp\_channel\_t \* channel, bt\_byte tran\_id, bt\_byte \* data, bt\_uint data\_len )

Send a response message to a remote device.

This function sends a response message to a remote device.

# Parameters

channel	AVCTP channel.
tran_id	Transaction Id. This value is obtained by calling bt_avctp_send_command.
data	Message body.
data_len	Message body length.

### Returns

- TRUE if the function succeeds.
- FALSE otherwise. No events will be generated.

3.14.3.17 void bt\_avctp\_set\_callback ( bt\_avctp\_mgr\_t \* mgr, bt\_avctp\_mgr\_callback\_fp callback, void \* callback\_param )

Register a AVCTP application callback.

In order to be notified of various events a consumer of the AVCTP layer has to register a callback function. The stack will call this function whenever a new event has been generated passing the code of the event as the second parameter. The event can be one of the following values:

• AVCTP\_EVT\_CHANNEL\_CONNECTED Channel connected.

- AVCTP\_EVT\_CHANNEL\_DISCONNECTED Channel disconnected.
- AVCTP\_EVT\_CONNECTION\_FAILED Channel connection failed (generated only if connection has been initiated by the local device).
- AVCTP\_EVT\_COMMAND\_RECEIVED Command received.
- AVCTP\_EVT\_RESPONE\_RECEIVED Response received.
- AVCTP\_EVT\_COMMAND\_SENT Command sent.
- AVCTP\_EVT\_RESPONSE\_SENT Response sent.
- AVCTP\_EVT\_COMMAND\_CANCELLED Command canceled.
- AVCTP\_EVT\_RESPONSE\_CANCELLED Response canceled.

mgr	AVCTP manager.
callback	The callback function that will be called when the AVCTP generates an event.
callback_param	A pointer to arbitrary data to be passed to the callback callback.

# 3.15 Configuration

This module describes parameters used to configure AVCTP layer.

# Macros

• #define AVCTP MAX CHANNELS

Maximum number of AVCTP channels.

#define AVCTP\_MAX\_TRANSPORT\_CHANNELS

Maximum number of AVCTP transports.

#define AVCTP\_ALLOCATE\_BUFFERS\_VARS()

Maximum size of received message.

# 3.15.1 Detailed Description

This module describes parameters used to configure AVCTP layer.

dotstack is customized using a configuration file. The configuration file tailors the dotstack to the application being built. It has to have the structure shown below.

```
#include "cdbt/bt/bt std.h"
// HCI, L2CAP and SDP must always be present
// HCI configuration parameters
#define HCI_MAX_CMD_BUFFERS
#define HCI_MAX_DATA_BUFFERS
#define HCI_MAX_HCI_CONNECTIONS
#define HCI_RX_BUFFER_LEN
#define HCI_TX_BUFFER_LEN
                                         . . .
#define HCI_L2CAP_BUFFER_LEN
#define HCI_MAX_CMD_PARAM_LEN
// L2CAP configuration parameters
#define L2CAP_MAX_CMD_BUFFERS
#define L2CAP_MAX_FRAME_BUFFERS
#define L2CAP_MAX_PSMS
#define L2CAP_MAX_CHANNELS
// SDP configuration parameters
#define SDP_MAX_SEARCH_RESULT LEN
#define SDP_MAX_ATTRIBUTE_RESULT_LEN
// Depending on protocols and profiles used below goes configuration parameters
// for each used module. E.g., to use and configure AVRCP,
\ensuremath{//} the following values must be defined:
#define BT_INCLUDE_AVCTP
                                        // tells dotstack to compile in AVCTP support
#define AVCTP_MAX_CHANNELS
                                        . . .
#define AVCTP_MAX_TRANSPORT_CHANNELS ...
#define AVCTP_MAX_RX_MESSAGE_LEN
#define AVCTP_MAX_MESSAGE_BUFFERS
#include "cdbt/bt/bt_oem_config.h"
```

# 3.15.2 Macro Definition Documentation

# 3.15.2.1 #define AVCTP\_ALLOCATE\_BUFFERS\_VARS( )

# Value:

3.15 Configuration 147

Maximum size of received message.

This parameter defines the maximum size of a fragmented message (command or response) a local device can accept from a remote device. If fragmented message are not expected this parameter can be set to 1. Maximum number of message buffers

This parameter defines the maximum number of buffer available for sending message.

# 3.15.2.2 #define AVCTP\_MAX\_CHANNELS

Maximum number of AVCTP channels.

This parameter defines the maximum number of channels a local device can have with remote devices.

# 3.15.2.3 #define AVCTP\_MAX\_TRANSPORT\_CHANNELS

Maximum number of AVCTP transports.

This parameter defines the maximum number of transports a local device can have with remote devices. This value should not exceed AVCTP\_MAX\_CHANNELS.

#### Audio/Video Remote Control Profile (AVRCP) 3.16

The Audio/Video Remote Control Profile (AVRCP) defines the features and procedures required in order to ensure interoperability between Bluetooth devices with audio/video control functions in the Audio/Video distribution scenarios.

# **Modules**

Configuration

This module describes parameters used to configure AVRCP layer.

# **Data Structures**

```
    struct bt_avrcp_evt_search_completed_t

     Parameter to AVRCP_EVT_SEARCH_COMPLETED event.
• struct bt_avrcp_evt_channel_connected_t
     Parameter to AVRCP EVT CONTROL CHANNEL CONNECTED event.

    struct bt_avrcp_evt_channel_disconnected_t

     Parameter to AVRCP_EVT_CONTROL_CHANNEL_DISCONNECTED event.
• struct bt_avrcp_evt_connection_failed_t
     Parameter to AVRCP_EVT_CONTROL_CONNECTION_FAILED event.

    struct bt_avrcp_evt_panel_response_received_t

     Parameter to AVRCP_EVT_PANEL_RESPONSE_RECEIVED event.

    struct bt_avrcp_evt_panel_command_received_t

     Parameter to AVRCP_EVT_PANEL_COMMAND_RECEIVED event.
union bt_avrcp_event_t
     Parameter to an application callback.

    struct bt_avrcp_channel_t

     AVRCP channel description.
struct bt_avrcp_mgr_t
     AVRCP manager.

    struct bt_av_response_t

     AV/C response header.
· struct bt av element id t
     Media element UID.

    struct bt_av_capability_company_id_t

     Parameter to AVRCP EVT COMPANY ID LIST RECEIVED event.

    struct bt_av_capability_event_id_t

     Parameter to AVRCP_EVT_EVENT_ID_LIST_RECEIVED event.
• struct bt_av_player_settings_t
     Parameter to AVRCP EVT PLAYER SETTING ATTRIBUTES RECEIVED event.

    struct bt_av_player_setting_values_t

     Parameter to AVRCP_EVT_PLAYER_SETTING_VALUES_RECEIVED event.
struct bt_av_player_setting_current_values_t
     Parameter to AVRCP EVT PLAYER CURRENT SETTING VALUES RECEIVED event.

    struct bt_av_player_settings_text_t

     Parameter to AVRCP_EVT_PLAYER_SETTING_ATTRIBUTES_TEXT_RECEIVED event.

    struct bt_av_player_setting_values_text_t
```

Parameter to AVRCP\_EVT\_PLAYER\_SETTING\_VALUES\_TEXT\_RECEIVED event.

struct bt\_av\_element\_attribute\_t

Media element attribute.

struct bt\_av\_element\_attributes\_t

Parameter to AVRCP\_EVT\_GET\_ELEMENT\_ATTRIBUTES\_RECEIVED event.

struct bt\_av\_play\_status\_t

Parameter to AVRCP\_EVT\_GET\_PLAY\_STATUS\_RECEIVED event.

struct bt\_av\_battery\_status\_of\_ct\_t

Parameter to AVRCP\_EVT\_BATTERY\_STATUS\_OF\_CT\_RECEIVED event.

struct bt av displayable character set t

Parameter to AVRCP\_EVT\_DISPLAYABLE\_CHARACTER\_SET\_RECEIVED event.

struct bt\_av\_notification\_playback\_status\_changed\_t

Parameter to AVRCP EVT PLAYBACK STATUS CHANGED event.

struct bt\_av\_notification\_track\_changed\_t

Parameter to AVRCP\_EVT\_TRACK\_CHANGED event.

struct bt\_av\_notification\_playback\_pos\_changed\_t

Parameter to AVRCP\_EVT\_PLAYBACK\_POS\_CHANGED event.

• struct bt\_av\_notification\_battery\_status\_t

Parameter to AVRCP\_EVT\_BATT\_STATUS\_CHANGED event.

struct bt\_av\_notification\_system\_status\_changed\_t

Parameter to AVRCP EVT SYSTEM STATUS CHANGED event.

struct bt\_av\_notification\_addressed\_player\_changed\_t

Parameter to AVRCP\_EVT\_ADDRESSED\_PLAYER\_CHANGED event.

• struct bt\_av\_notification\_uids\_changed\_t

Parameter to AVRCP\_EVT\_UIDS\_CHANGED event.

· struct bt\_av\_notification\_volume\_changed\_t

Parameter to AVRCP\_EVT\_VOLUME\_CHANGED event.

struct bt\_av\_notification\_t

Parameter to the following events:

• struct bt\_av\_set\_absolute\_volume\_t

Parameter to AVRCP\_EVT\_SET\_ABSOLUTE\_VOLUME\_COMPLETED event.

struct bt\_av\_set\_addressed\_player\_t

Parameter to AVRCP\_EVT\_SET\_ADDRESSED\_PLAYER\_COMPLETED event.

struct bt\_av\_play\_item\_t

Parameter to AVRCP\_EVT\_PLAY\_ITEM\_COMPLETED event.

struct bt\_av\_add\_to\_now\_playing\_t

Parameter to AVRCP\_EVT\_ADD\_TO\_NOW\_PLAYING\_COMPLETED event.

struct bt\_av\_get\_element\_attributes\_t

Parameter to AVRCP\_EVT\_ELEMENT\_ATTRIBUTES\_REQUESTED event.

· struct bt\_av\_register\_notification\_t

Parameter to AVRCP\_EVT\_REGISTER\_NOTIFICATION\_REQUESTED event.

## **Typedefs**

• typedef void(\* bt\_avrcp\_mgr\_callback\_fp) (bt\_avrcp\_mgr\_t \*mgr, bt\_byte evt, bt\_avrcp\_event\_t \*evt\_param, void \*callback\_param)

AVRCP application callback.

typedef void(\* bt\_avrcp\_find\_callback\_fp) (bt\_bool found, void \*param)

Find Controller/Target callback.

#### **Functions**

bt avrcp mgr t \* bt avrcp get mgr (void)

Return a pointer to an instance of the AVRCP manager.

void bt\_avrcp\_init\_target (bt\_ulong company\_id, bt\_uint supported\_events)

Initialize AVRCP to be used in target mode.

void bt\_avrcp\_init\_controller (void)

Initialize AVRCP to be used in controller mode.

• void bt\_avrcp\_start (bt\_avrcp\_mgr\_t \*mgr, bt\_avrcp\_mgr\_callback\_fp callback, void \*callback\_param)

Start the AVRCP layer.

• bt\_avrcp\_channel\_t \* bt\_avrcp\_create\_channel (bt\_avrcp\_mgr\_t \*mgr, bt\_bool create\_browsing\_channel)

Allocate AVRCP channel.

bt\_avrcp\_channel\_t \* bt\_avrcp\_create\_outgoing\_channel (bt\_avrcp\_mgr\_t \*mgr, bt\_bool create\_browsing
 — channel)

Allocate AVRCP channel.

• bt\_bool bt\_avrcp\_destroy\_channel (bt\_avrcp\_channel\_t \*channel)

Destroy AVRCP channel.

• bt\_bool bt\_avrcp\_listen (bt\_avrcp\_channel\_t \*channel)

Listen for incoming connections.

void bt\_avrcp\_cancel\_listen (bt\_avrcp\_channel\_t \*channel)

Cancel listening for incoming connections.

bt\_byte bt\_avrcp\_get\_control\_channel\_state (bt\_avrcp\_channel\_t \*channel)

Get AVCTP control channel state.

• bt\_byte bt\_avrcp\_get\_browsing\_channel\_state (bt\_avrcp\_channel\_t \*channel)

Get AVCTP browsing channel state.

• bt\_bdaddr\_t \* bt\_avrcp\_get\_channel\_remote\_address (bt\_avrcp\_channel\_t \*channel)

Get channel's remote BT address.

• bt bool bt avrcp connect (bt avrcp channel t \*channel, bt bdaddr t \*remote address)

Connect to a remote device.

• bt\_bool bt\_avrcp\_disconnect (bt\_avrcp\_channel\_t \*channel)

Disconnect from a remote device.

• bt\_hci\_conn\_state\_p \* bt\_avrcp\_get\_hci\_connection (bt\_avrcp\_channel\_t \*channel)

Get HCI connection for a channel.

• bt\_bool bt\_avrcp\_send\_cmd (bt\_avrcp\_channel\_t \*channel, bt\_av\_command\_t \*command)

Send AVRCP command.

bt\_bool bt\_avrcp\_add\_to\_now\_playing (bt\_avrcp\_channel\_t \*channel, bt\_byte scope, bt\_av\_element\_id\_

 t \*element\_id, bt\_uint counter)

Add to "now playing" list.

bt\_bool bt\_avrcp\_get\_company\_id\_list (bt\_avrcp\_channel\_t \*channel)

Get Company ID list.

bt\_bool bt\_avrcp\_get\_supported\_event\_id\_list (bt\_avrcp\_channel\_t \*channel)

Get supported events.

Get current player setting values.

bt\_bool bt\_avrcp\_get\_element\_attributes (bt\_avrcp\_channel\_t \*channel, bt\_av\_element\_id\_t \*element\_id, bt\_uint attr\_mask)

Get media element attributes.

bt\_bool bt\_avrcp\_get\_play\_status (bt\_avrcp\_channel\_t \*channel, bt\_uint repeat\_interval)

Get playback status.

bt\_bool bt\_avrcp\_get\_player\_application\_setting\_attr\_text (bt\_avrcp\_channel\_t \*channel, bt\_av\_player\_
 settings\_text\_t \*response\_buffer)

Get player setting attribute text.

• bt\_bool bt\_avrcp\_get\_player\_application\_setting\_value\_text (bt\_avrcp\_channel\_t \*channel, bt\_byte attr\_id, bt\_av\_player\_setting\_values\_text\_t \*response\_buffer)

Get player setting value text.

bt\_bool bt\_avrcp\_inform\_battery\_status (bt\_avrcp\_channel\_t \*channel, bt\_byte status)

Inform controller's battery status.

bt\_bool bt\_avrcp\_inform\_displayable\_character\_set (bt\_avrcp\_channel\_t \*channel, bt\_uint \*charset\_list, bt\_byte charset\_count)

Inform displayable character set.

bt\_bool bt\_avrcp\_list\_player\_application\_setting\_attributes (bt\_avrcp\_channel\_t \*channel, bt\_av\_player\_
 settings\_t \*response\_buffer)

Get supported player setting attributes.

bt\_bool bt\_avrcp\_list\_player\_application\_setting\_values (bt\_avrcp\_channel\_t \*channel, bt\_byte attr\_id, bt
 \_av\_player\_setting\_values\_t \*response\_buffer)

Get player setting attribute values.

• bt\_bool bt\_avrcp\_play\_item (bt\_avrcp\_channel\_t \*channel, bt\_byte scope, bt\_av\_element\_id\_t \*element\_id, bt\_uint counter)

Play media item.

bt\_bool bt\_avrcp\_register\_notification (bt\_avrcp\_channel\_t \*channel, bt\_byte event\_id, bt\_ulong playback
 \_interval)

Register notification.

• bt\_bool bt\_avrcp\_register\_notifications (bt\_avrcp\_channel\_t \*channel, bt\_uint event\_mask)

Register notifications.

• bt bool bt avrcp set absolute volume (bt avrcp channel t \*channel, bt byte volume)

Set absolute volume.

bt\_bool bt\_avrcp\_set\_addressed\_player (bt\_avrcp\_channel\_t \*channel, bt\_uint player\_id)

Set addressed player.

• bt\_bool bt\_avrcp\_set\_player\_application\_setting\_value (bt\_avrcp\_channel\_t \*channel, bt\_byte \*attr\_id\_list, bt\_byte \*attr\_value\_list, bt\_byte attr\_id\_count)

Set player setting attribute values.

• bt bool bt avrcp find targets (bt byte search length)

Find Targets.

bt\_bool bt\_avrcp\_cancel\_find (void)

Cancel finding Targets.

bt\_bool bt\_avrcp\_find\_target (bt\_bdaddr\_t \*deviceAddress, bt\_avrcp\_find\_callback\_fp callback, bt\_sdp\_

 client\_callback\_fp client\_callback, void \*callback\_param)

Find Target.

bt\_bool bt\_avrcp\_find\_controller (bt\_bdaddr\_t \*deviceAddress, bt\_avrcp\_find\_callback\_fp callback, bt\_sdp
 — client\_callback\_fp client\_callback, void \*callback\_param)

Find Controller.

void bt\_avrcp\_tg\_set\_play\_status (bt\_ulong song\_length, bt\_ulong song\_position, bt\_byte play\_status)

void bt\_avrcp\_tg\_set\_current\_track (bt\_av\_element\_id\_t \*track\_id, bt\_ulong song\_length, bt\_ulong song\_
 position)

Set current track.

Set playback status.

• void bt\_avrcp\_tg\_set\_absolute\_volume (bt\_byte volume)

Set absolute volume.

void bt\_avrcp\_tg\_set\_battery\_status (bt\_byte status)

Set battery status.

void bt\_avrcp\_tg\_set\_system\_status (bt\_byte status)

Set system status.

bt\_bool bt\_avrcp\_tg\_send\_element\_attributes (bt\_avrcp\_channel\_t \*channel, bt\_byte tran\_id, bt\_av\_
 element\_attribute\_t \*attrs, bt\_byte attr\_count)

Send media element attributes.

• bt bool bt avrcp get unit info (bt avrcp channel t\*channel)

Get unit info.

• bt\_bool bt\_avrcp\_get\_subuint\_info (bt\_avrcp\_channel\_t \*channel)

Get subunit info.

bt\_bool bt\_avrcp\_send\_simple\_panel\_cmd (bt\_avrcp\_channel\_t \*channel, bt\_byte ctype, bt\_byte opid, bt←byte button state)

Send AV/C Panel Subunit PASS THROUGH command.

• bt\_bool bt\_avrcp\_send\_press\_panel\_control (bt\_avrcp\_channel\_t \*channel, bt\_byte opid)

Send AV/C Panel Subunit "pressed" PASS THROUGH command.

bt\_bool bt\_avrcp\_send\_release\_panel\_control (bt\_avrcp\_channel\_t \*channel, bt\_byte opid)

Send AV/C Panel Subunit "released" PASS THROUGH command.

bt\_bool bt\_avrcp\_send\_panel\_control (bt\_avrcp\_channel\_t \*channel, bt\_byte opid, bt\_byte button\_state)

Send AV/C Panel Subunit "control" PASS THROUGH command.

bt\_bool bt\_avrcp\_send\_button\_click (bt\_avrcp\_channel\_t \*channel, bt\_byte button\_id)

Send AV/C Panel Subunit "click" PASS THROUGH command.

## **Events**

The following is a list of events AVRCP layer generates and can report to the upper layer when it completes executing an operation initiated by either local or remote device.

- #define AVCTP\_EVT\_NOTHING 0
- #define AVCTP\_EVT\_CHANNEL\_CONNECTED 1

This event is generated when a channel between two AVCTP entities has been established.

#define AVCTP EVT CHANNEL DISCONNECTED 2

This event is generated when a channel between two AVCTP entities has been terminated.

#define AVCTP\_EVT\_CONNECTION\_FAILED 3

This event is generated when a local device failed to create a channel between two AVCTP entities.

• #define AVCTP\_EVT\_COMMAND\_RECEIVED 50

This event is generated when a local device received a command.

• #define AVCTP\_EVT\_RESPONE\_RECEIVED 51

This event is generated when a local device received a response.

#define AVCTP\_EVT\_COMMAND\_SENT 52

This event is generated when a local device finished sending a command.

• #define AVCTP\_EVT\_RESPONSE\_SENT 53

This event is generated when a local device finished sending a response.

#define AVCTP\_EVT\_COMMAND\_CANCELLED 54

This event is generated when a command has been canceled.

#define AVCTP\_EVT\_RESPONSE\_CANCELLED 55

This event is generated when a response has been canceled.

## **Events**

The following is a list of events AVRCP layer generates and can report to the upper layer when it completes executing an operation initiated by either local or remote device.

- #define AVRCP\_EVT\_NOTHING 0
- #define AVRCP\_EVT\_CONTROL\_CHANNEL\_CONNECTED 1

This event is generated when a control channel between two AVRCP entities has been established.

#define AVRCP\_EVT\_CONTROL\_CHANNEL\_DISCONNECTED 2

This event is generated when a control channel between two AVRCP entities has been terminated.

• #define AVRCP EVT CONTROL CONNECTION FAILED 3

This event is generated when a local device failed to create a control channel between two AVRCP entities.

• #define AVRCP\_EVT\_BROWSING\_CHANNEL\_CONNECTED 4

This event is generated when a browsing channel between two AVRCP entities has been established.

#define AVRCP EVT BROWSING CHANNEL DISCONNECTED 5

This event is generated when a browsing channel between two AVRCP entities has been terminated.

#define AVRCP\_EVT\_BROWSING\_CONNECTION\_FAILED 6

This event is generated when a local device failed to create a browsing channel between two AVRCP entities.

#define AVRCP EVT SEARCH COMPLETED 7

This event is generated when a local device completed searching for nearby targets.

#define AVRCP EVT PANEL RESPONSE RECEIVED 50

This event is generated when a local device received a response to a PASS THROUGH command.

#define AVRCP\_EVT\_COMPANY\_ID\_LIST\_RECEIVED 51

This event is generated when a local device received a response to a "get company id" request.

#define AVRCP\_EVT\_EVENT\_ID\_LIST\_RECEIVED 52

This event is generated when a local device received a response to a "get supported events" request.

#define AVRCP EVT PLAYER SETTING ATTRIBUTES RECEIVED 53

This event is generated when a local device received a response to a "get supported player setting attributes" request.

#define AVRCP\_EVT\_PLAYER\_SETTING\_VALUES\_RECEIVED 54

This event is generated when a local device received a response to a "get player setting attribute values" request.

• #define AVRCP\_EVT\_PLAYER\_CURRENT\_SETTING\_VALUES\_RECEIVED 55

This event is generated when a local device received a response to a "get current player setting attribute values" request.

#define AVRCP EVT SET PLAYER SETTING VALUES COMPLETED 56

This event is generated when a local device received a response to a "set player setting attribute values" request.

#define AVRCP\_EVT\_PLAYER\_SETTING\_ATTRIBUTES\_TEXT\_RECEIVED 57

This event is generated when a local device received a response to a "get player setting attributes displayable text" request.

#define AVRCP EVT PLAYER SETTING VALUES TEXT RECEIVED 58

This event is generated when a local device received a response to a "get player setting attribute values displayable text" request.

#define AVRCP\_EVT\_INFORM\_DISPLAYABLE\_CHARACTER\_SET\_COMPLETED 59

This event is generated when a local device received a response to a "inform displayable character set" request.

#define AVRCP\_EVT\_INFORM\_BATTERY\_STATUS\_OF\_CT\_COMPLETED 60

This event is generated when a local device received a response to a "inform battery status" request.

#define AVRCP\_EVT\_GET\_ELEMENT\_ATTRIBUTES\_RECEIVED 61

This event is generated when a local device received a response to a "get media element attributes" request.

#define AVRCP\_EVT\_GET\_PLAY\_STATUS\_RECEIVED 62

This event is generated when a local device received a response to a "get play status" request.

#define AVRCP\_EVT\_PLAYBACK\_STATUS\_CHANGED 63

This event is generated when a local device received a "play status changed" notification.

#define AVRCP EVT TRACK CHANGED 64

This event is generated when a local device received a "track changed changed" notification.

• #define AVRCP EVT TRACK REACHED END 65

This event is generated when a local device received a "track reached end" notification.

#define AVRCP\_EVT\_TRACK\_REACHED\_START 66

This event is generated when a local device received a "track reached start" notification.

#define AVRCP\_EVT\_PLAYBACK\_POS\_CHANGED 67

This event is generated when a local device received a "playback position changed" notification.

• #define AVRCP\_EVT\_BATT\_STATUS\_CHANGED 68

This event is generated when a local device received a "battery status changed" notification.

• #define AVRCP\_EVT\_SYSTEM\_STATUS\_CHANGED 69

This event is generated when a local device received a "system status changed" notification.

• #define AVRCP\_EVT\_PLAYER\_APPLICATION\_SETTING\_CHANGED 70

This event is generated when a local device received a "player application setting changed" notification.

#define AVRCP EVT NOW PLAYING CONTENT CHANGED 71

This event is generated when a local device received a "now playing content changed" notification.

#define AVRCP\_EVT\_AVAILABLE\_PLAYERS\_CHANGED 72

This event is generated when a local device received a "available players changed" notification.

• #define AVRCP\_EVT\_ADDRESSED\_PLAYER\_CHANGED 73

This event is generated when a local device received a "addressed player changed" notification.

#define AVRCP\_EVT\_UIDS\_CHANGED 74

This event is generated when a local device received a "UIDs changed" notification.

#define AVRCP EVT VOLUME CHANGED 75

This event is generated when a local device received a "volume changed" notification.

• #define AVRCP\_EVT\_SET\_ABSOLUTE\_VOLUME\_COMPLETED 76

This event is generated when a local device received a response to a "set absolute volume" request.

#define AVRCP\_EVT\_SET\_ADDRESSED\_PLAYER\_COMPLETED 77

This event is generated when a local device received a response to a "set addressed player" request.

#define AVRCP EVT PLAY ITEM COMPLETED 78

This event is generated when a local device received a response to a "play item" request.

• #define AVRCP\_EVT\_ADD\_TO\_NOW\_PLAYING\_COMPLETED 79

This event is generated when a local device received a response to a "add to now playing" request.

#define AVRCP\_EVT\_REGISTER\_NOTIFICATIONS\_COMPLETED 80

This event is generated when a local device received a response to a "register notification" request.

#define AVRCP\_EVT\_PANEL\_COMMAND\_RECEIVED 81

This event is generated when a local device received a PASS THROUGH command.

#define AVRCP EVT SET ABSOLUTE VOLUME REQUESTED 82

This event is generated when a local device received a "set absolute volume" request.

• #define AVRCP EVT BATTERY STATUS OF CT RECEIVED 83

This event is generated when a local device received a "battery status of controller" command.

• #define AVRCP\_EVT\_DISPLAYABLE\_CHARACTER\_SET\_RECEIVED 84

This event is generated when a local device received a "displayable chracter set command" request.

• #define AVRCP\_EVT\_ELEMENT\_ATTRIBUTES\_REQUESTED 85

This event is generated when a local device received a "get element attributes" request.

#define AVRCP\_EVT\_REGISTER\_NOTIFICATION\_REQUESTED 86

This event is generated when a local device received a "register notification" request.

## **Command types**

- #define AVC CTYPE CONTROL 0
- #define AVC\_CTYPE\_STATUS 1
- #define AVC\_CTYPE\_SPECIFIC\_IQUIRY 2
- #define AVC CTYPE NOTIFY 3
- #define AVC\_CTYPE\_GENERAL\_INQUORY 4

# Response types

- #define AVC\_RESPONSE\_NOT\_IMPLEMENTED 0x8
- #define AVC\_RESPONSE\_ACCEPTED 0x9
- #define AVC RESPONSE REJECTED 0xA
- #define AVC RESPONSE IN TRANSITION 0xB
- #define AVC RESPONSE STABLE 0xC
- #define AVC RESPONSE IMPLEMENTED 0xC
- #define AVC\_RESPONSE\_CHANGED 0xD
- #define AVC\_RESPONSE\_INTERIM 0xF
- #define AVC RESPONSE TIMEOUT 0xF0

## Subunit types

- #define AVC SUBUNIT TYPE MONITOR 0x00
- #define AVC\_SUBUNIT\_TYPE\_AUDIO 0x01
- #define AVC\_SUBUNIT\_TYPE\_PRINTER 0x02
- #define AVC\_SUBUNIT\_TYPE\_DISC 0x03
- #define AVC\_SUBUNIT\_TYPE\_TAPE\_RECORDER\_PLAYER 0x04
- #define AVC SUBUNIT TYPE TUNER 0x05
- #define AVC\_SUBUNIT\_TYPE\_CA 0x06
- #define AVC SUBUNIT TYPE CAMERA 0x07
- #define AVC\_SUBUNIT\_TYPE\_PANEL 0x09
- #define AVC\_SUBUNIT\_TYPE\_BULLETIN\_BOARD 0x0A
- #define AVC\_SUBUNIT\_TYPE\_CAMERA\_STORAGE 0x0B
- #define AVC\_SUBUNIT\_TYPE\_VENDOR\_UNIQUE 0x1C
- #define AVC SUBUNIT TYPE EXTENDED TO NEXT BYTE 0x1E
- #define AVC\_SUBUNIT\_TYPE\_UNIT 0x1F

# AV/C Panel PASS THROUGH operation IDs

- #define AVC PANEL OPID SELECT 0x00
- #define AVC PANEL OPID UP 0x01
- #define AVC PANEL OPID DOWN 0x02
- #define AVC PANEL OPID LEFT 0x03
- #define AVC\_PANEL\_OPID\_RIGHT 0x04
- #define AVC PANEL OPID RIGHT UP 0x05
- #define AVC\_PANEL\_OPID\_RIGHT\_DOWN 0x06
- #define AVC\_PANEL\_OPID\_LEFT\_UP 0x07
- #define AVC\_PANEL\_OPID\_LEFT\_DOWN 0x08
- #define AVC\_PANEL\_OPID\_ROOT\_MENU 0x09
- #define AVC\_PANEL\_OPID\_SETUP\_MENU 0x0A
- #define AVC\_PANEL\_OPID\_CONTENTS\_MENU 0x0B
- #define AVC\_PANEL\_OPID\_FAVORITE\_MENU 0x0C
- #define AVC PANEL OPID EXIT 0x0D
- #define AVC PANEL OPID ON DEMAND MENU 0x0E
- #define AVC PANEL OPID APPS MENU 0x0F
- #define AVC PANEL OPID 0 0x20
- #define AVC\_PANEL\_OPID\_1 0x21
- #define AVC\_PANEL\_OPID\_2 0x22
- #define AVC\_PANEL\_OPID\_3 0x23
- #define AVC\_PANEL\_OPID\_4 0x24
- #define AVC PANEL OPID 5 0x25
- #define AVC PANEL OPID 6 0x26

- #define AVC PANEL OPID 7 0x27
- #define AVC\_PANEL\_OPID\_8 0x28
- #define AVC\_PANEL\_OPID\_9 0x29
- #define AVC PANEL OPID DOT 0x2A
- #define AVC PANEL OPID ENTER 0x2B
- #define AVC\_PANEL\_OPID\_CLEAR 0x2C
- #define AVC PANEL OPID CHANNEL UP 0x30
- #define AVC\_PANEL\_OPID\_CHANNEL\_DOWN 0x31
- #define AVC\_PANEL\_OPID\_PREVIOUS\_CHANNEL 0x32
- #define AVC PANEL OPID SOUND SELECT 0x33
- #define AVC PANEL OPID INPUT SELECT 0x34
- #define AVC PANEL OPID DISPLAY INFORMATION 0x35
- #define AVC\_PANEL\_OPID\_HELP 0x36
- #define AVC PANEL OPID PAGE UP 0x37
- #define AVC\_PANEL\_OPID\_PAGE\_DOWN 0x38
- #define AVC PANEL OPID LIVE TV 0x39
- #define AVC PANEL OPID ZOOM 0x3A
- #define AVC PANEL OPID LOCK 0x3B
- #define AVC PANEL OPID SKIP 0x3C
- #define AVC\_PANEL\_OPID\_NEXT\_DAY 0x3D
- #define AVC\_PANEL\_OPID\_PREVIOUS\_DAY 0x3E
- #define AVC PANEL OPID LINKED CONTENT 0x3F
- #define AVC PANEL OPID POWER TOGGLE 0x40
- #define AVC\_PANEL\_OPID\_VOLUME\_UP 0x41
- #define AVC PANEL OPID VOLUME DOWN 0x42
- #define AVC\_PANEL\_OPID\_MUTE 0x43
- #define AVC\_PANEL\_OPID\_PLAY 0x44
- #define AVC PANEL OPID STOP 0x45
- #define AVC PANEL OPID PAUSE 0x46
- #define AVC PANEL OPID RECORD 0x47
- #define AVC\_PANEL\_OPID\_REWIND 0x48
- #define AVC PANEL OPID FAST FORWARD 0x49
- #define AVC\_PANEL\_OPID\_EJECT 0x4A
- #define AVC\_PANEL\_OPID\_FORWARD 0x4B
- #define AVC\_PANEL\_OPID\_BACKWARD 0x4C
- #define AVC PANEL OPID LIST 0x4D
- #define AVC PANEL OPID ANGLE 0x50
- #define AVC\_PANEL\_OPID\_SUBPICTURE 0x51
- #define AVC\_PANEL\_OPID\_PIP\_MOVE 0x52
- #define AVC PANEL OPID PIP DOWN 0x53
- #define AVC PANEL OPID PIP UP 0x54
- #define AVC\_PANEL\_OPID\_RF\_BYPASS 0x55
- #define AVC\_PANEL\_OPID\_PLAY\_FUNCTION 0x60
- #define AVC\_PANEL\_OPID\_PAUSE\_PLAY\_FUNCTION 0x61
- #define AVC PANEL OPID RECORD FUNCTION 0x62
- #define AVC PANEL OPID PAUSE RECORD FUNCTION 0x63
- #define AVC PANEL OPID STOP FUNCTION 0x64
- #define AVC PANEL OPID MUTE FUNCTION 0x65
- #define AVC\_PANEL\_OPID\_RESTORE\_FOLUME\_FUNCTION 0x66
- #define AVC\_PANEL\_OPID\_TUNE\_FUNCTION 0x67
- #define AVC PANEL OPID SELECT DISK FUNCTION 0x68
- #define AVC\_PANEL\_OPID\_SELECT\_AV\_INPUT\_FUNCTION 0x69
- #define AVC\_PANEL\_OPID\_SELECT\_AUDIO\_INPUT\_FUNCTION 0x6A
- #define AVC\_PANEL\_OPID\_POWER\_STATE\_FUNCTION 0x6B
- #define AVC\_PANEL\_OPID\_KEYBORD\_FUNCTION 0x6C

- #define AVC PANEL OPID F1 0x71
- #define AVC\_PANEL\_OPID\_F2 0x72
- #define AVC PANEL OPID F3 0x73
- #define AVC PANEL OPID F4 0x74
- #define AVC PANEL OPID F5 0x75
- #define AVC\_PANEL\_OPID\_F6 0x76
- #define AVC\_PANEL\_OPID\_F7 0x77
- #define AVC\_PANEL\_OPID\_F8 0x78
- #define AVC PANEL OPID F9 0x79
- #define AVC\_PANEL\_OPID\_A 0x7A
- #define AVC\_PANEL\_OPID\_B 0x7B
- #define AVC\_PANEL\_OPID\_C 0x7C
- #define AVC\_PANEL\_OPID\_D 0x7D
- #define AVC PANEL OPID VENDOR UNIQUE 0x7E

## **Battery status**

- #define AVC BATTERY STATUS NORMAL 0
- #define AVC\_BATTERY\_STATUS\_WARNING 1
- #define AVC\_BATTERY\_STATUS\_CRITICAL 2
- #define AVC BATTERY STATUS EXTERNAL 3
- #define AVC BATTERY STATUS FULL CHARGE 4

#### Media attribute IDs

- #define AVC\_MEDIA\_ATTR\_ID\_TITLE 1
- #define AVC MEDIA ATTR ID ARTIST 2
- #define AVC\_MEDIA\_ATTR\_ID\_ALBUM 3
- #define AVC\_MEDIA\_ATTR\_ID\_NUMBER 4
- #define AVC MEDIA ATTR ID TOTAL NUMBER 5
- #define AVC\_MEDIA\_ATTR\_ID\_GENRE 6
- #define AVC\_MEDIA\_ATTR\_ID\_PLAYING\_TIME 7

## Media attribute bitmask

- #define AVC MEDIA ATTR FLAG TITLE 0x01
- #define AVC\_MEDIA\_ATTR\_FLAG\_ARTIST 0x02
- #define AVC\_MEDIA\_ATTR\_FLAG\_ALBUM 0x04
- #define AVC MEDIA ATTR FLAG NUMBER 0x08
- #define AVC\_MEDIA\_ATTR\_FLAG\_TOTAL\_NUMBER 0x10
- #define AVC\_MEDIA\_ATTR\_FLAG\_GENRE 0x20
- #define AVC\_MEDIA\_ATTR\_FLAG\_PLAYING\_TIME 0x40
- #define AVC\_MEDIA\_ATTR\_FLAG\_ALL 0x7F

# Play status

- #define AVC\_PLAY\_STATUS\_STOPPED 0
- #define AVC PLAY STATUS PLAYING 1
- #define AVC PLAY STATUS PAUSED 2
- #define AVC PLAY STATUS FW SEEK 3
- #define AVC PLAY STATUS REV SEEK 4
- #define AVC\_PLAY\_STATUS\_ERROR 0xFF

## **Notifications**

#define AVC EVENT PLAYBACK STATUS CHANGED 0x01

Change in playback status of the current track.

• #define AVC EVENT TRACK CHANGED 0x02

Change of current track.

#define AVC\_EVENT\_TRACK\_REACHED\_END 0x03

Reached end of a track.

• #define AVC\_EVENT\_TRACK\_REACHED\_START 0x04

Reached start of a track.

• #define AVC\_EVENT\_PLAYBACK\_POS\_CHANGED 0x05

Change in playback position. Returned after the specified playback notification change notification interval.

• #define AVC EVENT BATT STATUS CHANGED 0x06

Change in battery status.

#define AVC EVENT SYSTEM STATUS CHANGED 0x07

Change in system status.

• #define AVC EVENT PLAYER APPLICATION SETTING CHANGED 0x08

Change in player application setting.

#define AVC\_EVENT\_NOW\_PLAYING\_CONTENT\_CHANGED 0x09

The content of the Now Playing list has changed, see 6.9.5.

#define AVC\_EVENT\_AVAILABLE\_PLAYERS\_CHANGED 0x0a

The available players have changed, see 6.9.

• #define AVC\_EVENT\_ADDRESSED\_PLAYER\_CHANGED 0x0b

The Addressed Player has been changed, see 6.9.2.

#define AVC\_EVENT\_UIDS\_CHANGED 0x0c

The UIDs have changed, see 6.10.3.3.

• #define AVC\_EVENT\_VOLUME\_CHANGED 0x0d

The volume has been changed locally on the TG, see 6.13.3.

## **Notifications mask**

- #define AVC EVENT FLAG PLAYBACK STATUS CHANGED 0x0001
- #define AVC\_EVENT\_FLAG\_TRACK\_CHANGED 0x0002
- #define AVC EVENT FLAG TRACK REACHED END 0x0004
- #define AVC EVENT FLAG TRACK REACHED START 0x0008
- #define AVC EVENT FLAG PLAYBACK POS CHANGED 0x0010
- #define AVC EVENT FLAG BATT STATUS CHANGED 0x0020
- #define AVC\_EVENT\_FLAG\_SYSTEM\_STATUS\_CHANGED 0x0040
- #define AVC\_EVENT\_FLAG\_PLAYER\_APPLICATION\_SETTING\_CHANGED 0x0080
- #define AVC\_EVENT\_FLAG\_NOW\_PLAYING\_CONTENT\_CHANGED 0x0100
- #define AVC\_EVENT\_FLAG\_AVAILABLE\_PLAYERS\_CHANGED 0x0200
- #define AVC\_EVENT\_FLAG\_ADDRESSED\_PLAYER\_CHANGED 0x0400
- #define AVC\_EVENT\_FLAG\_UIDS\_CHANGED 0x0800
- #define AVC\_EVENT\_FLAG\_VOLUME\_CHANGED 0x1000
- #define AVC\_EVENT\_FLAG\_ALL 0x1FFF

# Media navigation scope

- #define AVC\_SCOPE\_MEDIA\_PLAYER\_LIST 0x00
- #define AVC\_MEDIA\_PLAYER\_VIRTUAL\_FILESYSTEM 0x01
- #define AVC SEARCH 0x02
- #define AVC\_NOW\_PLAYING 0x03

# 3.16.1 Detailed Description

The Audio/Video Remote Control Profile (AVRCP) defines the features and procedures required in order to ensure interoperability between Bluetooth devices with audio/video control functions in the Audio/Video distribution scenarios.

# 3.16.2 Typedef Documentation

3.16.2.1 typedef void(\* bt\_avrcp\_find\_callback\_fp) (bt\_bool found, void \*param)

Find Controller/Target callback.

This callback is called when search for Controller/Target has finished.

## **Parameters**

found	This can be one of the following values:
	TRUE if Controller/Target was found.
	• FALSE otherwise.
callback_param	A pointer to an arbitrary data set by a call to bt_avrcp_find_target/bt_avrcp_find_controller.

3.16.2.2 typedef void(\* bt\_avrcp\_mgr\_callback\_fp) (bt\_avrcp\_mgr\_t \*mgr, bt\_byte evt, bt\_avrcp\_event\_t \*evt\_param, void \*callback\_param)

# AVRCP application callback.

In order to be notified of various events a consumer of the AVRCP layer has to register a callback function (done with <a href="mailto:bt\_avrcp\_start">bt\_avrcp\_start</a>()). The stack will call that function whenever a new event has been generated.

# **Parameters**

mgr	AVRCP manager.
evt	AVRCP event. The event can be one of the following values:
	<ul> <li>AVRCP_EVT_CONTROL_CHANNEL_CONNECTED This event is generated when a control channel between two AVRCP entities has been established.</li> </ul>
	<ul> <li>AVRCP_EVT_CONTROL_CHANNEL_DISCONNECTED This event is generated when a control channel between two AVRCP entities has been terminated.</li> </ul>
	<ul> <li>AVRCP_EVT_CONTROL_CONNECTION_FAILED This event is generated when a lo- cal device failed to create a control channel between two AVRCP entities.</li> </ul>
	<ul> <li>AVRCP_EVT_BROWSING_CHANNEL_CONNECTED This event is generated when a browsing channel between two AVRCP entities has been established.</li> </ul>
	<ul> <li>AVRCP_EVT_BROWSING_CHANNEL_DISCONNECTED This event is generated when a browsing channel between two AVRCP entities has been terminated.</li> </ul>
	<ul> <li>AVRCP_EVT_BROWSING_CONNECTION_FAILED This event is generated when a local device failed to create a browsing channel between two AVRCP entities.</li> </ul>
	<ul> <li>AVRCP_EVT_SEARCH_COMPLETED This event is generated when a local device completed searching for nearby targets.</li> </ul>
	<ul> <li>AVRCP_EVT_PANEL_RESPONSE_RECEIVED This event is generated when a local device received a response to a PASS THROUGH command.</li> </ul>
	<ul> <li>AVRCP_EVT_COMPANY_ID_LIST_RECEIVED This event is generated when a local device received a response to a "get company id" request.</li> </ul>
	<ul> <li>AVRCP_EVT_EVENT_ID_LIST_RECEIVED This event is generated when a local device received a response to a "get supported events" request.</li> </ul>
	<ul> <li>AVRCP_EVT_PLAYER_SETTING_ATTRIBUTES_RECEIVED This event is generated when a local device received a response to a "get supported player setting attributes" request.</li> </ul>
	<ul> <li>AVRCP_EVT_PLAYER_SETTING_VALUES_RECEIVED This event is generated when a local device received a response to a "get player setting attribute values" request.</li> </ul>
	<ul> <li>AVRCP_EVT_PLAYER_CURRENT_SETTING_VALUES_RECEIVED This event is generated when a local device received a response to a "get current player setting attribute values" request.</li> </ul>
	<ul> <li>AVRCP_EVT_SET_PLAYER_SETTING_VALUES_COMPLETED This event is generated when a local device received a response to a "set player setting attribute values" request.</li> </ul>
	<ul> <li>AVRCP_EVT_PLAYER_SETTING_ATTRIBUTES_TEXT_RECEIVED This event is generated when a local device received a response to a "get player setting attributes displayable text" request.</li> </ul>
	<ul> <li>AVRCP_EVT_PLAYER_SETTING_VALUES_TEXT_RECEIVED This event is generated when a local device received a response to a "get player setting attribute values displayable text" request.</li> </ul>
	<ul> <li>AVRCP_EVT_INFORM_DISPLAYABLE_CHARACTER_SET_COMPLETED This event is generated when a local device received a response to a "inform displayable character set" request.</li> </ul>
	<ul> <li>AVRCP_EVT_INFORM_BATTERY_STATUS_OF_CT_COMPLETED This event is generated when a local device received a response to a "inform battery status" request.</li> </ul>
	• AVRCP_EVT_GET_ELEMENT_ATTRIBUTES_ <b>প্রস্থার্থা V</b> -2015 PFAR AV দার্ভার পর্যার্থার প্রতিষ্ঠিত when a local device received a response to a "get media element attributes" request.

• AVRCP\_EVT\_GET\_PLAY\_STATUS\_RECEIVED This event is generated when a local

## 3.16.3 Function Documentation

3.16.3.1 bt\_bool bt\_avrcp\_add\_to\_now\_playing ( bt\_avrcp\_channel\_t \* channel, bt\_byte scope, bt\_av\_element\_id\_t \* element\_id, bt\_uint counter )

Add to "now playing" list.

This function adds a media element specified by <code>element\_id</code> to the "now playing" list on the target.

#### **Parameters**

channel	AVRCP channel.
scope	The scope in which the element_id is valid. This value can be on the following values:
	AVC_SCOPE_MEDIA_PLAYER_LIST
	AVC_MEDIA_PLAYER_VIRTUAL_FILESYSTEM
	• AVC_SEARCH
	AVC_NOW_PLAYING
element_id	UID of the media element to be added to the "now playing" list.
counter	UID counter.

## Returns

- TRUE if the function succeeds.
- FALSE otherwise.

# 3.16.3.2 bt\_bool bt\_avrcp\_cancel\_find (void)

Cancel finding Targets.

This function stops AVRCP layer looking for targets on nearby devices. As a result of this operation the AVRCP\_← EVT\_SEARCH\_COMPLETED event will be generated.

#### Returns

- TRUE if the function succeeds.
- FALSE otherwise. the callback is not called in this case.

3.16.3.3 void bt\_avrcp\_cancel\_listen ( bt\_avrcp\_channel\_t \* channel )

Cancel listening for incoming connections.

This function stops listening for incoming connections on a specified channel.

## **Parameters**

channel	AVRCP channel.

- · TRUE if the function succeeds.
- FALSE otherwise.

3.16.3.4 bt\_bool bt\_avrcp\_connect ( bt\_avrcp\_channel\_t \* channel, bt\_bdaddr\_t \* remote\_address )

Connect to a remote device.

This function establishes a connection to a remote device specified by the remote\_address. If connection cannot be initiated for some reason, for example, there is not enough resources, it returns FALSE and no events are generated. Otherwise the result of an attempt to connect to the remote device is reported via the AVRCP callback. The events generated will either be AVRCP\_EVT\_CONTROL\_CHANNEL\_CONNECTED or AVRCP\_E VT\_CONTROL\_CONNECTION\_FAILED.

#### **Parameters**

channel	AVRCP channel.
remote_address	The address of a remote device.

#### Returns

- TRUE if connection establishment has been started.
- FALSE otherwise.

 $3.16.3.5 \quad bt\_avrcp\_channel\_t*\ bt\_avrcp\_create\_channel\ (\ bt\_avrcp\_mgr\_t*\ \textit{mgr,}\ bt\_bool\ \textit{create\_browsing\_channel}\ )$ 

Allocate AVRCP channel.

This function allocates a new incoming AVRCP channel. The channel is intended to be used to accept a connection from a remote device. There can be only one incoming channel.

#### **Parameters**

mgr	AVRCP manager.
create_←	Defines weather a browsing channel will be created.
browsing_←	
channel	

# Returns

- A pointer to the new AVRCP channel if the function succeeds.
- NULL otherwise.

3.16.3.6 bt\_avrcp\_channel\_t\* bt\_avrcp\_create\_outgoing\_channel ( bt\_avrcp\_mgr\_t \* mgr, bt\_bool create\_browsing\_channel )

Allocate AVRCP channel.

This function allocates a new outgoing AVRCP channel. The channel is intended to be used to create a connection to a remote device. There can be multiple outgoing channels.

#### **Parameters**

mgr	AVRCP manager.
create_ <i>←</i>	Defines weather a browsing channel will be created.
browsing_←	
channel	

- · A pointer to the new AVRCP channel if the function succeeds.
- NULL otherwise.

3.16.3.7 bt\_bool bt\_avrcp\_destroy\_channel ( bt\_avrcp\_channel\_t \* channel )

Destroy AVRCP channel.

This function frees memory used by the channel. The channel has to exist and be in the "idle" state for this function to succeed. I.e. the channel has to be disconnected before this function can be called.

#### **Parameters**

channel	AVRCP channel.	

#### Returns

- TRUE if the function succeeds.
- FALSE otherwise.

3.16.3.8 bt\_bool bt\_avrcp\_disconnect ( bt\_avrcp\_channel\_t \* channel )

Disconnect from a remote device.

This function closes a connection to a remote device.

#### **Parameters**

channel AVRCP channel.	
------------------------	--

## Returns

- TRUE if disconnection has been started.
- FALSE otherwise. No events will be generated.
- 3.16.3.9 bt\_bool bt\_avrcp\_find\_controller ( bt\_bdaddr\_t \* deviceAddress, bt\_avrcp\_find\_callback\_fp callback, bt\_sdp\_client\_callback\_fp client\_callback, void \* callback\_param )

## Find Controller.

This function looks for a controller on a remote device specified by  ${\tt deviceAddress}$ .

# Parameters

deviceAddress	The address of a remote device.
callback	The callback function that will be called when search has completed.
client_callback	The optional callback function that an application can set if it wants to be notified of state changes of the SDP client. The evt parameter of the callback can be one of the following values:
	SDP_CLIENT_STATE_IDLE
	SDP_CLIENT_STATE_CONNECTING
	SDP_CLIENT_STATE_DISCONNECTING
	SDP_CLIENT_STATE_CONNECTED
callback_param	A pointer to arbitrary data to be passed to the callback and client_callback call-
	backs.

- TRUE if the function succeeds.
- FALSE otherwise. the callback is not called in this case.

3.16.3.10 bt\_bool bt\_avrcp\_find\_target ( bt\_bdaddr\_t \* deviceAddress, bt\_avrcp\_find\_callback\_fp callback, bt\_sdp\_client\_callback\_fp client\_callback, void \* callback\_param )

# Find Target.

This function looks for a target on a remote device specified by <code>deviceAddress</code>.

#### **Parameters**

deviceAddress	The address of a remote device.
callback	The callback function that will be called when search has completed.
client_callback	The optional callback function that an application can set if it wants to be notified of state changes of the SDP client. The ${\tt evt}$ parameter of the callback can be one of the following values:
	SDP_CLIENT_STATE_IDLE
	SDP_CLIENT_STATE_CONNECTING
	SDP_CLIENT_STATE_DISCONNECTING
	SDP_CLIENT_STATE_CONNECTED
callback_param	A pointer to arbitrary data to be passed to the callback and client_callback call-
	backs.

## Returns

- TRUE if the function succeeds.
- FALSE otherwise. the callback is not called in this case.
- 3.16.3.11 bt\_bool bt\_avrcp\_find\_targets ( bt\_byte search\_length )

## Find Targets.

This function looks for targets on nearby devices. The AVRCP\_EVT\_SEARCH\_COMPLETED event is generated when the search has completed.

# **Parameters**

search_length	The amount of time the search will be performed for.

# Returns

- TRUE if the function succeeds.
- FALSE otherwise. the callback is not called in this case.
- 3.16.3.12 bt\_byte bt\_avrcp\_get\_browsing\_channel\_state ( bt\_avrcp\_channel\_t \* channel )

Get AVCTP browsing channel state.

This function returns status of the AVCTP browsing channel.

## **Parameters**

channel	AVRCP channel.

#### Returns

Returns of the following values:

- AVCTP\_CHANNEL\_STATE\_FREE
- AVCTP\_CHANNEL\_STATE\_IDLE
- AVCTP\_CHANNEL\_STATE\_CONNECTING
- AVCTP\_CHANNEL\_STATE\_CONNECTED
- AVCTP\_CHANNEL\_STATE\_DISCONNECTING

3.16.3.13 bt\_bdaddr\_t\* bt\_avrcp\_get\_channel\_remote\_address ( bt\_avrcp\_channel\_t \* channel )

Get channel's remote BT address.

This function returns the address of the remote device associated with the channel.

#### **Parameters**

channel AVRCP channel.

#### Returns

- The address of the remote device if channel is connected.
- · NULL otherwise.

3.16.3.14 bt\_bool bt\_avrcp\_get\_company\_id\_list ( bt\_avrcp\_channel\_t \* channel )

Get Company ID list.

This function requests a list of company id's supported by the remote device

# **Parameters**

channel AVRCP channel.

## Returns

- TRUE if the function succeeds.
- FALSE otherwise.

 $3.16.3.15 \quad bt\_byte\ bt\_avrcp\_get\_control\_channel\_state\ (\ bt\_avrcp\_channel\_t*\ \textit{channel}\ )$ 

Get AVCTP control channel state.

This function returns status of the AVCTP control channel.

#### **Parameters**

channel AVRCP channel.

## Returns

Returns of the following values:

- AVCTP\_CHANNEL\_STATE\_FREE
- · AVCTP CHANNEL STATE IDLE
- AVCTP\_CHANNEL\_STATE\_CONNECTING
- AVCTP\_CHANNEL\_STATE\_CONNECTED
- AVCTP\_CHANNEL\_STATE\_DISCONNECTING

3.16.3.16 bt\_bool bt\_avrcp\_get\_current\_player\_application\_setting\_value ( bt\_avrcp\_channel\_t \* channel, bt\_av\_player\_setting\_current\_values\_t \* response\_buffer )

Get current player setting values.

This function requests a list of current set values for the player application on the target. The list of attribute ids whose values have to be returned is passed via the response\_buffer parameter. The caller has to set bt \_\_av\_player\_setting\_current\_values\_t::setting\_id\_list to a list of player setting attribute ids, bt\_av\_player\_setting\_current\_values\_t::setting\_value \_\_current\_values\_t::setting\_value \_\_id\_list to a buffer where returned values will be stored.

## **Parameters**

channel	AVRCP channel.
response_buffer	Pointer to bt_av_player_setting_current_values_t structure initialized as stated above.

## Returns

- TRUE if the function succeeds.
- FALSE otherwise.

3.16.3.17 bt\_bool bt\_avrcp\_get\_element\_attributes ( bt\_avrcp\_channel\_t \* channel, bt\_av\_element\_id\_t \* element\_id, bt\_uint attr\_mask )

Get media element attributes.

This function requests the attributes of the element specified with  $element\_id$ .

#### Note

Currently element\_id is ignored. The AVRCP specification defines that only UID 0 can be used to return the attributes of the current track.

# **Parameters**

channel	AVRCP channel.
element_id	UID of the media element whose attributes are requested.
attr_mask	Bitmask that defines which attributes are requested. This value can be a combination of the
	following values:
	AVC_MEDIA_ATTR_FLAG_TITLE
	AVC_MEDIA_ATTR_FLAG_ARTIST
	AVC_MEDIA_ATTR_FLAG_ALBUM
	AVC_MEDIA_ATTR_FLAG_NUMBER
	AVC_MEDIA_ATTR_FLAG_TOTAL_NUMBER
	AVC_MEDIA_ATTR_FLAG_GENRE
	AVC_MEDIA_ATTR_FLAG_PLAYING_TIME
	AVC_MEDIA_ATTR_FLAG_ALL

- TRUE if the function succeeds.
- FALSE otherwise.

3.16.3.18 bt\_hci\_conn\_state\_p\* bt\_avrcp\_get\_hci\_connection ( bt\_avrcp\_channel\_t \* channel )

Get HCI connection for a channel.

This function returns a pointer to a structure that describes an HCI connection a channel is open on. The return value can be used to call various function from the HCI layer. For example, if an app wants to force disconnection from a remote device it can call bt\_hci\_disconnect.

#### **Parameters**

channel	AVRCP channel.

#### Returns

- Pointer to a structure that describes an HCI connection if the function succeeds.
- NULL otherwise. The function fails only if a channel specified by the channel parameter
- does not exist or there is no HCI connection between local and remote devices associated with the channel.

3.16.3.19 bt\_avrcp\_mgr\_t\* bt\_avrcp\_get\_mgr ( void )

Return a pointer to an instance of the AVRCP manager.

This function returns a pointer to an instance of the AVRCP manager. There is only one instance of the manager allocated by the stack.

3.16.3.20 bt\_bool bt\_avrcp\_get\_play\_status ( bt\_avrcp\_channel\_t \* channel, bt\_uint repeat\_interval )

Get playback status.

This function requests the status of the currently playing media at the target.

#### **Parameters**

channel	AVRCP channel.
repeat_interval	Interval in milliseconds at which AVRCP polls the target for playback status. If 0 is passed
	polling is stopped.

#### Returns

- TRUE if the function succeeds.
- FALSE otherwise.

```
3.16.3.21 bt_bool bt_avrcp_get_player_application_setting_attr_text ( bt_avrcp_channel_t * channel, bt_av_player_settings_text_t * response_buffer )
```

Get player setting attribute text.

This function requests the target device to provide supported player application setting attribute displayable text for the provided player application setting attributes. The list of attribute ids whose displayable text have to be returned is passed via the response\_buffer parameter. The caller has to set bt\_av\_player\_settings\_text\_t::setting\_cid\_list to a list of player setting attribute ids, bt\_av\_player\_settings\_text\_t::count to the number of entries in the list, bt\_av\_player\_settings\_text\_t::setting\_text\_list to a buffer where returned values will be stored.

#### **Parameters**

channel	AVRCP channel.
response_buffer	Pointer to bt_av_player_settings_text_t structure initialized as stated above.

#### Returns

- TRUE if the function succeeds.
- FALSE otherwise.

3.16.3.22 bt\_bool bt\_avrcp\_get\_player\_application\_setting\_value\_text ( bt\_avrcp\_channel\_t \* channel, bt\_byte attr\_id, bt\_av\_player\_setting\_values\_text\_t \* response\_buffer )

Get player setting value text.

This function request the target device to provide target supported player application setting value displayable text for the provided player application setting attribute values. The list of attribute ids whose value displayable text have to be returned is passed via the response\_buffer parameter. The caller has to set bt\_av\_player\_setting\_  $\leftarrow$  values\_text\_t::setting\_value\_id\_list to a list of player setting attribute value ids, bt\_av\_player\_setting\_values\_text t::count to the number of entries in the list, bt\_av\_player\_setting\_values\_text\_t::setting\_value\_text\_list to a buffer where returned values will be stored.

#### **Parameters**

channel	AVRCP channel.
response_buffer	Pointer to bt_av_player_setting_values_text_t structure initialized as stated above.

#### Returns

- TRUE if the function succeeds.
- FALSE otherwise.

3.16.3.23 bt\_bool bt\_avrcp\_get\_subuint\_info ( bt\_avrcp\_channel\_t \* channel )

Get subunit info.

This function is used to request subunit info from the target.

#### **Parameters**

channel AVRCP channel.
------------------------

# Returns

- TRUE if the function succeeds.
- FALSE otherwise.

3.16.3.24 bt\_bool bt\_avrcp\_get\_supported\_event\_id\_list ( bt\_avrcp\_channel\_t \* channel )

Get supported events.

This function requests a list of events supported by the remote device

#### **Parameters**

channel	AVRCP channel.	]
---------	----------------	---

## Returns

- TRUE if the function succeeds.
- FALSE otherwise.

# 3.16.3.25 bt\_bool bt\_avrcp\_get\_unit\_info ( bt\_avrcp\_channel\_t \* channel )

Get unit info.

This function is used to request unit info from the target.

#### **Parameters**

channel AVRCP channel.
------------------------

## Returns

- TRUE if the function succeeds.
- FALSE otherwise.

# 3.16.3.26 bt\_bool bt\_avrcp\_inform\_battery\_status ( bt\_avrcp\_channel\_t \* channel, bt\_byte status )

Inform controller's battery status.

This function is used to inform the target about the controller's battery status.

## **Parameters**

channel	AVRCP channel.
status	Battery status. This can be one of the following values:
	AVC_BATTERY_STATUS_NORMAL
	AVC_BATTERY_STATUS_WARNING
	AVC_BATTERY_STATUS_CRITICAL
	AVC_BATTERY_STATUS_EXTERNAL
	AVC_BATTERY_STATUS_FULL_CHARGE

## Returns

- TRUE if the function succeeds.
- FALSE otherwise.

# 3.16.3.27 bt\_bool bt\_avrcp\_inform\_displayable\_character\_set ( bt\_avrcp\_channel\_t \* channel, bt\_uint \* charset\_list, bt\_byte charset\_count )

Inform displayable character set.

This function informs the list of character set supported by the controller to the target.

#### **Parameters**

channel	AVRCP channel.
charset_list	List of displayable character sets.
charset_count	Number of entries in the list.

## Returns

- TRUE if the function succeeds.
- FALSE otherwise.

3.16.3.28 void bt\_avrcp\_init\_controller (void)

Initialize AVRCP to be used in controller mode.

This function initializes the AVRCP layer of the stack in controller mode. It must be called prior to any other AVRCP function can be called.

3.16.3.29 void bt\_avrcp\_init\_target ( bt\_ulong company\_id, bt\_uint supported\_events )

Initialize AVRCP to be used in target mode.

This function initializes the AVRCP layer of the stack in target mode. It must be called prior to any other AVRCP function can be called.

# **Parameters**

company_id	The 24-bit unique ID obtained from the IEEE Registration Authority Committee. If the vendor
	of a TG device does not have the unique ID, the value 0xFFFFF may be used.
supported_←	Bitmask that specifies events supported by the target. This value can be a combination of the
events	following values:
	AVC_EVENT_FLAG_PLAYBACK_STATUS_CHANGED
	AVC_EVENT_FLAG_TRACK_CHANGED
	AVC_EVENT_FLAG_TRACK_REACHED_END
	AVC_EVENT_FLAG_TRACK_REACHED_START
	AVC_EVENT_FLAG_PLAYBACK_POS_CHANGED
	AVC_EVENT_FLAG_BATT_STATUS_CHANGED
	AVC_EVENT_FLAG_SYSTEM_STATUS_CHANGED
	AVC_EVENT_FLAG_PLAYER_APPLICATION_SETTING_CHANGED
	AVC_EVENT_FLAG_NOW_PLAYING_CONTENT_CHANGED
	AVC_EVENT_FLAG_AVAILABLE_PLAYERS_CHANGED
	AVC_EVENT_FLAG_ADDRESSED_PLAYER_CHANGED
	AVC_EVENT_FLAG_UIDS_CHANGED
	AVC_EVENT_FLAG_VOLUME_CHANGED

3.16.3.30 bt\_bool bt\_avrcp\_list\_player\_application\_setting\_attributes ( bt\_avrcp\_channel\_t \* channel, bt\_av\_player\_settings\_t \* response\_buffer )

Get supported player setting attributes.

This function request the target device to provide target supported player application setting attributes. The list of attribute ids is stored in the setting\_id\_list member of the response\_buffer parameter. The caller has to set bt\_av\_player\_settings\_t::setting\_id\_list to a buffer where returned values will be stored and bt\_av\_player\_settings t::count to the number of entries in the list.

#### **Parameters**

channel	AVRCP channel.
response_buffer	Pointer to bt_av_player_settings_t structure initialized as stated above.

#### Returns

- TRUE if the function succeeds.
- FALSE otherwise.

```
3.16.3.31 bt_bool bt_avrcp_list_player_application_setting_values ( bt_avrcp_channel_t * channel, bt_byte attr_id, bt_av_player_setting_values_t * response_buffer )
```

Get player setting attribute values.

This function requests the target device to list the set of possible values for the requested player application setting attribute. The list of attribute value ids is stored in the setting\_value\_id\_list member of the response\_buffer parameter. The caller has to set bt\_av\_player\_setting\_values\_t::setting\_value\_id\_list to a buffer where returned values will be stored and bt\_av\_player\_setting\_values\_t::count to the number of entries in the list.

#### **Parameters**

channel	AVRCP channel.
response_buffer	Pointer to bt_av_player_setting_values_t structure initialized as stated above.

#### Returns

- TRUE if the function succeeds.
- FALSE otherwise.

3.16.3.32 bt\_bool bt\_avrcp\_listen ( bt\_avrcp\_channel\_t \* channel )

Listen for incoming connections.

This function enables incoming connections on the specified AVRCP channel.

## **Parameters**

channel	AVRCP channel.

- TRUE if the function succeeds.
- FALSE otherwise.

3.16.3.33 bt\_bool bt\_avrcp\_play\_item ( bt\_avrcp\_channel\_t \* channel, bt\_byte scope, bt\_av\_element\_id\_t \* element\_id, bt\_uint counter )

Play media item.

This function starts playing an item indicated by the UID.

## **Parameters**

channel	AVRCP channel.
scope	The scope in which the element_id is valid. This value can be on the following values:
	AVC_SCOPE_MEDIA_PLAYER_LIST
	AVC_MEDIA_PLAYER_VIRTUAL_FILESYSTEM
	• AVC_SEARCH
	AVC_NOW_PLAYING
element_id	UID of the media element to be played.
counter	UID counter.

## Returns

- TRUE if the function succeeds.
- FALSE otherwise.

3.16.3.34 bt\_bool bt\_avrcp\_register\_notification ( bt\_avrcp\_channel\_t \* channel, bt\_byte event\_id, bt\_ulong playback\_interval )

# Register notification.

This function registers with the target to receive notifications asynchronously based on specific events occurring.

# **Parameters**

channel	AVRCP channel.
event_id	Event ld. This value can be one of the following values:
	AVC_EVENT_PLAYBACK_STATUS_CHANGED
	AVC_EVENT_TRACK_CHANGED
	AVC_EVENT_TRACK_REACHED_END
	AVC_EVENT_TRACK_REACHED_START
	AVC_EVENT_PLAYBACK_POS_CHANGED
	AVC_EVENT_BATT_STATUS_CHANGED
	AVC_EVENT_SYSTEM_STATUS_CHANGED
	AVC_EVENT_PLAYER_APPLICATION_SETTING_CHANGED
	AVC_EVENT_NOW_PLAYING_CONTENT_CHANGED
	AVC_EVENT_AVAILABLE_PLAYERS_CHANGED
	AVC_EVENT_ADDRESSED_PLAYER_CHANGED
	AVC_EVENT_UIDS_CHANGED
	AVC_EVENT_VOLUME_CHANGED

playback_←	The time interval (in seconds) at which the change in playback position will be notified. Appli-
interval	cable for AVC_EVENT_PLAYBACK_POS_CHANGED event.

## Returns

- TRUE if the function succeeds.
- FALSE otherwise.

3.16.3.35 bt\_bool bt\_avrcp\_register\_notifications ( bt\_avrcp\_channel\_t \* channel, bt\_uint event\_mask )

# Register notifications.

This function registers with the target to receive notifications asynchronously based on specific events occurring. This function is used to register multiple notifications with one call.

## Note

This function cannot be used to register for AVC\_EVENT\_PLAYBACK\_POS\_CHANGED event.

## **Parameters**

channel	AVRCP channel.
event_mask	Bitmask that specifies which events to register for. This value can be a combination of the following values:
	AVC_EVENT_FLAG_PLAYBACK_STATUS_CHANGED
	AVC_EVENT_FLAG_TRACK_CHANGED
	AVC_EVENT_FLAG_TRACK_REACHED_END
	AVC_EVENT_FLAG_TRACK_REACHED_START
	AVC_EVENT_FLAG_BATT_STATUS_CHANGED
	AVC_EVENT_FLAG_SYSTEM_STATUS_CHANGED
	<ul> <li>AVC_EVENT_FLAG_PLAYER_APPLICATION_SETTING_CHANGED</li> </ul>
	<ul> <li>AVC_EVENT_FLAG_NOW_PLAYING_CONTENT_CHANGED</li> </ul>
	<ul> <li>AVC_EVENT_FLAG_AVAILABLE_PLAYERS_CHANGED</li> </ul>
	<ul> <li>AVC_EVENT_FLAG_ADDRESSED_PLAYER_CHANGED</li> </ul>
	AVC_EVENT_FLAG_UIDS_CHANGED
	AVC_EVENT_FLAG_VOLUME_CHANGED
	AVC_EVENT_FLAG_ALL

# Returns

- TRUE if the function succeeds.
- FALSE otherwise.

3.16.3.36 bt\_bool bt\_avrcp\_send\_button\_click ( bt\_avrcp\_channel\_t \* channel, bt\_byte button\_id )

Send AV/C Panel Subunit "click" PASS THROUGH command.

This function is used to send a button click. Two PATH THROUGTH commands are sent. The first command has button state set to AVC\_PANEL\_BUTTON\_PRESSED. The second command gas button state set to AVC\_PAN  $\leftarrow$  EL\_BUTTON\_RELEASED

#### **Parameters**

channel	AVRCP channel.
button_id	Operation Id. This value can be on of the AVC_PANEL_OPID constants

## Returns

- TRUE if the function succeeds.
- FALSE otherwise.

3.16.3.37 bt\_bool bt\_avrcp\_send\_cmd ( bt\_avrcp\_channel\_t \* channel, bt\_av\_command\_t \* command )

Send AVRCP command.

This function sends a command to the remote device.

#### **Parameters**

channel	AVRCP channel.
command	Command to be sent.

## Returns

- TRUE if the function succeeds.
- FALSE otherwise.

3.16.3.38 bt\_bool bt\_avrcp\_send\_panel\_control ( bt\_avrcp\_channel\_t \* channel, bt\_byte opid, bt\_byte button\_state )

Send AV/C Panel Subunit "control" PASS THROUGH command.

This function is used to send AV/C Panel Subunit PASS THROUGH command with command type set to AVC $_{\leftarrow}$  CTYPE $_{\leftarrow}$ CONTROL.

# Parameters

channel	AVRCP channel.
opid	Operation Id. This value can be on of the AVC_PANEL_OPID constants
button_state	Button state. This can be on of the following values:
	AVC_PANEL_BUTTON_PRESSED     AVC_PANEL_BUTTON_RELEASED

#### Returns

- TRUE if the function succeeds.
- FALSE otherwise.

3.16.3.39 bt\_bool bt\_avrcp\_send\_press\_panel\_control ( bt\_avrcp\_channel\_t \* channel, bt\_byte opid )

Send AV/C Panel Subunit "pressed" PASS THROUGH command.

This function is used to send AV/C Panel Subunit PASS THROUGH command with button state set to AVC\_PA $\hookleftarrow$  NEL BUTTON PRESSED.

#### **Parameters**

channel	AVRCP channel.	]
opid	Operation Id. This value can be on of the AVC_PANEL_OPID constants	1

#### Returns

- TRUE if the function succeeds.
- FALSE otherwise.

3.16.3.40 bt\_bool bt\_avrcp\_send\_release\_panel\_control( bt\_avrcp\_channel\_t \* channel, bt\_byte opid)

Send AV/C Panel Subunit "released" PASS THROUGH command.

This function is used to send AV/C Panel Subunit PASS THROUGH command with button state set to AVC\_PA⇔ NEL\_BUTTON\_RELEASED.

## **Parameters**

channel	AVRCP channel.
opid	Operation Id. This value can be on of the AVC_PANEL_OPID constants

#### Returns

- TRUE if the function succeeds.
- FALSE otherwise.

3.16.3.41 bt\_bool bt\_avrcp\_send\_simple\_panel\_cmd ( bt\_avrcp\_channel\_t \* channel, bt\_byte ctype, bt\_byte opid, bt\_byte button\_state )

Send AV/C Panel Subunit PASS THROUGH command.

This function is used to send AV/C Panel Subunit PASS THROUGH command to the target.

## **Parameters**

channel	AVRCP channel.
ctype	Command type. This value can be on of the following values:
	AVC_CTYPE_CONTROL 0
	AVC_CTYPE_STATUS 1
	AVC_CTYPE_SPECIFIC_IQUIRY 2
	AVC_CTYPE_NOTIFY 3
	AVC_CTYPE_GENERAL_INQUORY 4
opid	Operation Id. This value can be on of the AVC_PANEL_OPID constants
button_state	Button state. This can be on of the following values:
	AVC_PANEL_BUTTON_PRESSED     AVC_PANEL_BUTTON_RELEASED

- TRUE if the function succeeds.
- FALSE otherwise.

3.16.3.42 bt\_bool bt\_avrcp\_set\_absolute\_volume ( bt\_avrcp\_channel\_t \* channel, bt\_byte volume )

Set absolute volume.

This function is used to set an absolute volume to be used by the rendering device.

#### **Parameters**

Cl		AVRCP channel.
ν	<i>rolume</i>	Volume

#### Returns

- TRUE if the function succeeds.
- FALSE otherwise.

3.16.3.43 bt\_bool bt\_avrcp\_set\_addressed\_player ( bt\_avrcp\_channel\_t \* channel, bt\_uint player\_id )

Set addressed player.

This function is used to inform the target of which media player the controller wishes to control.

#### **Parameters**

channel	AVRCP channel.
player_id	Player Id.

#### Returns

- TRUE if the function succeeds.
- FALSE otherwise.

3.16.3.44 bt\_bool bt\_avrcp\_set\_player\_application\_setting\_value ( bt\_avrcp\_channel\_t \* channel, bt\_byte \* attr\_id\_list, bt\_byte \* attr\_value\_list, bt\_byte attr\_id\_count )

Set player setting attribute values.

This function requests to set the player application setting list of player application setting values on the target device for the corresponding defined list of setting attributes. for the requested player application setting attribute. The list of attribute value ids is stored in the setting\_value\_id\_list member of the response\_buffer parameter. The caller has to set bt\_av\_player\_setting\_values\_t::setting\_value\_id\_list to a buffer where returned values will be stored and bt\_av\_player\_setting\_values\_t::count to the number of entries in the list.

#### **Parameters**

channel	AVRCP channel.
attr_id_list	List of setting attribute ids.
attr_value_list	List of setting attribute value ids.
attr_id_count	The number of entries in both lists.

- TRUE if the function succeeds.
- FALSE otherwise.

3.16.3.45 void bt\_avrcp\_start ( bt\_avrcp\_mgr\_t \* mgr, bt\_avrcp\_mgr\_callback\_fp callback, void \* callback\_param )

# Start the AVRCP layer.

In order to be notified of various events a consumer of the AVRCP layer has to register a callback function. The stack will call the callback function whenever a new event has been generated passing the code of the event as the second parameter. bt\_avrcp\_start() stores pointers to the callback and callback\_param in the bt\_avrcp — mgr\_t structure.

#### **Parameters**

mgr	AVRCP manager.
callback	The callback function that will be called when the AVRCP generates an event.
callback_param	A pointer to arbitrary data to be passed to the callback callback.

3.16.3.46 bt\_bool bt\_avrcp\_tg\_send\_element\_attributes ( bt\_avrcp\_channel\_t \* channel, bt\_byte tran\_id, bt\_av\_element\_attribute\_t \* attrs, bt\_byte attr\_count )

Send media element attributes.

This function is used to send the system status when AVRCP is running in target mode. If there are active connections with controllers, the ones that registered for AVC\_EVENT\_SYSTEM\_STATUS\_CHANGED event will be notified.

#### **Parameters**

status	Battery status

3.16.3.47 void bt\_avrcp\_tg\_set\_absolute\_volume ( bt\_byte volume )

Set absolute volume.

This function is used to set the absolute volume when AVRCP is running in target mode. If there are active connections with controllers, the ones that registered for AVC\_EVENT\_VOLUME\_CHANGED event will be notified.

# Parameters

track id	Track Id
<del>-</del>	
song_position	The position of the current track.

3.16.3.48 void bt\_avrcp\_tg\_set\_battery\_status ( bt\_byte status )

Set battery status.

This function is used to set the battery status when AVRCP is running in target mode. If there are active connections with controllers, the ones that registered for AVC\_EVENT\_BATT\_STATUS\_CHANGED event will be notified.

## **Parameters**

status	Battery status

3.16.3.49 void bt\_avrcp\_tg\_set\_current\_track ( bt\_av\_element\_id\_t \* track\_id, bt\_ulong song\_length, bt\_ulong song\_position )

Set current track.

This function is used to set the current track when AVRCP is running in target mode. If there are active connections with controllers, the ones that registered for AVC EVENT TRACK CHANGED event will be notified.

#### **Parameters**

track_id	Track Id
song_length	The length of the current track.
song_position	The position of the current track.

3.16.3.50 void bt\_avrcp\_tg\_set\_play\_status ( bt\_ulong song\_length, bt\_ulong song\_position, bt\_byte play\_status )

Set playback status.

This function is used to set the playback status when AVRCP is running in target mode. If there are active connections with controllers, the ones that registered for AVC\_EVENT\_PLAYBACK\_STATUS\_CHANGED event will be notified.

## **Parameters**

song_length	The length of the current track.
song_position	The position of the current track.
play_status	Playback status. This value can be one of the following values:
	• AVC_PLAY_STATUS_STOPPED
	AVC_PLAY_STATUS_PLAYING
	AVC_PLAY_STATUS_PAUSED
	AVC_PLAY_STATUS_FW_SEEK
	AVC_PLAY_STATUS_REV_SEEK

3.16.3.51 void bt\_avrcp\_tg\_set\_system\_status ( bt\_byte status )

Set system status.

This function is used to set the system status when AVRCP is running in target mode. If there are active connections with controllers, the ones that registered for AVC\_EVENT\_SYSTEM\_STATUS\_CHANGED event will be notified.

# **Parameters**

status	Battery status
	,

3.17 Configuration 181

# 3.17 Configuration

This module describes parameters used to configure AVRCP layer.

#### **Macros**

#define AVRCP\_MAX\_CHANNELS

Maximum number of remote devices a local device can be connected to.

• #define AVRCP MAX CMD BUFFERS 1

Maximum number of command buffers.

#define AVRCP\_MAX\_CMD\_PARAM\_LEN 512

Maximum length of command parameters.

• #define AVRCP\_MAX\_SEARCH\_RESULTS 7

Maximum number of devices to find.

• #define AVRCP\_MAX\_DEVICE\_NAME\_LEN 20

Maximum length of device name.

#define AVRCP\_CMD\_TIMEOUT 10000

Command timeout.

## 3.17.1 Detailed Description

This module describes parameters used to configure AVRCP layer.

dotstack is customized using a configuration file. The configuration file tailors the dotstack to the application being built. It has to have the structure shown below.

```
#include "cdbt/bt/bt std.h"
// HCI, L2CAP and SDP must always be present
// HCI configuration parameters
#define HCI_MAX_CMD_BUFFERS
#define HCI_MAX_DATA_BUFFERS
#define HCI_MAX_HCI_CONNECTIONS
#define HCI_RX_BUFFER_LEN
#define HCI_TX_BUFFER_LEN
#define HCI_L2CAP_BUFFER_LEN
                                          . . .
#define HCI_MAX_CMD_PARAM_LEN
// L2CAP configuration parameters
#define L2CAP_MAX_CMD_BUFFERS
#define L2CAP_MAX_FRAME_BUFFERS
                                         . . .
#define L2CAP_MAX_PSMS
#define L2CAP_MAX_CHANNELS
\ensuremath{//} SDP configuration parameters
#define SDP_MAX_SEARCH_RESULT_LEN
#define SDP_MAX_SEARCH_RESULT_LEN ...
#define SDP_MAX_ATTRIBUTE_RESULT_LEN ...
// Depending on protocols and profiles used below goes configuration parameters
// for each used module. E.g., to use and configure AVRCP,
// the following values must be defined:
#define BT INCLUDE AVCTP
                                         // tells dotstack to compile in AVCTP support
#define AVCTP_MAX_CHANNELS
#define AVCTP_MAX_TRANSPORT_CHANNELS ...
#define AVCTP_MAX_RX_MESSAGE_LEN
#define AVCTP_MAX_MESSAGE_BUFFERS
#define BT INCLUDE AVRCP
                                         // tells dotstack to compile in AVRCP support
#define AVRCP_MAX_CHANNELS
                                         . . .
#define AVRCP_MAX_CMD_BUFFERS
#define AVRCP_MAX_CMD_PARAM_LEN
#define AVRCP_MAX_SEARCH_RESULTS
#define AVRCP_MAX_DEVICE_NAME_LEN
#define AVRCP_CMD_TIMEOUT
#include "cdbt/bt/bt_oem_config.h"
```

## 3.17.2 Macro Definition Documentation

## 3.17.2.1 #define AVRCP\_CMD\_TIMEOUT 10000

Command timeout.

This parameter defines the amount of time in milliseconds AVRCP waits for a response to a request. If not defined the default value of 10000 (10 secconds) is used.

## 3.17.2.2 #define AVRCP\_MAX\_CHANNELS

Maximum number of remote devices a local device can be connected to.

This parameter defines the number of remote devices a local device can have simultaneous connections to (i.e. channels). This value should not exceed AVCTP\_MAX\_CHANNELS.

## 3.17.2.3 #define AVRCP\_MAX\_CMD\_BUFFERS 1

Maximum number of command buffers.

This parameter defines the number of buffers reserved for sending commands to a remote device over its control channel. Each channel uses its own buffers so the total number of buffers is AVRCP\_MAX\_CHANNELS ∗ AVR← CP\_MAX\_CMD\_BUFFERS. The minimum value is 1. The maximum value is 255. If not define one buffer for each channel is reserved.

## 3.17.2.4 #define AVRCP\_MAX\_CMD\_PARAM\_LEN 512

Maximum length of command parameters.

This parameter defines the maximum length of all command parameters. If not defined the default value of 512 is used.

## 3.17.2.5 #define AVRCP\_MAX\_DEVICE\_NAME\_LEN 20

Maximum length of device name.

This parameter defines the size of the buffer used to store device's name while searching for nearby targets with bt\_avrcp\_find\_targets. If the name of the device is longer than AVRCP\_MAX\_DEVICE\_NAME\_LEN it is truncated to AVRCP\_MAX\_DEVICE\_NAME\_LEN. If not defined the default value of 20 is used.

# 3.17.2.6 #define AVRCP\_MAX\_SEARCH\_RESULTS 7

Maximum number of devices to find.

This parameter defines the maximum number of devices bt\_avrcp\_find\_targets can find. If not defined the default value of 7 is used.

3.18 ATT 183

# 3.18 ATT

The attribute protocol allows a device referred to as the server to expose a set of attributes and their associated values to a peer device referred to as the client.

#### **Modules**

ATT Server

This module describes functions and data structures used to implement an ATT server (peripheral).

ATT Client

This module describes functions and data structures used to implement an ATT client (central).

## **Error Codes**

#define ATT\_ERROR\_SUCCESS 0x00

The operation completed with no errors.

#define ATT ERROR INVALID HANDLE 0x01

The attribute handle given was not valid on this server.

• #define ATT\_ERROR\_READ\_NOT\_PERMITTED 0x02

The attribute cannot be read.

#define ATT\_ERROR\_WRITE\_NOT\_PERMITTED 0x03

The attribute cannot be written.

#define ATT\_ERROR\_INVALID\_PDU 0x04

The attribute PDU was invalid.

#define ATT\_ERROR\_INSUFFICIENT\_AUTHENTICATION 0x05

The attribute requires authentication before it can be read or written.

#define ATT\_ERROR\_REQUEST\_NOT\_SUPPORTED 0x06

Attribute server does not support the request received from the client.

#define ATT\_ERROR\_INVALID\_OFFSET 0x07

Offset specified was past the end of the attribute.

• #define ATT ERROR INSUFFICIENT AUTHORIZATION 0x08

The attribute requires authorization before it can be read or written.

#define ATT\_ERROR\_PREPARE\_QUEUE\_FULL 0x09

Too many prepare writes have been queued.

• #define ATT\_ERROR\_ATTRIBUTE\_NOT\_FOUND 0x0A

No attribute found within the given attribute handle range.

#define ATT\_ERROR\_ATTRIBUTE\_NOT\_LONG 0x0B

The attribute cannot be read or written using the Read Blob Request.

#define ATT\_ERROR\_INSUFFICIENT\_ENCRYPTION\_KEY\_SIZE 0x0C

The Encryption Key Size used for encrypting this link is insufficient.

• #define ATT\_ERROR\_INVALID\_ATTRIBUTE\_VALUE\_LENGTH 0x0D

The attribute value length is invalid for the operation.

#define ATT\_ERROR\_UNLIKELY\_ERROR 0x0E

The attribute request that was requested has encountered an error.

• #define ATT\_ERROR\_INSUFFICIENT\_ENCRYPTION 0x0F

The attribute requires encryption before it can be read or written.

#define ATT\_ERROR\_UNSUPPORTED\_GROUP\_TYPE 0x10

The attribute type is not a supported.

#define ATT\_ERROR\_INSUFFICIENT\_RESOURCES 0x11

Insufficient Resources to complete the request.

# 3.18.1 Detailed Description

The attribute protocol allows a device referred to as the server to expose a set of attributes and their associated values to a peer device referred to as the client.

These attributes exposed by the server can be discovered, read, and written by a client, and can be indicated and notified by the server.

# 3.18.2 Macro Definition Documentation

# 3.18.2.1 #define ATT\_ERROR\_UNLIKELY\_ERROR 0x0E

The attribute request that was requested has encountered an error.

that was unlikely, and therefore could not be completed as requested.

# $3.18.2.2 \quad \hbox{\#define ATT\_ERROR\_UNSUPPORTED\_GROUP\_TYPE 0x10}$

The attribute type is not a supported.

grouping attribute as defined by a higher layer specification.

3.19 ATT Server 185

# 3.19 ATT Server

This module describes functions and data structures used to implement an ATT server (peripheral).

#### **Modules**

Configuration

This module describes parameters used to configure ATT server layer.

# **Data Structures**

struct bt\_att\_evt\_session\_connected\_t

Parameter to ATT\_SERVER\_EVT\_SESSION\_CONNECTED event.

struct bt\_att\_evt\_session\_disconnected\_t

Parameter to ATT\_SERVER\_EVT\_SESSION\_DISCONNECTED event.

struct bt\_att\_evt\_attr\_value\_changed\_t

Parameter to ATT\_SERVER\_EVT\_ATTR\_VALUE\_CHANGED\_BY\_SERVER and ATT\_SERVER\_EVT\_ATTR\_V↔ ALUE\_CHANGED\_BY\_CLIENT events.

struct bt\_att\_evt\_attr\_value\_read\_t

Parameter to ATT\_SERVER\_EVT\_ATTR\_VALUE\_READ event.

· struct bt\_att\_evt\_attr\_notification\_sent\_t

Parameter to ATT\_SERVER\_EVT\_ATTR\_VALUE\_NOTIFIED event.

struct bt\_att\_evt\_attr\_indication\_sent\_t

Parameter to ATT\_SERVER\_EVT\_ATTR\_VALUE\_INDICATED event.

struct bt\_att\_evt\_tran\_timeout\_t

Parameter to ATT\_SERVER\_EVT\_TRAN\_TIMEOUT event.

struct bt\_att\_evt\_authorization\_requested\_t

Parameter to ATT\_SERVER\_EVT\_AUTHORIZATION\_REQUESTED event.

struct bt\_att\_mgr\_t

ATT manager.

# **Typedefs**

typedef void(\* bt\_att\_mgr\_callback\_fp) (bt\_int evt, void \*evt\_param, void \*param)
 ATT application callback.

# **Functions**

bt\_bool bt\_att\_init (void)

Initialize the ATT layer.

bt\_att\_mgr\_t \* bt\_att\_get\_mgr (void)

Return a pointer to an instance of the ATT manager.

bt\_bool bt\_att\_update\_conn\_parameters (bt\_att\_session\_t \*session, bt\_uint min\_interval, bt\_uint max\_
interval, bt\_uint slave\_latency, bt\_uint supervision\_timeout)

Request connection parameters update.

bt\_bdaddr\_t \* bt\_att\_get\_remote\_address (const bt\_att\_session\_t \*session)

Get the address of the remote device this device is connected to.

• bt\_bool bt\_att\_server\_start (bt\_byte \*att\_var\_db, bt\_uint att\_var\_db\_len, const bt\_byte \*att\_const\_db, bt\_uint att\_const\_db\_len, bt\_bool listen\_on\_dynamic\_channel)

Start ATT server.

void bt\_att\_server\_stop (void)

Stop ATT server.

• void bt att server register callback (bt att server callback fp callback, void \*param)

Register a ATT application callback.

bt\_byte bt\_att\_server\_write\_attribute\_value (bt\_uint handle, const bt\_byte \*value, bt\_uint value\_len, bt\_uint offset)

Write attribute's value.

bt\_bool bt\_att\_server\_disconnect (bt\_att\_session\_t \*session)

Disconnect from a remote device.

void bt\_att\_server\_indicate\_to\_all (bt\_att\_attribute\_t \*attr)

Indicate attribute's value to all connected clients.

void bt\_att\_server\_indicate (bt\_att\_session\_t \*session, bt\_att\_attribute\_t \*attr)

Indicate attribute's value.

void bt\_att\_server\_notify (bt\_att\_attribute\_t \*attr)

Notify attribute's value to all connected clients.

 void bt\_att\_server\_authorize\_access (bt\_att\_session\_t \*session, bt\_att\_attribute\_t \*attr, bt\_byte opcode, bt bool authorize)

Authorize access to an attribute.

# **Events**

#define ATT\_SERVER\_EVT\_SESSION\_CONNECTED ATT\_MGR\_EVT\_SESSION\_CONNECTED
 A client connected to the server.

- #define ATT\_SERVER\_EVT\_SESSION\_DISCONNECTED ATT\_MGR\_EVT\_SESSION\_DISCONNECTED A client disconnected from the server.

Attribute's value has been changed locally on the server.

A client has changed the attribute's value.

A value notification has bee sent to the client.

- #define ATT\_SERVER\_EVT\_ATTR\_VALUE\_NOTIFIED ATT\_MGR\_EVT\_ATTR\_VALUE\_NOTIFIED
- #define ATT\_SERVER\_EVT\_ATTR\_VALUE\_INDICATED ATT\_MGR\_EVT\_ATTR\_VALUE\_INDICATED
- A value indication has been sent to the client.

   #define ATT\_SERVER\_EVT\_TRAN\_TIMEOUT ATT\_MGR\_EVT\_TRAN\_TIMEOUT

Operation timed out.

Authorization is required in order to access the attribute's value.

#define ATT\_SERVER\_EVT\_ATTR\_VALUE\_READ\_BY\_CLIENT ATT\_MGR\_EVT\_ATTR\_VALUE\_READ
 —BY\_CLIENT

A client has read the attribute's value.

#define ATT\_SERVER\_EVT\_STOPPED 50

Server stooped.

# 3.19.1 Detailed Description

This module describes functions and data structures used to implement an ATT server (peripheral).

3.19 ATT Server 187

# 3.19.2 Typedef Documentation

3.19.2.1 typedef void(\* bt\_att\_mgr\_callback\_fp) (bt\_int evt, void \*evt\_param, void \*param)

# ATT application callback.

In order to be notified of various events a consumer of the ATT layer has to register a callback function. The stack will call this function whenever a new event has been generated passing the code of the event as the first parameter.

#### Parameters 4 8 1

evt	ATT event
evt_param	Event parameter. This can be one of the following values:
	ATT_SERVER_EVT_SESSION_CONNECTED A client connected to the server
	ATT_SERVER_EVT_SESSION_DISCONNECTED A client disconnected from the server
	ATT_SERVER_EVT_ATTR_VALUE_CHANGED_BY_SERVER Attribute's value has been changed locally on the server
	ATT_SERVER_EVT_ATTR_VALUE_CHANGED_BY_CLIENT A client has changed the attribute's value
	ATT_SERVER_EVT_ATTR_VALUE_NOTIFIED A value notification has bee sent to the client
	ATT_SERVER_EVT_ATTR_VALUE_INDICATED A value indication has been sent to the client
	ATT_SERVER_EVT_TRAN_TIMEOUT Operation timed out
	ATT_SERVER_EVT_AUTHORIZATION_REQUESTED Authorization is required in order to access the attribute's value
	ATT_SERVER_EVT_STOPPED Server stooped

callback\_param | A pointer to an arbitrary data set by a call to bt\_att\_server\_register\_callback.

#### 3.19.3 Function Documentation

```
3.19.3.1 bt_att_mgr_t* bt_att_get_mgr ( void )
```

Return a pointer to an instance of the ATT manager.

This function returns a pointer to an instance of the ATT manager. There is only one instance of the manager allocated by the stack.

3.19.3.2 bt\_bdaddr\_t\* bt\_att\_get\_remote\_address ( const bt\_att\_session\_t \* session )

Get the address of the remote device this device is connected to.

# **Parameters**

session ATT session.

## Returns

• A pointer to bt\_bdaddr structure that contains the address of the remote device.

3.19.3.3 bt\_bool bt\_att\_init ( void )

Initialize the ATT layer.

This function initializes the ATT layer of the stack. It must be called prior to any other ATT function can be called.

3.19.3.4 void bt\_att\_server\_authorize\_access ( bt\_att\_session\_t \* session, bt\_att\_attribute\_t \* attr, bt\_byte opcode, bt\_bool authorize )

Authorize access to an attribute.

If an attribute requires authorization before its value can be read or written, ATT generates a ATT\_SERVER\_E 

VT\_AUTHORIZATION\_REQUESTED event. In response to this event the application should obtain authorization 
(how this is done is implementation specific) and call bt\_att\_server\_authorize\_access.

3.19 ATT Server 189

#### **Parameters**

session	ATT session.
attr	Attribute.
opcode	Attribute Opcode. The opcode that requires authorization is passed in the ATT_SERV ER_EVT_AUTHORIZATION_REQUESTED event. The application should use the passed opcode when calling bt_att_server_authorize_access. The opcode can be one the following values:
	ATT_OPCODE_READ_REQUEST
	ATT_OPCODE_READ_BLOB_REQUEST
	ATT_OPCODE_READ_BY_TYPE_REQUEST
	ATT_OPCODE_READ_BY_GROUP_TYPE_REQUEST
	ATT_OPCODE_READ_MULTIPLE_REQUEST
	ATT_OPCODE_WRITE_REQUEST
	ATT_OPCODE_PREPARE_WRITE_REQUEST
authorize	Specifies whether access to the attribute has been authorized or not.

3.19.3.5 bt\_bool bt\_att\_server\_disconnect ( bt\_att\_session\_t \* session )

Disconnect from a remote device.

This function closes a connection to a remote device.

# **Parameters**

session	ATT session.
---------	--------------

# Returns

- TRUE if disconnection has been started.
- $\bullet \ \ \mathtt{FALSE}$  otherwise. No events will be generated.

3.19.3.6 void bt\_att\_server\_indicate ( bt\_att\_session\_t \* session, bt\_att\_attribute\_t \* attr )

Indicate attribute's value.

This function sends the attribute's value specified with the attr parameter to a client sepecified with the session parameter.

# **Parameters**

session	ATT session.
attr	Attribute.

3.19.3.7 void bt\_att\_server\_indicate\_to\_all ( bt\_att\_attribute\_t \* attr )

Indicate attribute's value to all connected clients.

This function sends the attribute's value specified with the attr parameter to all connected clients.

#### **Parameters**

attr	Attribute.

3.19.3.8 void bt\_att\_server\_notify ( bt\_att\_attribute\_t \* attr )

Notify attribute's value to all connected clients.

This function sends the attribute's value specified with the attr parameter to all connected clients.

#### **Parameters**

attr	Attribute.

3.19.3.9 void bt\_att\_server\_register\_callback ( bt\_att\_server\_callback\_fp callback, void \* param )

Register a ATT application callback.

In order to be notified of various events a consumer of the ATT layer has to register a callback function. The stack will call this function whenever a new event has been generated passing the code of the event as the first parameter. The event can be one of the following values:

- ATT\_SERVER\_EVT\_SESSION\_CONNECTED A client connected to the server
- ATT\_SERVER\_EVT\_SESSION\_DISCONNECTED A client disconnected from the server
- ATT\_SERVER\_EVT\_ATTR\_VALUE\_CHANGED\_BY\_SERVER Attribute's value has been changed locally on the server
- ATT\_SERVER\_EVT\_ATTR\_VALUE\_CHANGED\_BY\_CLIENT A client has changed the attribute's value
- ATT\_SERVER\_EVT\_ATTR\_VALUE\_NOTIFIED A value notification has bee sent to the client
- ATT\_SERVER\_EVT\_ATTR\_VALUE\_INDICATED A value indication has been sent to the client
- ATT\_SERVER\_EVT\_TRAN\_TIMEOUT Transaction timed out
- ATT\_SERVER\_EVT\_AUTHORIZATION\_REQUESTED Authorization is required in order to access the attribute's value
- ATT SERVER EVT STOPPED Server stooped

# Parameters

callback	The callback function that will be called when ATT generates an event.
callback_param	A pointer to arbitrary data to be passed to the callback callback.

3.19.3.10 bt\_bool bt\_att\_server\_start ( bt\_byte \* att\_var\_db, bt\_uint att\_var\_db\_len, const bt\_byte \* att\_const\_db, bt\_uint att\_const\_db\_len, bt\_bool listen\_on\_dynamic\_channel )

Start ATT server.

This function starts the ATT server.

**Parameters** 

Copyright © 2015 SEARAN LLC. All right reserved.

3.19 ATT Server 191

att_var_db	A pointer to ATT database that holds writable attributes.
att_var_db_len	The length of the writable ATT database.
att_const_db	A pinter to ATT database that holds read-only attributes.
att_const_db_←	The length of the read-only ATT database.
len	
listen_on_←	Specifies whether ATT server should accept connections on BR/EDR links.
dynamic_←	
channel	

# Returns

- TRUE if the function succeeds.
- FALSE otherwise.

3.19.3.11 void bt\_att\_server\_stop (void)

Stop ATT server.

This function stops the ATT server.

#### Returns

- TRUE if the function succeeds.
- FALSE otherwise.

3.19.3.12 bt\_byte bt\_att\_server\_write\_attribute\_value ( bt\_uint handle, const bt\_byte \* value, bt\_uint value\_len, bt\_uint offset )

Write attribute's value.

This function writes attribute's value.

# Parameters

handle	Attribute handle.
value	Attribute value.
value_len	Attribute value length.
offset	Offset from which to write the value.

# Returns

Error code. This can be one of the following values:

- ATT\_ERROR\_SUCCESS
- ATT\_ERROR\_INVALID\_HANDLE
- ATT\_ERROR\_READ\_NOT\_PERMITTED
- ATT\_ERROR\_WRITE\_NOT\_PERMITTED
- ATT\_ERROR\_INVALID\_PDU
- ATT\_ERROR\_INSUFFICIENT\_AUTHENTICATION
- ATT\_ERROR\_REQUEST\_NOT\_SUPPORTED
- ATT\_ERROR\_INVALID\_OFFSET
- ATT\_ERROR\_INSUFFICIENT\_AUTHORIZATION
- ATT\_ERROR\_PREPARE\_QUEUE\_FULL
- ATT\_ERROR\_ATTRIBUTE\_NOT\_FOUND

- ATT\_ERROR\_ATTRIBUTE\_NOT\_LONG
- ATT\_ERROR\_INSUFFICIENT\_ENCRYPTION\_KEY\_SIZE
- ATT\_ERROR\_INVALID\_ATTRIBUTE\_VALUE\_LENGTH
- ATT\_ERROR\_UNLIKELY\_ERROR
- ATT\_ERROR\_INSUFFICIENT\_ENCRYPTION
- ATT\_ERROR\_UNSUPPORTED\_GROUP\_TYPE
- ATT ERROR INSUFFICIENT RESOURCES

3.19.3.13 bt\_bool bt\_att\_update\_conn\_parameters ( bt\_att\_session\_t \* session, bt\_uint min\_interval, bt\_uint max\_interval, bt\_uint slave\_latency, bt\_uint supervision\_timeout )

Request connection parameters update.

This function sends a request to the client (central) to update connection parameters.

#### **Parameters**

session	ATT session.
min_interval	Minimum connection interval expressed in 1.25ms units.
max_interval	Maximum connection interval expressed in 1.25ms units.
slave_latency	Slave latency expressed as number of connection events.
supervision_←	Link supervision timeout expressed in 10ms units.
timeout	

- TRUE if request has been sent to the central.
- FALSE otherwise.

3.20 Configuration 193

# 3.20 Configuration

This module describes parameters used to configure ATT server layer.

#### **Macros**

• #define ATT MAX CLIENTS

Maximum number of clients.

• #define ATT MAX FOUND ATTRIBUTES

Maximum number of attributes that can be returned in one multi-attribute response.

• #define ATT\_MAX\_QUEUED\_WRITE\_BUFFER\_SIZE 0

Size of the buffer for storing queued writes.

# 3.20.1 Detailed Description

This module describes parameters used to configure ATT server layer.

dotstack is customized using a configuration file. The configuration file tailors the dotstack to the application being built. It has to have the structure shown below.

```
#include "cdbt/bt/bt_std.h"
// HCI and L2CAP must always be present
// SDP is required only if stack is running in dual mode. This is the default mode.
// To run the stack in single mode (i.e. only BLE is supported) a BT_BLE_SINGLE_MODE symbol
// must be defined:
// #define BT_BLE_SINGLE_MODE
// HCI configuration parameters
#define HCI_MAX_CMD_BUFFERS
#define HCI_MAX_DATA_BUFFERS
#define HCI_MAX_HCI_CONNECTIONS
#define HCI_RX_BUFFER_LEN
#define HCI_TX_BUFFER_LEN
#define HCI_L2CAP_BUFFER_LEN
#define HCI_MAX_CMD_PARAM_LEN
// L2CAP configuration parameters
#define L2CAP_MAX_CMD_BUFFERS
#define L2CAP_MAX_FRAME_BUFFERS
#define L2CAP_MAX_PSMS
#define L2CAP_MAX_CHANNELS
// SDP configuration parameters
#define SDP_MAX_SEARCH_RESULT_LEN
#define SDP_MAX_SEARCH_RESULT_LEN ... #define SDP_MAX_ATTRIBUTE_RESULT_LEN ...
// Depending on protocols and profiles used below goes configuration parameters
  for each used module. E.g., to use and configure ATT,
// the following values must be defined:
                                    // tells dotstack to compile in ATT support
#define BT INCLUDE ATT
#define ATT MAX CLIENTS
#define ATT_MAX_FOUND_ATTRIBUTES
#define ATT_MAX_QUEUED_WRITE_BUFFER_SIZE
#include "cdbt/bt/bt_oem_config.h"
```

# 3.20.2 Macro Definition Documentation

## 3.20.2.1 #define ATT\_MAX\_CLIENTS

Maximum number of clients.

This parameter defines the maximum number of clients that can be simultaneously connected to the server. If ATT is used only over LE this number must be set to 1. Any other numbers will be a waste of RAM because an LE slave (server) can have only one connection to a master (client) at any given moment.

# 3.20.2.2 #define ATT\_MAX\_FOUND\_ATTRIBUTES

Maximum number of attributes that can be returned in one multi-attribute response.

This parameter defines the maximum number of attributes that can be returned in one multi-attribute response. The minimum value is 1. The maximum value is 255. This number must be set to as large value as possible (given there is enough RAM) to minimize the number of request needed by the client to find all requested attributes.

3.20.2.3 #define ATT\_MAX\_QUEUED\_WRITE\_BUFFER\_SIZE 0

Size of the buffer for storing queued writes.

This parameter defines the Size of the buffer for storing queued writes. The minimum value is 0. The maximum value is 65535. If this is set to 0 queued writes will not be supported.

3.21 ATT Client 195

# 3.21 ATT Client

This module describes functions and data structures used to implement an ATT client (central).

## **Modules**

Configuration

This module describes parameters used to configure ATT client layer.

# **Data Structures**

```
· struct bt_att_find_info_response_t
```

Structure to store response to a "find info" request.

struct bt\_att\_find\_by\_type\_value\_response\_t

Structure to store response to a "find by type value" request.

struct bt\_att\_read\_by\_type\_response\_t

Structure to store response to a "read by type" request.

struct bt\_att\_read\_by\_group\_type\_response\_t

Structure to store response to a "read by group type" request.

struct bt\_att\_client\_evt\_mtu\_response\_t

Parameter to ATT\_CLIENT\_EVT\_EXCHANGE\_MTU\_RESPONSE event.

struct bt\_att\_client\_evt\_info\_response\_t

Parameter to ATT\_CLIENT\_EVT\_FIND\_INFO\_RESPONSE event.

struct bt\_att\_client\_evt\_find\_by\_type\_value\_response\_t

Parameter to ATT\_CLIENT\_EVT\_FIND\_BY\_TYPE\_VALUE\_RESPONSE event.

struct bt\_att\_client\_evt\_read\_by\_type\_response\_t

Parameter to ATT\_CLIENT\_EVT\_READ\_BY\_TYPE\_RESPONSE event.

· struct bt\_att\_client\_evt\_read\_response\_t

Parameter to ATT\_CLIENT\_EVT\_READ\_RESPONSE event.

· struct bt\_att\_client\_evt\_read\_blob\_response\_t

Parameter to ATT\_CLIENT\_EVT\_READ\_BLOB\_RESPONSE event.

struct bt\_att\_client\_evt\_read\_multiple\_response\_t

Parameter to ATT\_CLIENT\_EVT\_READ\_MULTIPLE\_RESPONSE event.

struct bt\_att\_client\_evt\_read\_by\_group\_type\_response\_t

Parameter to ATT\_CLIENT\_EVT\_READ\_BY\_GROUP\_TYPE\_RESPONSE event.

struct bt\_att\_client\_evt\_write\_response\_t

Parameter to ATT CLIENT EVT WRITE RESPONSE event.

struct bt\_att\_client\_evt\_prepare\_write\_response\_t

Parameter to ATT\_CLIENT\_EVT\_PREPARE\_WRITE\_RESPONSE event.

struct bt\_att\_client\_evt\_execute\_write\_response\_t

Parameter to ATT\_CLIENT\_EVT\_EXECUTE\_WRITE\_RESPONSE event.

struct bt\_att\_client\_evt\_value\_notification\_t

Parameter to ATT\_CLIENT\_EVT\_VALUE\_NOTIFICATION event.

struct bt\_att\_client\_evt\_value\_indication\_t

Parameter to ATT\_CLIENT\_EVT\_VALUE\_INDICATION event.

struct bt\_att\_client\_evt\_conn\_param\_update\_t

Parameter to ATT\_CLIENT\_EVT\_CONN\_PARAM\_UPDATE\_REQUEST event.

struct bt\_att\_client\_evt\_conn\_param\_update\_completed\_t

Parameter to ATT CLIENT EVT CONN PARAM UPDATE COMPLETED event.

union bt\_att\_client\_evt\_t

Parameter to ATT client application callback.

# **Functions**

bt bool bt att client init (void)

Initialize the ATT client.

bt\_att\_client\_mgr\_t \* bt\_att\_get\_client\_mgr (void)

Return a pointer to an instance of the ATT client manager.

 bt\_att\_client\_session\_t \* bt\_att\_client\_allocate\_session (bt\_att\_client\_session\_callback\_fp callback, void \*callback\_param)

Allocate ATT client session.

void bt\_att\_client\_free\_session (bt\_att\_client\_session\_t \*session)

Destroy ATT client session.

• bt\_bool bt\_att\_client\_connect (bt\_att\_client\_session\_t \*session, bt\_bdaddr\_t \*addr)

Connect to a remote device (peripheral).

bt\_bool bt\_att\_client\_disconnect (bt\_att\_client\_session\_t \*session)

Disconnect from a remote device.

• bt\_bool bt\_att\_client\_exchange\_mtu (bt\_att\_client\_session\_t \*session, bt\_uint mtu)

Exchange MTU.

• bt\_bool bt\_att\_client\_find\_info (bt\_att\_client\_session\_t \*session, bt\_uint start\_handle, bt\_uint end\_handle, bt\_att\_find\_info\_response\_t \*result, bt\_uint max\_results)

Find Information.

bt\_bool bt\_att\_client\_find\_by\_type\_value (bt\_att\_client\_session\_t \*session, bt\_uint start\_handle, bt\_uint end\_handle, bt\_uint type, const bt\_byte \*value, bt\_uint len, bt\_att\_find\_by\_type\_value\_response\_t \*result, bt\_uint max\_results)

Find By Type Value.

bt\_bool bt\_att\_client\_read\_by\_type (bt\_att\_client\_session\_t \*session, bt\_uint start\_handle, bt\_uint end\_
 handle, bt\_uint type, bt\_att\_read\_by\_type\_response\_t \*result, bt\_uint max\_results)

Read By Type (16-bit).

• bt\_bool bt\_att\_client\_read\_by\_type\_80 (bt\_att\_client\_session\_t \*session, bt\_uint start\_handle, bt\_uint end 
\_ handle, bt\_uuid\_t \*type, bt\_att\_read\_by\_type\_response\_t \*result, bt\_uint max\_results)

Read By Type (128-bit).

bt\_bool bt\_att\_client\_read (bt\_att\_client\_session\_t \*session, bt\_uint handle)

Read Attribute value.

• bt bool bt att client read blob (bt att client session t \*session, bt uint handle, bt uint offset)

Read Blob.

- bt\_bool bt\_att\_client\_read\_multiple (bt\_att\_client\_session\_t \*session, const bt\_uint \*handles, bt\_uint count)
   Read Multiple.
- bt\_bool bt\_att\_client\_read\_by\_group\_type (bt\_att\_client\_session\_t \*session, bt\_uint start\_handle, bt\_uint end\_handle, bt\_uint group\_type, bt\_att\_read\_by\_group\_type\_response\_t \*result, bt\_uint max\_results)

Read By Group Type (16-bit).

bt\_bool bt\_att\_client\_read\_by\_group\_type\_80 (bt\_att\_client\_session\_t \*session, bt\_uint start\_handle, bt
 — uint end\_handle, bt\_uuid\_t \*group\_type, bt\_att\_read\_by\_group\_type\_response\_t \*result, bt\_uint max\_
 results)

Read By Group Type (128-bit).

bt\_bool bt\_att\_client\_write (bt\_att\_client\_session\_t \*session, bt\_uint handle, const bt\_byte \*value, bt\_uint len)

Write Attribute Value.

• bt\_bool bt\_att\_client\_write\_cmd (bt\_att\_client\_session\_t \*session, bt\_uint handle, const bt\_byte \*value, bt 
uint len)

Write Command.

• bt\_bool bt\_att\_client\_signed\_write (bt\_att\_client\_session\_t \*session, bt\_uint handle, const bt\_byte \*value, bt\_uint len, const bt\_byte \*signature)

Signed Write Command.

3.21 ATT Client 197

 bt\_bool bt\_att\_client\_prepare\_write (bt\_att\_client\_session\_t \*session, bt\_uint handle, bt\_uint value\_offset, const bt\_byte \*value, bt\_uint len)

Prepare Write.

bt bool bt att client execute write (bt att client session t \*session, bt byte flags)

Execute Write.

bt\_bool bt\_att\_client\_accept\_conn\_param\_update (bt\_att\_client\_session\_t \*session, bt\_uint min\_interval, bt\_uint max\_interval, bt\_uint slave\_latency, bt\_uint supervision\_timeout, bt\_uint min\_ce\_length, bt\_uint max\_ce\_length)

Accept connection parameters update.

void bt att client reject conn param update (bt att client session t \*session)

Reject connection parameters update.

• bt\_bdaddr\_t \* bt\_att\_client\_get\_remote\_address (const bt att client session t \*session)

Get the address of the remote device this device is connected to.

#### **Events**

#define ATT\_CLIENT\_SESSION\_EVT\_SESSION\_CONNECTED 1

This event is generated when client connected to the server.

#define ATT CLIENT SESSION EVT SESSION DISCONNECTED 2

This event is generated when client disconnected from the server.

#define ATT\_CLIENT\_SESSION\_EVT\_CONNECTION\_FAILED 3

This event is generated when client failed to connect to the server.

• #define ATT\_CLIENT\_SESSION\_EVT\_TRAN\_TIMEOUT 7

This event is generated if operation (find, read, write) timed out.

• #define ATT\_CLIENT\_EVT\_EXCHANGE\_MTU\_RESPONSE 10

This event is generated when client received a response (either positive or negative) to a "exchange MTU" request.

• #define ATT CLIENT EVT FIND INFO RESPONSE 11

This event is generated when client received a response (either positive or negative) to a "find info" request.

#define ATT\_CLIENT\_EVT\_FIND\_BY\_TYPE\_VALUE\_RESPONSE 12

This event is generated when client received a response (either positive or negative) to a "find by type value" request.

#define ATT CLIENT EVT READ BY TYPE RESPONSE 13

This event is generated when client received a response (either positive or negative) to a "read by type" request.

#define ATT\_CLIENT\_EVT\_READ\_RESPONSE 14

This event is generated when client received a response (either positive or negative) to a "read" request.

#define ATT CLIENT EVT READ BLOB RESPONSE 15

This event is generated when client received a response (either positive or negative) to a "read blob" request.

#define ATT\_CLIENT\_EVT\_READ\_MULTIPLE\_RESPONSE 16

This event is generated when client received a response (either positive or negative) to a "read multiple" request.

#define ATT\_CLIENT\_EVT\_READ\_BY\_GROUP\_TYPE\_RESPONSE 17

This event is generated when client received a response (either positive or negative) to a "read by group type" request.

• #define ATT CLIENT EVT WRITE RESPONSE 18

This event is generated when client received a response (either positive or negative) to a "write" request.

#define ATT\_CLIENT\_EVT\_PREPARE\_WRITE\_RESPONSE 19

This event is generated when client received a response (either positive or negative) to a "preapare write" request.

#define ATT\_CLIENT\_EVT\_EXECUTE\_WRITE\_RESPONSE 20

This event is generated when client received a response (either positive or negative) to a "execute write" request.

#define ATT\_CLIENT\_EVT\_VALUE\_NOTIFICATION 21

This event is generated when client received attribute value notification.

#define ATT CLIENT EVT VALUE INDICATION 22

This event is generated when client received attribute value indication.

#define ATT\_CLIENT\_EVT\_CONN\_PARAM\_UPDATE\_REQUEST 50

This event is generated when client received a "connection parameter update" request.

#define ATT\_CLIENT\_EVT\_CONN\_PARAM\_UPDATE\_COMPLETED 51

This event is generated after the new connection parameters have been set.

# 3.21.1 Detailed Description

This module describes functions and data structures used to implement an ATT client (central).

# 3.21.2 Function Documentation

3.21.2.1 bt\_bool bt\_att\_client\_accept\_conn\_param\_update ( bt\_att\_client\_session\_t \* session, bt\_uint min\_interval, bt\_uint max\_interval, bt\_uint slave\_latency, bt\_uint supervision\_timeout, bt\_uint min\_ce\_length, bt\_uint max\_ce\_length )

Accept connection parameters update.

When a server sends "connection parameters update" request a ATT\_CLIENT\_EVT\_CONN\_PARAM\_UPDATE  $\leftarrow$  \_REQUEST event is generated. Client has to either accept the request or deny it. bt\_att\_client\_accept  $\leftarrow$  \_conn\_param\_update is used to accept the request.

#### **Parameters**

session	ATT session.
min_interval	Minimum connection interval expressed in 1.25ms units.
max_interval	Maximum connection interval expressed in 1.25ms units.
slave_latency	Slave latency expressed as number of connection events.
supervision_←	Link supervision timeout expressed in 10ms units.
timeout	
min_ce_length	Information parameter about the minimum length of connection needed for this LE connection
	expressed in 0.625ms units.
max_ce_length	Information parameter about the maximum length of connection needed for this LE connec-
	tion expressed in 0.625ms units.

## Returns

- TRUE if a request has been sent.
- ${\tt FALSE}$  otherwise. No events will be generated.

3.21.2.2 bt\_att\_client\_session\_t\* bt\_att\_client\_allocate\_session ( bt\_att\_client\_session\_callback\_fp callback, void \* callback\_param )

Allocate ATT client session.

This function allocates a new ATT client session.

# **Parameters**

callback	The callback function that will be called when the ATT client generates an event.
callback_param	A pointer to arbitrary data to be passed to the callback callback.

- A pointer to the new ATT client session structure if the function succeeds.
- NULL otherwise.

3.21 ATT Client 199

3.21.2.3 bt\_bool bt\_att\_client\_connect ( bt\_att\_client\_session\_t \* session, bt\_bdaddr\_t \* addr )

Connect to a remote device (peripheral).

This function establishes a connection to a remote device specified by the addr. If connection cannot be initiated for some reason, for example, there is not enough resources, it returns FALSE and no events are generated. Otherwise the result of an attempt to connect to the remote device is reported via the ATT client callback. The events generated will either be ATT\_CLIENT\_SESSION\_EVT\_SESSION\_CONNECTED or ATT\_CLIENT\_SESSION\_EVT\_CONNECTED.

#### **Parameters**

session	ATT session.
addr	The address of a remote device.

#### Returns

- · TRUE if connection establishment has been started.
- FALSE otherwise.

3.21.2.4 bt\_bool bt\_att\_client\_disconnect ( bt\_att\_client\_session\_t \* session )

Disconnect from a remote device.

This function closes a connection to a remote device. After the connection has been terminated the ATT client callback is called with ATT\_CLIENT\_SESSION\_EVT\_SESSION\_CONNECTED event.

#### **Parameters**

session	ATT session.
---------	--------------

# Returns

- TRUE if disconnection has been started.
- FALSE otherwise. No events will be generated.

3.21.2.5 bt\_bool bt\_att\_client\_exchange\_mtu ( bt\_att\_client\_session\_t \* session, bt\_uint mtu )

## Exchange MTU.

This function informs the server about the client's MTU. In response to the "exchange MTU" request the server sends its MTU to the client.

# Parameters

session	ATT session.
mtu	Client's MTU.

# Returns

- TRUE if a request has been sent.
- FALSE otherwise. No events will be generated.

3.21.2.6 bt\_bool bt\_att\_client\_execute\_write ( bt\_att\_client\_session\_t \* session, bt\_byte flags )

## Execute Write.

This function sends a "execute write request to the client. ATT\_CLIENT\_EVT\_EXECUTE\_WRITE\_RESPONSE event is generated when the response from the client has been received.

#### **Parameters**

session	ATT session.
flags	This can be one of the following values:
	<ul> <li>0x00 - Cancel all prepared writes.</li> <li>0x01 - Immediately write all pending prepared values.</li> </ul>

# Returns

- TRUE if a request has been sent.
- FALSE otherwise. No events will be generated.
- 3.21.2.7 bt\_bool bt\_att\_client\_find\_by\_type\_value ( bt\_att\_client\_session\_t \* session, bt\_uint start\_handle, bt\_uint end\_handle, bt\_uint type, const bt\_byte \* value, bt\_uint len, bt\_att\_find\_by\_type\_value\_response\_t \* result, bt\_uint max\_results )

Find By Type Value.

This function sends a "find by type value request to the client. ATT\_CLIENT\_EVT\_FIND\_BY\_TYPE\_VALUE\_RE ← SPONSE event is generated when the response from the client has been received.

#### **Parameters**

session	ATT session.
start_handle	First requested handle number.
end_handle	Last requested handle number.
type	Attribute type.
value	Attribute value to find.
len	Length of the value.
result	Pointer to a buffer where response will be stored.
max_result	The maximum number of responses that result can store.

# Returns

- TRUE if a request has been sent.
- FALSE otherwise. No events will be generated.
- 3.21.2.8 bt\_bool bt\_att\_client\_find\_info ( bt\_att\_client\_session\_t \* session, bt\_uint start\_handle, bt\_uint end\_handle, bt\_att\_find\_info\_response\_t \* result, bt\_uint max\_results )

# Find Information.

This function sends a "find information request to the client. ATT\_CLIENT\_EVT\_FIND\_INFO\_RESPONSE event is generated when the response from the client has been received.

# **Parameters**

session	ATT session.
start_handle	First requested handle number.
end_handle	Last requested handle number.

3.21 ATT Client 201

result	Pointer to a buffer where response will be stored.
max_result	The maximum number of responses that result can store.

# Returns

- TRUE if a request has been sent.
- FALSE otherwise. No events will be generated.

3.21.2.9 void bt\_att\_client\_free\_session ( bt\_att\_client\_session\_t \* session )

Destroy ATT client session.

This function frees memory used by the session. The session has to exist and be in the "idle" state for this function to succeed. I.e. the session has to be disconnected before this function can be called.

## **Parameters**

session ATT session	
---------------------	--

## Returns

- · TRUE if the function succeeds.
- FALSE otherwise.

3.21.2.10 bt\_bdaddr\_t\* bt\_att\_client\_get\_remote\_address ( const bt\_att\_client\_session\_t \* session )

Get the address of the remote device this device is connected to.

#### **Parameters**

session	ATT client session.
---------	---------------------

# Returns

• A pointer to bt\_bdaddr structure that contains the address of the remote device.

3.21.2.11 bt\_bool bt\_att\_client\_init ( void )

Initialize the ATT client.

This function initializes the ATT client of the stack. It must be called prior to any other ATT client function can be called.

3.21.2.12 bt\_bool bt\_att\_client\_prepare\_write ( bt\_att\_client\_session\_t \* session, bt\_uint handle, bt\_uint value\_offset, const bt\_byte \* value, bt\_uint len )

# Prepare Write.

This function sends a "prepare write request to the client. ATT\_CLIENT\_EVT\_PREPARE\_WRITE\_RESPONSE event is generated when the response from the client has been received.

# **Parameters**

session	ATT session.
handle	The handle of the attribute to be written.
value_offset	The offset of the first octet to be written.
value	The value to be written to the attribute.
len	The length of the value.

# Returns

- TRUE if a request has been sent.
- FALSE otherwise. No events will be generated.

3.21.2.13 bt\_bool bt\_att\_client\_read ( bt\_att\_client\_session\_t \* session, bt\_uint handle )

#### Read Attribute value.

This function sends a "read request to the client. ATT\_CLIENT\_EVT\_READ\_RESPONSE event is generated when the response from the client has been received.

#### **Parameters**

session	ATT session.
handle	The handle of the attribute to be read.

#### Returns

- TRUE if a request has been sent.
- FALSE otherwise. No events will be generated.

3.21.2.14 bt\_bool bt\_att\_client\_read\_blob ( bt\_att\_client\_session\_t \* session, bt\_uint handle, bt\_uint offset )

## Read Blob.

This function sends a "read blob request to the client. ATT\_CLIENT\_EVT\_READ\_BLOB\_RESPONSE event is generated when the response from the client has been received.

## **Parameters**

session	ATT session.
handle	The handle of the attribute to be read.
offset	The offset of the first octet to be read.

## Returns

- TRUE if a request has been sent.
- FALSE otherwise. No events will be generated.

3.21.2.15 bt\_bool bt\_att\_client\_read\_by\_group\_type ( bt\_att\_client\_session\_t \* session, bt\_uint start\_handle, bt\_uint end\_handle, bt\_uint group\_type, bt\_att\_read\_by\_group\_type\_response\_t \* result, bt\_uint max\_results )

Read By Group Type (16-bit).

This function sends a "read by group type request to the client. ATT\_CLIENT\_EVT\_READ\_BY\_GROUP\_TYPE\_← RESPONSE event is generated when the response from the client has been received.

3.21 ATT Client 203

#### **Parameters**

session	ATT session.
start_handle	First requested handle number.
end_handle	Last requested handle number.
group_type	16-bit Attribute Group Type.
result	Pointer to a buffer where response will be stored.
max_result	The maximum number of responses that result can store.

#### Returns

- TRUE if a request has been sent.
- FALSE otherwise. No events will be generated.

3.21.2.16 bt\_bool bt\_att\_client\_read\_by\_group\_type\_80 ( bt\_att\_client\_session\_t \* session, bt\_uint start\_handle, bt\_uint end\_handle, bt\_uuid\_t \* group\_type, bt\_att\_read\_by\_group\_type\_response\_t \* result, bt\_uint max\_results )

Read By Group Type (128-bit).

This function sends a "read by group type request to the client. ATT\_CLIENT\_EVT\_READ\_BY\_GROUP\_TYPE\_← RESPONSE event is generated when the response from the client has been received.

# **Parameters**

session	ATT session.
start_handle	First requested handle number.
end_handle	Last requested handle number.
group_type	128-bit Attribute Group Type.
result	Pointer to a buffer where response will be stored.
max_result	The maximum number of responses that result can store.

# Returns

- TRUE if a request has been sent.
- ${\tt FALSE}$  otherwise. No events will be generated.

3.21.2.17 bt\_bool bt\_att\_client\_read\_by\_type ( bt\_att\_client\_session\_t \* session, bt\_uint start\_handle, bt\_uint end\_handle, bt\_uint type, bt\_att\_read\_by\_type\_response\_t \* result, bt\_uint max\_results )

Read By Type (16-bit).

This function sends a "read by type request to the client. ATT\_CLIENT\_EVT\_READ\_BY\_TYPE\_RESPONSE event is generated when the response from the client has been received.

#### **Parameters**

session	ATT session.
start_handle	First requested handle number.
end_handle	Last requested handle number.
type	16-bit attribute type.
result	Pointer to a buffer where response will be stored.
max_result	The maximum number of responses that result can store.

- TRUE if a request has been sent.
- FALSE otherwise. No events will be generated.

3.21.2.18 bt\_bool bt\_att\_client\_read\_by\_type\_80 ( bt\_att\_client\_session\_t \* session, bt\_uint start\_handle, bt\_uint end\_handle, bt\_uuid\_t \* type, bt\_att\_read\_by\_type\_response\_t \* result, bt\_uint max\_results )

Read By Type (128-bit).

This function sends a "read by type request to the client. ATT\_CLIENT\_EVT\_READ\_BY\_TYPE\_RESPONSE event is generated when the response from the client has been received.

#### **Parameters**

session	ATT session.
start_handle	First requested handle number.
end_handle	Last requested handle number.
type	128-bit attribute type.
result	Pointer to a buffer where response will be stored.
max_result	The maximum number of responses that result can store.

#### Returns

- TRUE if a request has been sent.
- FALSE otherwise. No events will be generated.

3.21.2.19 bt\_bool bt\_att\_client\_read\_multiple ( bt\_att\_client\_session\_t \* session, const bt\_uint \* handles, bt\_uint count )

# Read Multiple.

This function sends a "read blob request to the client. ATT\_CLIENT\_EVT\_READ\_MULTIPLE\_RESPONSE event is generated when the response from the client has been received.

# **Parameters**

session	ATT session.
handles	The list of two or more handles.
count	The number of handles in handles.

# Returns

- TRUE if a request has been sent.
- FALSE otherwise. No events will be generated.

3.21.2.20 void bt\_att\_client\_reject\_conn\_param\_update ( bt\_att\_client\_session\_t \* session )

Reject connection parameters update.

When a server sends "connection parameters update" request a ATT\_CLIENT\_EVT\_CONN\_PARAM\_UPDATE — \_\_REQUEST event is generated. Client has to either accept the request or deny it. bt\_att\_client\_reject — \_\_conn\_param\_update is used to deny the request.

## **Parameters**

session	ATT session.

- TRUE if a request has been sent.
- FALSE otherwise. No events will be generated.

3.21 ATT Client 205

3.21.2.21 bt\_bool bt\_att\_client\_signed\_write ( bt\_att\_client\_session\_t \* session, bt\_uint handle, const bt\_byte \* value, bt\_uint len, const bt\_byte \* signature )

# Signed Write Command.

This function sends a "signed write command request to the client. No event is generated.

#### Parameters 4 8 1

session	ATT session.
handle	The handle of the attribute to be written.
value	The value to be written to the attribute.
len	The length of the value.
signature	Authentication signature (12 bytes).

#### Returns

- TRUE if a request has been sent.
- FALSE otherwise. No events will be generated.

3.21.2.22 bt\_bool bt\_att\_client\_write ( bt\_att\_client\_session\_t \* session, bt\_uint handle, const bt\_byte \* value, bt\_uint len )

# Write Attribute Value.

This function sends a "write request to the client. ATT\_CLIENT\_EVT\_WRITE\_RESPONSE event is generated when the response from the client has been received.

#### **Parameters**

session	ATT session.
handle	The handle of the attribute to be written.
value	The value to be written to the attribute.
len	The length of the value.

# Returns

- TRUE if a request has been sent.
- FALSE otherwise. No events will be generated.

3.21.2.23 bt\_bool bt\_att\_client\_write\_cmd ( bt\_att\_client\_session\_t \* session, bt\_uint handle, const bt\_byte \* value, bt\_uint len )

# Write Command.

This function sends a "write command request to the client. No event is generated.

# **Parameters**

session	ATT session.
handle	The handle of the attribute to be written.
value	The value to be written to the attribute.
len	The length of the value.

- TRUE if a request has been sent.
- FALSE otherwise. No events will be generated.

3.21.2.24 bt\_att\_client\_mgr\_t\* bt\_att\_get\_client\_mgr ( void )

Return a pointer to an instance of the ATT client manager.

This function returns a pointer to an instance of the ATT client manager. There is only one instance of the manager allocated by the stack.

# Returns

• A pointer to the ATT client manager.

3.22 Configuration 207

# 3.22 Configuration

This module describes parameters used to configure ATT client layer.

## **Macros**

• #define ATT\_CLIENT\_MAX\_SESSIONS

Maximum number of ATT sessions.

# 3.22.1 Detailed Description

This module describes parameters used to configure ATT client layer.

dotstack is customized using a configuration file. The configuration file tailors the dotstack to the application being built. It has to have the structure shown below.

```
#include "cdbt/bt/bt_std.h"
// HCI and L2CAP must always be present
// SDP is required only if stack is running in dual mode. This is the default mode.
// To run the stack in single mode (i.e. only BLE is supported) a BT_BLE_SINGLE_MODE symbol
// must be defined:
// #define BT_BLE_SINGLE_MODE
// HCI configuration parameters
#define HCI_MAX_CMD_BUFFERS
#define HCI_MAX_DATA_BUFFERS
#define HCI MAX HCI CONNECTIONS
#define HCI_RX_BUFFER_LEN
#define HCI_TX_BUFFER_LEN
#define HCI_L2CAP_BUFFER_LEN
#define HCI_MAX_CMD_PARAM_LEN
// L2CAP configuration parameters
#define L2CAP_MAX_CMD_BUFFERS
#define L2CAP_MAX_FRAME_BUFFERS
                                           . . .
#define L2CAP_MAX_PSMS
#define L2CAP_MAX_CHANNELS
\ensuremath{//} SDP configuration parameters
#define SDP_MAX_SEARCH_RESULT_LEN #define SDP_MAX_ATTRIBUTE_RESULT_LEN
// Depending on protocols and profiles used below goes configuration parameters
// for each used module. E.g., to use and configure ATT,
// the following values must be defined:
#define BT INCLUDE ATT
                                         // tells dotstack to compile in ATT support
#define ATT_CLIENT_MAX_SESSIONS
#include "cdbt/bt/bt_oem_config.h"
```

# 3.22.2 Macro Definition Documentation

# 3.22.2.1 #define ATT\_CLIENT\_MAX\_SESSIONS

Maximum number of ATT sessions.

This parameter defines the maximum number of connections a client can have with servers. The minimum value is 1. The maximum value is 7.

# 3.23 **GATT**

The GATT profile is designed to be used by an application or another profile, so that a client can communicate with a server.

#### **Modules**

GATT Server

This module describes functions and data structures used to implement a GATT server (peripheral).

GATT Client

This module describes functions and data structures used to implement a GATT client (central).

# 3.23.1 Detailed Description

The GATT profile is designed to be used by an application or another profile, so that a client can communicate with a server.

The server contains a number of attributes, and the GATT Profile defines how to use the Attribute Protocol to discover, read, write and obtain indications of these attributes, as well as configuring broadcast of attributes.

The Generic Attribute Profile (GATT) defines a service framework using the Attribute Protocol. This framework defines procedures and formats of services and their characteristics. The procedures defined include discovering, reading, writing, notifying and indicating characteristics, as well as configuring the broadcast of characteristics.

3.24 GATT Server 209

# 3.24 GATT Server

This module describes functions and data structures used to implement a GATT server (peripheral).

#### **Data Structures**

• struct bt\_gatt\_evt\_header\_t

Common to all event parameters header.

· struct bt\_gatt\_evt\_client\_config\_changed\_t

Parameter to GATT\_SERVER\_EVT\_CLIENT\_CONFIG\_CHANGED event.

struct bt\_gatt\_evt\_server\_config\_changed\_t

Parameter to GATT\_SERVER\_EVT\_SERVER\_CONFIG\_CHANGED event.

· struct bt\_gatt\_evt\_ext\_properties\_changed\_t

Parameter to GATT SERVER EVT EXTENDED PROPERTIES CHANGED event.

• struct bt\_gatt\_evt\_value\_changed\_t

Parameter to GATT\_SERVER\_EVT\_VALUE\_CHANGED event.

• struct bt\_gatt\_evt\_value\_read\_t

Parameter to GATT\_SERVER\_EVT\_VALUE\_READ event.

#### **Macros**

#define bt\_gatt\_server\_start(att\_var\_db, att\_var\_db\_len, att\_const\_db, att\_const\_db\_len) bt\_gatt\_server\_
 start\_ex(att\_var\_db, att\_var\_db\_len, att\_const\_db, att\_const\_db\_len, BT\_FALSE)

Start GATT server.

#define bt\_gatt\_server\_authorize\_access(session, attr, opcode, authorize) bt\_att\_server\_authorize\_
 access(session, attr, opcode, authorize)

Authorize access to an attribute.

## **Functions**

• bt\_bool bt\_gatt\_server\_start\_ex (bt\_byte \*att\_var\_db, bt\_uint att\_var\_db\_len, const bt\_byte \*att\_const\_db, bt\_uint att\_const\_db len, bt\_bool listen on dynamic channel)

Start GATT server (extended version)

void bt\_gatt\_server\_register\_callback (bt\_gatt\_server\_callback\_fp callback, void \*param)

Register a GATT application callback.

bt\_bool bt\_gatt\_server\_register\_listener (bt\_gatt\_listener\_t \*listener)

Register a GATT event listener.

void bt\_gatt\_server\_unregister\_listener (bt\_gatt\_listener\_t \*listener)

Unregister a GATT event listener.

• bt\_byte bt\_gatt\_server\_write\_char\_value (bt\_uuid16 service\_type, bt\_uuid16 service\_id, bt\_uuid16 characteristic\_type, const bt\_byte \*value, bt\_uint value\_len, bt\_uint offset)

Write characteristic's value (16-bit).

bt\_byte bt\_gatt\_server\_write\_char\_value\_80 (bt\_uuid\_t \*service\_type, bt\_uuid\_t \*service\_id, bt\_uuid\_
 t \*characteristic\_type, const bt\_byte \*value, bt\_uint value\_len, bt\_uint offset)

Write characteristic's value (128-bit).

bt\_byte bt\_gatt\_server\_notify\_char\_value (bt\_uuid16 service\_type, bt\_uuid16 service\_id, bt\_uuid16 characteristic\_type)

Notify characteristic's value (16-bit).

bt\_byte bt\_gatt\_server\_notify\_char\_value\_80 (bt\_uuid\_t \*service\_type, bt\_uuid\_t \*service\_id, bt\_uuid\_
 t \*characteristic\_type)

Notify characteristic's value (128-bit).

bt\_byte bt\_gatt\_server\_indicate\_char\_value (bt\_uuid16 service\_type, bt\_uuid16 service\_id, bt\_uuid16 characteristic\_type)

Indicate characteristic's value (16-bit).

• bt\_byte bt\_gatt\_server\_indicate\_char\_value\_80 (bt\_uuid\_t \*service\_type, bt\_uuid\_t \*service\_id, bt\_uuid\_t \*characteristic\_type)

Indicate characteristic's value (128-bit).

#### **Events**

- #define GATT\_SERVER\_EVT\_SESSION\_CONNECTED ATT\_SERVER\_EVT\_SESSION\_CONNECTED A client connected to the server.
- #define GATT\_SERVER\_EVT\_SESSION\_DISCONNECTED ATT\_SERVER\_EVT\_SESSION\_DISCONN←
   ECTED

A client disconnected from the server.

- #define GATT\_SERVER\_EVT\_ATTR\_VALUE\_NOTIFIED ATT\_SERVER\_EVT\_ATTR\_VALUE\_NOTIFIED A value notification has bee sent to the client.

A value indication has been sent to the client.

#define GATT\_SERVER\_EVT\_AUTHORIZATION\_REQUESTED ATT\_SERVER\_EVT\_AUTHORIZATIO
 N REQUESTED

Authorization is required in order to access the attribute's value.

#define GATT\_SERVER\_EVT\_CLIENT\_CONFIG\_CHANGED 100

Characteristic's client configuration changed.

#define GATT\_SERVER\_EVT\_EXTENDED\_PROPERTIES\_CHANGED 101

Characteristic's extended properties changed.

#define GATT\_SERVER\_EVT\_SERVER\_CONFIG\_CHANGED 102

Characteristic's server configuration changed.

• #define GATT\_SERVER\_EVT\_VALUE\_CHANGED 103

Characteristic's value changed.

#define GATT\_SERVER\_EVT\_VALUE\_READ 104

Characteristic's value has been read.

# 3.24.1 Detailed Description

This module describes functions and data structures used to implement a GATT server (peripheral).

# 3.24.2 Macro Definition Documentation

3.24.2.1 #define bt\_gatt\_server\_authorize\_access( session, attr, opcode, authorize ) bt\_att\_server\_authorize\_access(session, attr, opcode, authorize)

Authorize access to an attribute.

If an attribute requires authorization before its value can be read or written, GATT generates a GATT\_SERVER\_ EVT\_AUTHORIZATION\_REQUESTED event. In response to this event the application should obtain authorization (how this is done is implementation specific) and call bt gatt server authorize access.

3.24 GATT Server 211

#### **Parameters**

session	ATT session.
attr	Attribute.
opcode	Attribute Opcode. The opcode that requires authorization is passed in the ATT_SERV -
	ER_EVT_AUTHORIZATION_REQUESTED event. The application should use the passed
	opcode when calling bt_att_server_authorize_access. The opcode can be one the following
	values:
	ATT_OPCODE_READ_REQUEST
	ATT_OPCODE_READ_BLOB_REQUEST
	ATT_OPCODE_READ_BY_TYPE_REQUEST
	ATT_OPCODE_READ_BY_GROUP_TYPE_REQUEST
	ATT_OPCODE_READ_MULTIPLE_REQUEST
	ATT_OPCODE_WRITE_REQUEST
	ATT_OPCODE_PREPARE_WRITE_REQUEST
authorize	Specifies whether access to the attribute has been authorized or not.

3.24.2.2 #define bt\_gatt\_server\_start( att\_var\_db, att\_var\_db\_len, att\_const\_db, att\_const\_db\_len ) bt\_gatt\_server\_start\_ex(att\_var\_db, att\_var\_db\_len, att\_const\_db, att\_const\_db\_len, BT\_FALSE)

# Start GATT server.

This function starts the GATT server. This function start the server only on LE links.

# **Parameters**

att_var_db	A pointer to ATT database that holds writable attributes.
att_var_db_len	The length of the writable ATT database.
att_const_db	A pinter to ATT database that holds read-only attributes.
att_const_db_ <i>←</i>	The length of the read-only ATT database.
len	
listen_on_←	Specifies whether ATT server should accept connections on BR/EDR links.
dynamic_ <i>←</i>	
channel	

# Returns

- TRUE if the function succeeds.
- FALSE otherwise.

# 3.24.3 Function Documentation

3.24.3.1 bt\_byte bt\_gatt\_server\_indicate\_char\_value ( bt\_uuid16 service\_type, bt\_uuid16 service\_id, bt\_uuid16 characteristic\_type )

Indicate characteristic's value (16-bit).

This function sends current characteristic's value to the client. The value is sent regardless of characteristic's client configuration. GATT\_SERVER\_EVT\_ATTR\_VALUE\_INDICATED event is generated after receiving confirmation from the client.

#### **Parameters**

service_type	16-bit service type UUID.
service_id	16-bit service type UUID.
characteristic_←	16-bit characteristic type UUID.
type	

#### Returns

Error code. This can be one of the following values:

- ATT\_ERROR\_SUCCESS
- ATT\_ERROR\_INVALID\_HANDLE
- · ATT ERROR READ NOT PERMITTED
- ATT\_ERROR\_WRITE\_NOT\_PERMITTED
- ATT\_ERROR\_INVALID\_PDU
- ATT\_ERROR\_INSUFFICIENT\_AUTHENTICATION
- ATT\_ERROR\_REQUEST\_NOT\_SUPPORTED
- ATT\_ERROR\_INVALID\_OFFSET
- · ATT ERROR INSUFFICIENT AUTHORIZATION
- ATT\_ERROR\_PREPARE\_QUEUE\_FULL
- ATT\_ERROR\_ATTRIBUTE\_NOT\_FOUND
- ATT\_ERROR\_ATTRIBUTE\_NOT\_LONG
- · ATT ERROR INSUFFICIENT ENCRYPTION KEY SIZE
- ATT\_ERROR\_INVALID\_ATTRIBUTE\_VALUE\_LENGTH
- ATT\_ERROR\_UNLIKELY\_ERROR
- ATT ERROR INSUFFICIENT ENCRYPTION
- ATT\_ERROR\_UNSUPPORTED\_GROUP\_TYPE
- ATT\_ERROR\_INSUFFICIENT\_RESOURCES

3.24.3.2 bt\_byte bt\_gatt\_server\_indicate\_char\_value\_80 ( bt\_uuid\_t \* service\_type, bt\_uuid\_t \* service\_id, bt\_uuid\_t \* characteristic\_type )

Indicate characteristic's value (128-bit).

This function sends current characteristic's value to the client. The value is sent regardless of characteristic's client configuration. GATT\_SERVER\_EVT\_ATTR\_VALUE\_INDICATED event is generated after receiving confirmation from the client. In order to use standard 16-bit UUIDs with this function they have to be converted to 128-bit UUID by combining with a base UUID which is 00000000 (this part is replaced with 16-bit UUID) - 00001000 - 80000080 - 5F9B34FB.

# **Parameters**

service_type	128-bit service type UUID.
service_id	128-bit service type UUID.
characteristic_←	128-bit characteristic type UUID.
type	

#### Returns

Error code. This can be one of the following values:

- · ATT ERROR SUCCESS
- ATT ERROR INVALID HANDLE

3.24 GATT Server 213

- ATT\_ERROR\_READ\_NOT\_PERMITTED
- · ATT ERROR WRITE NOT PERMITTED
- · ATT ERROR INVALID PDU
- ATT\_ERROR\_INSUFFICIENT\_AUTHENTICATION
- ATT\_ERROR\_REQUEST\_NOT\_SUPPORTED
- ATT\_ERROR\_INVALID\_OFFSET
- · ATT ERROR INSUFFICIENT AUTHORIZATION
- ATT\_ERROR\_PREPARE\_QUEUE\_FULL
- ATT\_ERROR\_ATTRIBUTE\_NOT\_FOUND
- · ATT ERROR ATTRIBUTE NOT LONG
- · ATT ERROR INSUFFICIENT ENCRYPTION KEY SIZE
- ATT\_ERROR\_INVALID\_ATTRIBUTE\_VALUE\_LENGTH
- ATT\_ERROR\_UNLIKELY\_ERROR
- ATT ERROR INSUFFICIENT ENCRYPTION
- ATT\_ERROR\_UNSUPPORTED\_GROUP\_TYPE
- ATT\_ERROR\_INSUFFICIENT\_RESOURCES

# 3.24.3.3 bt\_byte bt\_gatt\_server\_notify\_char\_value ( bt\_uuid16 service\_type, bt\_uuid16 service\_id, bt\_uuid16 characteristic\_type )

Notify characteristic's value (16-bit).

This function sends current characteristic's value to the client. The value is sent regardless of characteristic's client configuration. GATT\_SERVER\_EVT\_ATTR\_VALUE\_NOTIFIED is generated right after the value has been sent.

#### **Parameters**

service_type	16-bit service type UUID.
service_id	16-bit service type UUID.
characteristic_←	16-bit characteristic type UUID.
type	

#### Returns

Error code. This can be one of the following values:

- · ATT ERROR SUCCESS
- ATT\_ERROR\_INVALID\_HANDLE
- ATT\_ERROR\_READ\_NOT\_PERMITTED
- ATT\_ERROR\_WRITE\_NOT\_PERMITTED
- ATT\_ERROR\_INVALID\_PDU
- ATT\_ERROR\_INSUFFICIENT\_AUTHENTICATION
- ATT\_ERROR\_REQUEST\_NOT\_SUPPORTED
- · ATT ERROR INVALID OFFSET
- ATT\_ERROR\_INSUFFICIENT\_AUTHORIZATION
- ATT\_ERROR\_PREPARE\_QUEUE\_FULL
- ATT\_ERROR\_ATTRIBUTE\_NOT\_FOUND
- ATT\_ERROR\_ATTRIBUTE\_NOT\_LONG
- ATT\_ERROR\_INSUFFICIENT\_ENCRYPTION\_KEY\_SIZE
- ATT\_ERROR\_INVALID\_ATTRIBUTE\_VALUE\_LENGTH
- ATT\_ERROR\_UNLIKELY\_ERROR
- ATT\_ERROR\_INSUFFICIENT\_ENCRYPTION
- ATT\_ERROR\_UNSUPPORTED\_GROUP\_TYPE
- ATT\_ERROR\_INSUFFICIENT\_RESOURCES

3.24.3.4 bt\_byte bt\_gatt\_server\_notify\_char\_value\_80 ( bt\_uuid\_t \* service\_type, bt\_uuid\_t \* service\_id, bt\_uuid\_t \* characteristic\_type )

Notify characteristic's value (128-bit).

This function sends current characteristic's value to the client. The value is sent regardless of characteristic's client configuration. GATT\_SERVER\_EVT\_ATTR\_VALUE\_NOTIFIED is generated right after the value has been sent. In order to use standard 16-bit UUIDs with this function they have to be converted to 128-bit UUID by combining with a base UUID which is 00000000 (this part is replaced with 16-bit UUID) - 00001000 - 80000080 - 5F9B34FB.

#### **Parameters**

service_type	128-bit service type UUID.
service_id	128-bit service type UUID.
characteristic_←	128-bit characteristic type UUID.
type	

#### Returns

Error code. This can be one of the following values:

- · ATT ERROR SUCCESS
- ATT\_ERROR\_INVALID\_HANDLE
- · ATT ERROR READ NOT PERMITTED
- · ATT ERROR WRITE NOT PERMITTED
- ATT ERROR INVALID PDU
- · ATT ERROR INSUFFICIENT AUTHENTICATION
- ATT ERROR REQUEST NOT SUPPORTED
- ATT\_ERROR\_INVALID\_OFFSET
- ATT\_ERROR\_INSUFFICIENT\_AUTHORIZATION
- ATT\_ERROR\_PREPARE\_QUEUE\_FULL
- ATT\_ERROR\_ATTRIBUTE\_NOT\_FOUND
- · ATT ERROR ATTRIBUTE NOT LONG
- ATT\_ERROR\_INSUFFICIENT\_ENCRYPTION\_KEY\_SIZE
- · ATT ERROR INVALID ATTRIBUTE VALUE LENGTH
- ATT\_ERROR\_UNLIKELY\_ERROR
- ATT ERROR INSUFFICIENT ENCRYPTION
- · ATT ERROR UNSUPPORTED GROUP TYPE
- ATT\_ERROR\_INSUFFICIENT\_RESOURCES

3.24.3.5 void bt\_gatt\_server\_register\_callback ( bt\_gatt\_server\_callback\_fp callback, void \* param )

Register a GATT application callback.

In order to be notified of various events a consumer of the GATT layer has to register a callback function. The stack will call this function whenever a new event has been generated passing the code of the event as the first parameter. The event can be one of the following values:

- GATT\_SERVER\_EVT\_SESSION\_CONNECTED A client connected to the server.
- GATT SERVER EVT SESSION DISCONNECTED A client disconnected from the server.
- GATT\_SERVER\_EVT\_ATTR\_VALUE\_NOTIFIED A value notification has bee sent to the client.
- GATT\_SERVER\_EVT\_ATTR\_VALUE\_INDICATED A value indication has been sent to the client.

3.24 GATT Server 215

GATT\_SERVER\_EVT\_AUTHORIZATION\_REQUESTED Authorization is required in order to access the attribute's value.

- GATT\_SERVER\_EVT\_CLIENT\_CONFIG\_CHANGED Characteristic's client configuration changed
- GATT\_SERVER\_EVT\_EXTENDED\_PROPERTIES\_CHANGED Characteristic's extended properties changed
- GATT\_SERVER\_EVT\_SERVER\_CONFIG\_CHANGED Characteristic's server configuration changed
- GATT SERVER EVT VALUE CHANGED Characteristic's value changed

#### **Parameters**

callback	The callback function that will be called when GATT generates an event.
callback_param	A pointer to arbitrary data to be passed to the callback callback.

3.24.3.6 bt\_bool bt\_gatt\_server\_register\_listener ( bt\_gatt\_listener\_t \* listener )

Register a GATT event listener.

In order to be notified of various events a consumer of the GATT layer may register an event listener. The listener is a structure that contains a pointer to a callback and callback parameter - a pointer to arbitrary data to be passed to the callback . The stack will call the callback whenever a new event has been generated passing the code of the event as the first parameter. The event can be one of the following values:

- GATT\_SERVER\_EVT\_SESSION\_CONNECTED A client connected to the server.
- GATT\_SERVER\_EVT\_SESSION\_DISCONNECTED A client disconnected from the server.
- GATT\_SERVER\_EVT\_ATTR\_VALUE\_NOTIFIED A value notification has bee sent to the client.
- GATT\_SERVER\_EVT\_ATTR\_VALUE\_INDICATED A value indication has been sent to the client.
- GATT\_SERVER\_EVT\_AUTHORIZATION\_REQUESTED Authorization is required in order to access the attribute's value.
- GATT\_SERVER\_EVT\_CLIENT\_CONFIG\_CHANGED Characteristic's client configuration changed
- GATT\_SERVER\_EVT\_EXTENDED\_PROPERTIES\_CHANGED Characteristic's extended properties changed
- GATT\_SERVER\_EVT\_SERVER\_CONFIG\_CHANGED Characteristic's server configuration changed
- GATT\_SERVER\_EVT\_VALUE\_CHANGED Characteristic's value changed

# **Parameters**

listener	Listener.

3.24.3.7 bt\_bool bt\_gatt\_server\_start\_ex ( bt\_byte \* att\_var\_db, bt\_uint att\_var\_db\_len, const bt\_byte \* att\_const\_db, bt\_uint att\_const\_db\_len, bt\_bool listen\_on\_dynamic\_channel )

Start GATT server (extended version)

This function starts the GATT server. This function can be used to start the server on BR/EDR links.

#### **Parameters**

att_var_db	A pointer to ATT database that holds writable attributes.
att_var_db_len	The length of the writable ATT database.
att_const_db	A pinter to ATT database that holds read-only attributes.
att_const_db_←	The length of the read-only ATT database.
len	
listen_on_←	Specifies whether ATT server should accept connections on BR/EDR links.
dynamic_←	
channel	

#### Returns

- TRUE if the function succeeds.
- FALSE otherwise.

3.24.3.8 void bt\_gatt\_server\_unregister\_listener ( bt\_gatt\_listener\_t \* listener )

Unregister a GATT event listener.

Remove a listener previosuly registered with bt\_gatt\_server\_register\_listener.

#### **Parameters**

	listener	Listener.
--	----------	-----------

3.24.3.9 bt\_byte bt\_gatt\_server\_write\_char\_value ( bt\_uuid16 service\_type, bt\_uuid16 service\_id, bt\_uuid16 characteristic\_type, const bt\_byte \* value, bt\_uint value\_len, bt\_uint offset )

Write characteristic's value (16-bit).

This function updates characteristic's value. If the client configured characteristic to be notified or indicated this function sends new value to the client.

## Parameters

service_type	16-bit service type UUID.
service_id	16-bit service type UUID.
characteristic_←	16-bit characteristic type UUID.
type	

## Returns

Error code. This can be one of the following values:

- ATT\_ERROR\_SUCCESS
- ATT\_ERROR\_INVALID\_HANDLE
- ATT\_ERROR\_READ\_NOT\_PERMITTED
- ATT\_ERROR\_WRITE\_NOT\_PERMITTED
- ATT\_ERROR\_INVALID\_PDU
- ATT\_ERROR\_INSUFFICIENT\_AUTHENTICATION
- ATT\_ERROR\_REQUEST\_NOT\_SUPPORTED
- · ATT ERROR INVALID OFFSET
- ATT\_ERROR\_INSUFFICIENT\_AUTHORIZATION
- ATT\_ERROR\_PREPARE\_QUEUE\_FULL
- ATT\_ERROR\_ATTRIBUTE\_NOT\_FOUND

3.24 GATT Server 217

- ATT\_ERROR\_ATTRIBUTE\_NOT\_LONG
- ATT\_ERROR\_INSUFFICIENT\_ENCRYPTION\_KEY\_SIZE
- · ATT ERROR INVALID ATTRIBUTE VALUE LENGTH
- ATT\_ERROR\_UNLIKELY\_ERROR
- ATT\_ERROR\_INSUFFICIENT\_ENCRYPTION
- ATT\_ERROR\_UNSUPPORTED\_GROUP\_TYPE
- · ATT ERROR INSUFFICIENT RESOURCES

3.24.3.10 bt\_byte bt\_gatt\_server\_write\_char\_value\_80 ( bt\_uuid\_t \* service\_type, bt\_uuid\_t \* service\_id, bt\_uuid\_t \* characteristic\_type, const bt\_byte \* value, bt\_uint value\_len, bt\_uint offset )

Write characteristic's value (128-bit).

This function updates characteristic's value. If the client configured characteristic to be notified or indicated this function sends new value to the client. In order to use standard 16-bit UUIDs with this function they have to be converted to 128-bit UUID by combining with a base UUID which is 00000000 (this part is replaced with 16-bit UUID) - 00001000 - 80000080 - 5F9B34FB.

#### **Parameters**

service_type	128-bit service type UUID.
service_id	128-bit service type UUID.
characteristic_←	128-bit characteristic type UUID.
type	

#### Returns

Error code. This can be one of the following values:

- · ATT ERROR SUCCESS
- ATT\_ERROR\_INVALID\_HANDLE
- ATT\_ERROR\_READ\_NOT\_PERMITTED
- ATT\_ERROR\_WRITE\_NOT\_PERMITTED
- ATT\_ERROR\_INVALID\_PDU
- ATT\_ERROR\_INSUFFICIENT\_AUTHENTICATION
- ATT\_ERROR\_REQUEST\_NOT\_SUPPORTED
- · ATT ERROR INVALID OFFSET
- ATT\_ERROR\_INSUFFICIENT\_AUTHORIZATION
- ATT\_ERROR\_PREPARE\_QUEUE\_FULL
- ATT\_ERROR\_ATTRIBUTE\_NOT\_FOUND
- · ATT ERROR ATTRIBUTE NOT LONG
- ATT\_ERROR\_INSUFFICIENT\_ENCRYPTION\_KEY\_SIZE
- ATT\_ERROR\_INVALID\_ATTRIBUTE\_VALUE\_LENGTH
- ATT\_ERROR\_UNLIKELY\_ERROR
- ATT\_ERROR\_INSUFFICIENT\_ENCRYPTION
- ATT\_ERROR\_UNSUPPORTED\_GROUP\_TYPE
- ATT\_ERROR\_INSUFFICIENT\_RESOURCES

# 3.25 GATT Client

This module describes functions and data structures used to implement a GATT client (central).

#### **Modules**

Configuration

This module describes parameters used to configure GATT client.

# **Data Structures**

```
• struct bt_att_client_uuid_t
```

UUID.

struct bt\_gatt\_client\_service\_definition\_t

Service Definition.

struct bt\_gatt\_client\_inc\_service\_declaration\_t

Included Service Declaration.

struct bt\_gatt\_client\_char\_declaration\_t

Characteristic Declaration.

struct bt\_gatt\_client\_char\_value\_t

Characteristic Value.

• struct bt\_gatt\_client\_char\_descriptor\_t

Characteristic Descriptor Declaration.

struct bt\_gatt\_client\_evt\_exchange\_mtu\_completed\_t

Parameter to GATT\_CLIENT\_EVT\_EXCHANGE\_MTU\_COMPLETED event.

struct bt\_gatt\_client\_evt\_discover\_all\_services\_completed\_t

Parameter to GATT\_CLIENT\_EVT\_DISCOVER\_ALL\_SERVICES\_COMPLETED event.

struct bt\_gatt\_client\_evt\_discover\_service\_by\_uuid\_completed\_t

 ${\it Parameter to GATT\_CLIENT\_EVT\_DISCOVER\_SERVICE\_BY\_UUID\_COMPLETED \ event.}$ 

· struct bt gatt client evt find included completed t

Parameter to GATT\_CLIENT\_EVT\_FIND\_INCLUDED\_SERVICES\_COMPLETED event.

struct bt\_gatt\_client\_evt\_discover\_all\_chars\_completed\_t

Parameter to GATT CLIENT EVT DISCOVER ALL CHARS COMPLETED event.

struct bt\_gatt\_client\_evt\_discover\_char\_by\_uuid\_completed\_t

Parameter to GATT\_CLIENT\_EVT\_DISCOVER\_CHAR\_BY\_UUID\_COMPLETED event.

struct bt\_gatt\_client\_evt\_read\_char\_value\_completed\_t

Parameter to GATT\_CLIENT\_EVT\_READ\_CHAR\_VALUE\_COMPLETED event.

struct bt\_gatt\_client\_evt\_read\_by\_char\_uuid\_completed\_t

Parameter to GATT\_CLIENT\_EVT\_READ\_USING\_CHAR\_UUID\_COMPLETED event.

struct bt\_gatt\_client\_evt\_read\_multiple\_char\_values\_completed\_t

Parameter to GATT CLIENT EVT READ MULTIPLE CHAR VALUES COMPLETED event.

struct bt\_gatt\_client\_evt\_read\_char\_long\_value\_completed\_t

Parameter to GATT\_CLIENT\_EVT\_READ\_CHAR\_LONG\_VALUE\_COMPLETED event.

· struct bt gatt client evt write char value completed t

Parameter to GATT\_CLIENT\_EVT\_WRITE\_CHAR\_VALUE\_COMPLETED event.

struct bt\_gatt\_client\_evt\_write\_char\_long\_value\_completed\_t

Parameter to GATT\_CLIENT\_EVT\_WRITE\_CHAR\_LONG\_VALUE\_COMPLETED event.

struct bt\_gatt\_client\_evt\_value\_notification\_t

Parameter to GATT\_CLIENT\_EVT\_VALUE\_NOTIFICATION event.

• struct bt\_gatt\_client\_evt\_discover\_descriptors\_completed\_t

3.25 GATT Client 219

Parameter to GATT\_CLIENT\_EVT\_DISCOVER\_CHAR\_DESCRIPTORS\_COMPLETED event.

struct bt\_gatt\_client\_evt\_read\_char\_descriptor\_completed\_t

Parameter to GATT CLIENT EVT READ CHAR DESCRIPTOR COMPLETED event.

struct bt\_gatt\_client\_evt\_read\_char\_long\_descriptor\_completed\_t

Parameter to GATT CLIENT EVT READ CHAR LONG DESCRIPTOR COMPLETED event.

struct bt\_gatt\_client\_evt\_write\_char\_descriptor\_completed\_t

Parameter to GATT\_CLIENT\_EVT\_WRITE\_CHAR\_DESCRIPTOR\_COMPLETED event.

struct bt\_gatt\_client\_evt\_write\_char\_long\_descriptor\_completed\_t

Parameter to GATT\_CLIENT\_EVT\_WRITE\_CHAR\_LONG\_DESCRIPTOR\_COMPLETED event.

struct bt\_gatt\_client\_evt\_conn\_param\_update\_t

Parameter to GATT\_CLIENT\_EVT\_CONN\_PARAM\_UPDATE\_REQUEST event.

struct bt\_gatt\_client\_evt\_conn\_param\_update\_completed\_t

Parameter to GATT CLIENT EVT CONN PARAM UPDATE COMPLETED event.

union bt\_gatt\_client\_evt\_t

Parameter to GATT client application callback.

## **Macros**

#define bt\_gatt\_client\_accept\_conn\_param\_update(session, min\_interval, max\_interval, slave\_latency, supervision\_timeout, min\_ce\_length, max\_ce\_length) bt\_att\_client\_accept\_conn\_param\_update(session->att\_session, min\_interval, max\_interval, slave\_latency, supervision\_timeout, min\_ce\_length, max\_ce\_
 length)

Accept connection parameters update.

 #define bt\_gatt\_client\_reject\_conn\_param\_update(session) bt\_att\_client\_reject\_conn\_param\_update(session->att\_session)

Reject connection parameters update.

# **Functions**

bt\_bool bt\_gatt\_client\_init (void)

Initialize the GATT client.

bt\_gatt\_client\_mgr\_t \* bt\_gatt\_get\_client\_mgr (void)

Return a pointer to an instance of the GATT client manager.

 bt\_gatt\_client\_session\_t \* bt\_gatt\_client\_allocate\_session (bt\_gatt\_client\_session\_callback\_fp callback, void \*callback\_param)

Allocate GATT client session.

void bt\_gatt\_client\_free\_sessions (bt\_gatt\_client\_session\_t \*session)

Destroy GATT client session.

• bt\_bool bt\_gatt\_client\_connect (bt\_gatt\_client\_session\_t \*session, bt\_bdaddr\_t \*addr)

Connect to a remote device (peripheral).

• bt bool bt gatt client disconnect (bt gatt client session t \*session)

Disconnect from a remote device.

bt\_bool bt\_gatt\_exchange\_mtu (bt\_gatt\_client\_session\_t \*session, bt\_uint mtu)

Exchange MTU.

• bt\_bool bt\_gatt\_client\_discover\_all\_services\_ex (bt\_gatt\_client\_session\_t \*session, bt\_uint start\_handle, bt\_gatt\_client\_service\_definition\_t \*result, bt\_uint max\_results)

Discover All Services.

• bt\_bool bt\_gatt\_client\_discover\_by\_service\_uuid (bt\_gatt\_client\_session\_t \*session, bt\_uint service\_uuid, bt\_gatt\_client\_service\_definition\_t \*result, bt\_uint max\_results)

Discover Services by UUID (16-bit).

bt\_bool bt\_gatt\_client\_find\_included\_services\_ex (bt\_gatt\_client\_session\_t \*session, const bt\_gatt\_client
 \_service\_definition\_t \*service, bt\_uint start\_handle, bt\_gatt\_client\_inc\_service\_declaration\_t \*result, bt\_uint
 max results)

Find Included Services.

bt\_bool bt\_gatt\_client\_discover\_by\_service\_uuid\_80 (bt\_gatt\_client\_session\_t \*session, bt\_uuid\_

 t \*service uuid, bt gatt client service definition t \*result, bt uint max results)

Discover Services by UUID (128-bit).

Discover All Characteristics of a Service.

Discover Characteristics by UUID (16-bit).

bt\_bool bt\_gatt\_client\_discover\_char\_by\_uuid\_80 (bt\_gatt\_client\_session\_t \*session, const bt\_gatt\_client←
 \_service\_definition\_t \*service, bt\_uuid\_t \*char\_uuid, bt\_gatt\_client\_char\_declaration\_t \*result, bt\_uint
 max results)

Discover Characteristics by UUID (128-bit).

bt\_bool bt\_gatt\_client\_read\_char\_value (bt\_gatt\_client\_session\_t \*session, const bt\_gatt\_client\_char\_
 declaration t \*characteristic, bt byte \*buffer, bt uint len)

Read Characteristic Value.

bt\_bool bt\_gatt\_client\_read\_by\_char\_uuid (bt\_gatt\_client\_session\_t \*session, const bt\_gatt\_client\_service
 \_\_definition\_t \*service, bt\_uint char\_uuid, bt\_gatt\_client\_char\_value\_t \*result, bt\_uint max\_results)

Read Using Characteristic UUID (16-bit).

- bt\_bool bt\_gatt\_client\_read\_by\_char\_uuid\_80 (bt\_gatt\_client\_session\_t \*session, const bt\_gatt\_client\_⇔ service\_definition\_t \*service, bt\_uuid\_t \*char\_uuid, bt\_gatt\_client\_char\_value\_t \*result, bt\_uint max\_results)

  Read Using Characteristic UUID (128-bit).

Read Long Characteristic Value.

• bt\_bool bt\_gatt\_client\_read\_multiple\_char\_values (bt\_gatt\_client\_session\_t \*session, const bt\_uint \*handles, bt\_uint count, bt\_byte \*buffer, bt\_uint len)

Read Multiple Characteristic Values.

• bt\_bool bt\_gatt\_client\_write\_char\_value (bt\_gatt\_client\_session\_t \*session, const bt\_gatt\_client\_char\_ declaration\_t \*characteristic, const bt\_byte \*value, bt\_uint len)

Write Characteristic Value.

bt\_bool bt\_gatt\_client\_set\_char\_value (bt\_gatt\_client\_session\_t \*session, const bt\_gatt\_client\_char\_
 declaration\_t \*characteristic, const bt\_byte \*value, bt\_uint len)

Write Without Response.

Write Long Characteristic Value.

bt\_bool bt\_gatt\_client\_discover\_char\_descriptors (bt\_gatt\_client\_session\_t \*session, const bt\_gatt\_client←
 \_char\_declaration\_t \*characteristic, bt\_gatt\_client\_char\_descriptor\_t \*result, bt\_uint max\_results)

Discover All Characteristic Descriptors.

bt\_bool bt\_gatt\_client\_read\_char\_descriptor (bt\_gatt\_client\_session\_t \*session, const bt\_gatt\_client\_char
 \_\_descriptor\_t \*descriptor, bt\_byte \*buffer, bt\_uint len)

Read Characteristic Descriptor.

bt\_bool bt\_gatt\_client\_read\_char\_long\_descriptor (bt\_gatt\_client\_session\_t \*session, const bt\_gatt\_client client client client client session\_t \*session, const bt\_gatt\_client client cl

Read Long Characteristic Descriptor.

bt\_bool bt\_gatt\_client\_write\_char\_descriptor (bt\_gatt\_client\_session\_t \*session, const bt\_gatt\_client\_char
 \_\_descriptor\_t \*descriptor, const bt\_byte \*value, bt\_uint len)

3.25 GATT Client 221

Write Characteristic Descriptor.

bt\_bool bt\_gatt\_client\_write\_char\_long\_descriptor (bt\_gatt\_client\_session\_t \*session, const bt\_gatt\_client
 —char\_descriptor\_t \*descriptor, bt\_uint value\_offset, const bt\_byte \*value, bt\_uint len)

Write Long Characteristic Descriptor.

bt\_hci\_conn\_state\_t \* bt\_gatt\_client\_get\_hci\_connection (const bt\_gatt\_client\_session\_t \*session)
 Get HCI connection for a session.

## **Events**

#define GATT CLIENT EVT SESSION CONNECTED 1

This event is generated when the client connected to the server.

#define GATT\_CLIENT\_EVT\_SESSION\_DISCONNECTED 2

This event is generated when the client disconnected from the server.

#define GATT CLIENT EVT CONNECTION FAILED 3

This event is generated when the client failed to connect to the server.

#define GATT\_CLIENT\_EVT\_TRAN\_TIMEOUT 7

This event is generated if operation (discover, read, write) timed out.

#define GATT CLIENT EVT DISCOVER ALL SERVICES COMPLETED 10

This event is generated when the "discover all services" operation has completed.

#define GATT CLIENT EVT DISCOVER SERVICE BY UUID COMPLETED 11

This event is generated when the "discover service by UUID" operation has completed.

#define GATT CLIENT EVT DISCOVER ALL CHARS COMPLETED 12

This event is generated when the "discover all cgaracteristics" operation has completed.

#define GATT\_CLIENT\_EVT\_READ\_CHAR\_VALUE\_COMPLETED 13

This event is generated when the "read characteristic value" operation has completed.

#define GATT\_CLIENT\_EVT\_READ\_CHAR\_LONG\_VALUE\_COMPLETED 14

This event is generated when the "read long characteristic value" operation has completed.

#define GATT\_CLIENT\_EVT\_WRITE\_CHAR\_VALUE\_COMPLETED 15

This event is generated when the "write characteristic value" operation has completed.

• #define GATT\_CLIENT\_EVT\_WRITE\_CHAR\_LONG\_VALUE\_COMPLETED 16

This event is generated when the "write long characteristic value" operation has completed.

• #define GATT\_CLIENT\_EVT\_VALUE\_NOTIFICATION 17

This event is generated when the client received value notification or indication.

• #define GATT CLIENT EVT DISCOVER CHAR DESCRIPTORS COMPLETED 18

This event is generated when the "discover characteristic descriptiors" operation has completed.

• #define GATT\_CLIENT\_EVT\_WRITE\_CHAR\_DESCRIPTOR\_COMPLETED 19

This event is generated when the "write characteristic descriptor" operation has completed.

• #define GATT\_CLIENT\_EVT\_WRITE\_CHAR\_LONG\_DESCRIPTOR\_COMPLETED 20

This event is generated when the "write characteristic long descriptor" operation has completed.

#define GATT\_CLIENT\_EVT\_READ\_CHAR\_DESCRIPTOR\_COMPLETED 21

This event is generated when the "read characteristic descriptor" operation has completed.

• #define GATT\_CLIENT\_EVT\_READ\_CHAR\_LONG\_DESCRIPTOR\_COMPLETED 22

This event is generated when the "read characteristic long descriptor" operation has completed.

#define GATT\_CLIENT\_EVT\_EXCHANGE\_MTU\_COMPLETED 23

This event is generated when the "exchange MTU" operation has completed.

• #define GATT\_CLIENT\_EVT\_FIND\_INCLUDED\_SERVICES\_COMPLETED 24

This event is generated when the "find included services" operation has completed.

#define GATT\_CLIENT\_EVT\_DISCOVER\_CHAR\_BY\_UUID\_COMPLETED 25

This event is generated when the "discover characteristic by UUID" operation has completed.

• #define GATT\_CLIENT\_EVT\_READ\_USING\_CHAR\_UUID\_COMPLETED 26

This event is generated when the "read characteristic value using characteristic UUID" operation has completed.

#define GATT\_CLIENT\_EVT\_READ\_MULTIPLE\_CHAR\_VALUES\_COMPLETED 27

This event is generated when the "read multiple characteristic values" operation has completed.

#define GATT\_CLIENT\_EVT\_CONN\_PARAM\_UPDATE\_REQUEST 28

This event is generated when the client received "connection parameter update" request.

• #define GATT\_CLIENT\_EVT\_CONN\_PARAM\_UPDATE\_COMPLETED 29

This event is generated after the new connection parameters have been set.

- #define GATT\_CLIENT\_EVT\_PROFILE\_FINDER\_STARTED 100
- #define GATT\_CLIENT\_EVT\_PROFILE\_FINDER\_COMPLETED 101
- #define GATT\_CLIENT\_EVT\_PROFILE\_FOUND 102

## 3.25.1 Detailed Description

This module describes functions and data structures used to implement a GATT client (central).

### 3.25.2 Macro Definition Documentation

3.25.2.1 #define bt\_gatt\_client\_accept\_conn\_param\_update( session, min\_interval, max\_interval, slave\_latency, supervision\_timeout, min\_ce\_length, max\_ce\_length ) bt\_att\_client\_accept\_conn\_param\_← update(session->att\_session, min\_interval, max\_interval, slave\_latency, supervision\_timeout, min\_ce\_length, max\_ce\_length)

Accept connection parameters update.

When a server sends "connection parameters update" request a GATT\_CLIENT\_EVT\_CONN\_PARAM\_UPDA  $\leftarrow$  TE\_REQUEST event is generated. Client has to either accept the request or deny it. bt\_gatt\_client\_ $\leftarrow$  accept\_conn\_param\_update is used to accept the request.

### **Parameters**

session	GATT session.
min_interval	Minimum connection interval expressed in 1.25ms units.
max_interval	Maximum connection interval expressed in 1.25ms units.
slave_latency	Slave latency expressed as number of connection events.
supervision_←	Link supervision timeout expressed in 10ms units.
timeout	
min_ce_length	Information parameter about the minimum length of connection needed for this LE connection
	expressed in 0.625ms units.
max_ce_length	Information parameter about the maximum length of connection needed for this LE connec-
	tion expressed in 0.625ms units.

## Returns

- TRUE if a request has been sent.
- FALSE otherwise. No events will be generated.

3.25.2.2 #define bt\_gatt\_client\_reject\_conn\_param\_update( session ) bt\_att\_client\_reject\_conn\_param\_← update(session->att\_session)

Reject connection parameters update.

When a server sends "connection parameters update" request a GATT\_CLIENT\_EVT\_CONN\_PARAM\_UPDA  $\leftarrow$  TE\_REQUEST event is generated. Client has to either accept the request or deny it. bt\_gatt\_client\_ $\leftarrow$  reject\_conn\_param\_update is used to deny the request.

3.25 GATT Client 223

### **Parameters**

session	GATT session.
---------	---------------

## Returns

- TRUE if a request has been sent.
- FALSE otherwise. No events will be generated.

### 3.25.3 Function Documentation

3.25.3.1 bt\_gatt\_client\_session\_t\* bt\_gatt\_client\_allocate\_session ( bt\_gatt\_client\_session\_callback\_fp callback, void \* callback\_param )

Allocate GATT client session.

This function allocates a new GATT client session.

### **Parameters**

callback	The callback function that will be called when the GATT client generates an event.
callback_param	A pointer to arbitrary data to be passed to the callback callback.

#### Returns

- A pointer to the new GATT client session structure if the function succeeds.
- NULL otherwise.

 $3.25.3.2 \quad \text{bt\_bool bt\_gatt\_client\_connect (} \ \ \text{bt\_gatt\_client\_session\_t} * \textit{session,} \ \ \text{bt\_bdaddr\_t} * \textit{addr} \ \text{)}$ 

Connect to a remote device (peripheral).

This function establishes a connection to a remote device specified by the addr. If connection cannot be initiated for some reason, for example, there is not enough resources, it returns FALSE and no events are generated. Otherwise the result of an attempt to connect to the remote device is reported via the GATT client callback. The events generated will either be GATT\_CLIENT\_SESSION\_EVT\_SESSION\_CONNECTED or GATT\_CLIENT\_S ESSION\_EVT\_CONNECTION\_FAILED.

## **Parameters**

session	GATT session.
addr	The address of a remote device.

## Returns

- TRUE if connection establishment has been started.
- FALSE otherwise.

3.25.3.3 bt\_bool bt\_gatt\_client\_disconnect ( bt\_gatt\_client\_session\_t \* session )

Disconnect from a remote device.

This function closes a connection to a remote device. After the connection has been terminated the GATT client callback is called with GATT\_CLIENT\_SESSION\_EVT\_SESSION\_DISCONNECTED event.

### **Parameters**

session	GATT session.
---------	---------------

## Returns

- TRUE if disconnection has been started.
- FALSE otherwise. No events will be generated.

3.25.3.4 bt\_bool bt\_gatt\_client\_discover\_all\_chars\_ex ( bt\_gatt\_client\_session\_t \* session, const bt\_gatt\_client\_service\_definition\_t \* service, bt\_uint start\_handle, bt\_gatt\_client\_char\_declaration\_t \* result, bt\_uint max\_results )

Discover All Characteristics of a Service.

This function finds all the characteristic declarations within a service definition on a server.

#### **Parameters**

session	GATT session.
service	Service definition.
start_handle	First attribute handle from which the server has to start searching for characteristic declara-
	tions.
result	Pointer to a buffer where the client will store found characteristic declarations.
max_result	The maximum number of characteristic declarations that can be stored in result.

#### Returns

- TRUE if a request has been sent.
- FALSE otherwise. No events will be generated.
- 3.25.3.5 bt\_bool bt\_gatt\_client\_discover\_all\_services\_ex ( bt\_gatt\_client\_session\_t \* session, bt\_uint start\_handle, bt\_gatt\_client\_service\_definition\_t \* result, bt\_uint max\_results )

Discover All Services.

This function discovers all the primary services on the server.

## **Parameters**

session	GATT session.
start_handle	First attribute handle from which the server has to start searching for services.
result	Pointer to a buffer where the client will store definitions of the found services.
max_result	The maximum number of service definitions that can be stored in result.

## Returns

- TRUE if a request has been sent.
- FALSE otherwise. No events will be generated.
- 3.25.3.6 bt\_bool bt\_gatt\_client\_discover\_by\_service\_uuid ( bt\_gatt\_client\_session\_t \* session, bt\_uint service\_uuid, bt\_gatt\_client\_service\_definition\_t \* result, bt\_uint\_max\_results\_)

Discover Services by UUID (16-bit).

This function discovers a specific primary service on the server when only the Service UUID is known.

3.25 GATT Client 225

### **Parameters**

session	GATT session.
service_uuid	16-bit Service UUID.
result	Pointer to a buffer where the client will store definitions of the found services.
max_result	The maximum number of service definitions that can be stored in result.

### Returns

- TRUE if a request has been sent.
- FALSE otherwise. No events will be generated.

3.25.3.7 bt\_bool bt\_gatt\_client\_discover\_by\_service\_uuid\_80 ( bt\_gatt\_client\_session\_t \* session, bt\_uuid\_t \* service\_uuid, bt\_gatt\_client\_service\_definition\_t \* result, bt\_uint max\_results )

Discover Services by UUID (128-bit).

This function discovers a specific primary service on the server when only the Service UUID is known.

### **Parameters**

session	GATT session.
service_uuid	128-bit Service UUID.
result	Pointer to a buffer where the client will store definitions of the found services.
max_result	The maximum number of service definitions that can be stored in result.

### Returns

- TRUE if a request has been sent.
- FALSE otherwise. No events will be generated.
- 3.25.3.8 bt\_bool bt\_gatt\_client\_discover\_char\_by\_uuid ( bt\_gatt\_client\_session\_t \* session, const bt\_gatt\_client\_service\_definition\_t \* service, bt\_uint char\_uuid, bt\_gatt\_client\_char\_declaration\_t \* result, bt\_uint max\_results )

Discover Characteristics by UUID (16-bit).

This function finds characteristic declarations within a service definition on a server by the characteristic UUID.

## **Parameters**

session	GATT session.
service	Service definition.
char_uuid	16-bit characteristic UUID.
result	Pointer to a buffer where the client will store found characteristic declarations.
max_result	The maximum number of characteristic declarations that can be stored in result.

## Returns

- TRUE if a request has been sent.
- FALSE otherwise. No events will be generated.
- 3.25.3.9 bt\_bool bt\_gatt\_client\_discover\_char\_by\_uuid\_80 ( bt\_gatt\_client\_session\_t \* session, const bt\_gatt\_client\_service\_definition\_t \* service, bt\_uuid\_t \* char\_uuid, bt\_gatt\_client\_char\_declaration\_t \* result, bt\_uint max\_results )

Discover Characteristics by UUID (128-bit).

This function finds characteristic declarations within a service definition on a server by the characteristic UUID.

### **Parameters**

session	GATT session.
service	Service definition.
char_uuid	128-bit characteristic UUID.
result	Pointer to a buffer where the client will store found characteristic declarations.
max_result	The maximum number of characteristic declarations that can be stored in result.

### Returns

- TRUE if a request has been sent.
- FALSE otherwise. No events will be generated.
- 3.25.3.10 bt\_bool bt\_gatt\_client\_discover\_char\_descriptors ( bt\_gatt\_client\_session\_t \* session, const bt\_gatt\_client\_char\_declaration\_t \* characteristic, bt\_gatt\_client\_char\_descriptor\_t \* result, bt\_uint max\_results )

Discover All Characteristic Descriptors.

This function finds all the characteristic descriptors.

#### **Parameters**

session	GATT session.
characteristic	Characteristic declaration.
start_handle	First attribute handle from which the server has to start searching for characteristic declara-
	tions.
result	Pointer to a buffer where the client will store found characteristic descriptors.
max_result	The maximum number of characteristic descriptors that can be stored in result.

## Returns

- TRUE if a request has been sent.
- FALSE otherwise. No events will be generated.
- 3.25.3.11 bt\_bool bt\_gatt\_client\_find\_included\_services\_ex ( bt\_gatt\_client\_session\_t \* session, const bt\_gatt\_client\_← service\_definition\_t \* service, bt\_uint start\_handle, bt\_gatt\_client\_inc\_service\_declaration\_t \* result, bt\_uint max\_results )

Find Included Services.

This function finds include service declarations within a service definition on a server.

## **Parameters**

session	GATT session.
service	Service definition.
start_handle	First attribute handle from which the server has to start searching for include service decla-
	rations.
result	Pointer to a buffer where the client will store found include service declarations.
max_result	The maximum number of include service declarations that can be stored in result.

### Returns

- TRUE if a request has been sent.
- FALSE otherwise. No events will be generated.

3.25 GATT Client 227

3.25.3.12 void bt\_gatt\_client\_free\_sessions ( bt\_gatt\_client\_session\_t \* session )

Destroy GATT client session.

This function frees memory used by the session. The session has to exist and be in the "idle" state for this function to succeed. I.e. the session has to be disconnected before this function can be called.

### **Parameters**

```
session GATT session.
```

#### Returns

- TRUE if the function succeeds.
- FALSE otherwise.

3.25.3.13 bt\_hci\_conn\_state\_t\* bt\_gatt\_client\_get\_hci\_connection ( const bt\_gatt\_client\_session\_t \* session\_)

Get HCI connection for a session.

This function returns a pointer to a structure that describes an HCI connection a session is open on. The return value can be used to call various function from the HCI layer. For example, if an app wants to force disconnection from a remote device it can call bt\_hci\_disconnect.

### **Parameters**

```
session GATT session.
```

### Returns

- Pointer to a structure that describes an HCI connection if the function succeeds.
- NULL otherwise. The function fails only if a session specified by the session parameter
- does not exist or there is no HCI connection between local and remote devices associated with the session.

```
3.25.3.14 bt_bool bt_gatt_client_init ( void )
```

Initialize the GATT client.

This function initializes the GATT client of the stack. It must be called prior to any other GATT client function can be called.

### Returns

Always TRUE

```
3.25.3.15 bt_bool bt_gatt_client_read_by_char_uuid ( bt_gatt_client_session_t * session, const bt_gatt_client_service_definition_t * service, bt_uint char_uuid, bt_gatt_client_char_value_t * result, bt_uint max_results )
```

Read Using Characteristic UUID (16-bit).

This function reads characteristic values from a server when only the characteristic UUID is known.

### **Parameters**

session	GATT session.
service	Service definition.
char_uuid	16-bit characteristic UUID.
result	Pointer to a buffer where the client will store characteristic values.
max_result	The maximum number of characteristic values that can be stored in result.

### Returns

- TRUE if a request has been sent.
- FALSE otherwise. No events will be generated.

3.25.3.16 bt\_bool bt\_gatt\_client\_read\_by\_char\_uuid\_80 ( bt\_gatt\_client\_session\_t \* session, const bt\_gatt\_client\_service\_definition\_t \* service, bt\_uuid\_t \* char\_uuid, bt\_gatt\_client\_char\_value\_t \* result, bt\_uint max\_results )

Read Using Characteristic UUID (128-bit).

This function reads characteristic values from a server when only the characteristic UUID is known.

### **Parameters**

session	GATT session.
service	Service definition.
char_uuid	128-bit characteristic UUID.
result	Pointer to a buffer where the client will store characteristic values.
max_result	The maximum number of characteristic values that can be stored in result.

#### Returns

- TRUE if a request has been sent.
- FALSE otherwise. No events will be generated.

3.25.3.17 bt\_bool bt\_gatt\_client\_read\_char\_descriptor ( bt\_gatt\_client\_session\_t \* session, const bt\_gatt\_client\_char\_descriptor\_t \* descriptor, bt\_byte \* buffer, bt\_uint len )

Read Characteristic Descriptor.

This function reads characteristic descriptor.

### **Parameters**

session	GATT session.
descriptor	Characteristic descriptor.
buffer	Pointer to a buffer where the client will store the characteristic's descriptor.
len	The length of the buffer.

### Returns

- TRUE if a request has been sent.
- FALSE otherwise. No events will be generated.

3.25.3.18 bt\_bool bt\_gatt\_client\_read\_char\_long\_descriptor ( bt\_gatt\_client\_session\_t \* session, const bt\_gatt\_client\_char\_descriptor\_t \* descriptor, bt\_uint offset, bt\_byte \* buffer, bt\_uint len )

Read Long Characteristic Descriptor.

This function reads long characteristic descriptor.

3.25 GATT Client 229

### **Parameters**

session	GATT session.
descriptor	Characteristic descriptor.
offset	The offset of the first octet to read.
buffer	Pointer to a buffer where the client will store the characteristic's descriptor.
len	The length of the buffer.

#### Returns

- TRUE if a request has been sent.
- FALSE otherwise. No events will be generated.

```
3.25.3.19 bt_bool bt_gatt_client_read_char_long_value ( bt_gatt_client_session_t * session, const bt_gatt_client_char_declaration_t * characteristic, bt_uint offset, bt_byte * buffer, bt_uint len )
```

Read Long Characteristic Value.

This function reads characteristic value.

#### **Parameters**

session	GATT session.
characteristic	Characteristic declaration.
offset	The offset of the first octet to read.
buffer	Pointer to a buffer where the client will store the characteristic's value.
len	The length of the buffer.

### Returns

- TRUE if a request has been sent.
- FALSE otherwise. No events will be generated.

```
3.25.3.20 bt_bool bt_gatt_client_read_char_value ( bt_gatt_client_session_t * session, const bt_gatt_client_char_declaration_t * characteristic, bt_byte * buffer, bt_uint len )
```

Read Characteristic Value.

This function reads characteristic value.

### **Parameters**

session	GATT session.
characteristic	Characteristic declaration.
buffer	Pointer to a buffer where the client will store the characteristic's value.
len	The length of the buffer.

## Returns

- TRUE if a request has been sent.
- $\bullet \ \ \mathtt{FALSE}$  otherwise. No events will be generated.
- 3.25.3.21 bt\_bool bt\_gatt\_client\_read\_multiple\_char\_values ( bt\_gatt\_client\_session\_t \* session, const bt\_uint \* handles, bt\_uint count, bt\_byte \* buffer, bt\_uint len )

Read Multiple Characteristic Values.

This function reads multiple characteristic values from a server when the characteristic value handles are known.

### **Parameters**

session	GATT session.
handles	An array of characteristic value handles.
count	The number of handles.
buffer	Pointer to a buffer where the client will store the characteristic values.
len	The length of the buffer.

### Returns

- TRUE if a request has been sent.
- FALSE otherwise. No events will be generated.

```
3.25.3.22 bt_bool bt_gatt_client_set_char_value ( bt_gatt_client_session_t * session, const bt_gatt_client_char_declaration_t * characteristic, const bt_byte * value, bt_uint len )
```

Write Without Response.

This function writes characteristic value to a server using ATT "write" command, i.e. the server will not send a response after writing the value.

#### **Parameters**

session	GATT session.
characteristic	Characteristic declaration.
value	Characteristic value.
len	The length of the value.

#### Returns

- TRUE if a request has been sent.
- $\bullet \;\; \mathtt{FALSE}$  otherwise. No events will be generated.

```
3.25.3.23 bt_bool bt_gatt_client_write_char_descriptor ( bt_gatt_client_session_t * session, const bt_gatt_client_char_descriptor_t * descriptor, const bt_byte * value, bt_uint len )
```

Write Characteristic Descriptor.

This function writes characteristic descriptor.

## Parameters

session	GATT session.
descriptor	Characteristic descriptor.
value	Characteristic descriptor value.
len	The length of the value.

## Returns

- TRUE if a request has been sent.
- FALSE otherwise. No events will be generated.

```
3.25.3.24 bt_bool bt_gatt_client_write_char_long_descriptor ( bt_gatt_client_session_t * session, const bt_gatt_client_char_descriptor_t * descriptor, bt_uint value_offset, const bt_byte * value, bt_uint len )
```

Write Long Characteristic Descriptor.

This function writes characteristic descriptor.

3.25 GATT Client 231

### **Parameters**

session	GATT session.
descriptor	Characteristic descriptor.
value_offset	The offset of the first octet to be written.
value	Characteristic descriptor value.
len	The length of the buffer.

### Returns

- TRUE if a request has been sent.
- FALSE otherwise. No events will be generated.

```
3.25.3.25 bt_bool bt_gatt_client_write_char_long_value ( bt_gatt_client_session_t * session, const bt_gatt_client_char_declaration_t * characteristic, bt_uint value_offset, const bt_byte * value, bt_uint len )
```

Write Long Characteristic Value.

This function writes characteristic value to a server using ATT "prepare write" and "execute write" requests.

### **Parameters**

session	GATT session.
characteristic	Characteristic declaration.
value_offset	The offset of the first octet to be written.
value	Characteristic value.
len	The length of the value.

#### Returns

- TRUE if a request has been sent.
- FALSE otherwise. No events will be generated.

```
3.25.3.26 bt_bool bt_gatt_client_write_char_value ( bt_gatt_client_session_t * session, const bt_gatt_client_char_declaration_t * characteristic, const bt_byte * value, bt_uint len )
```

Write Characteristic Value.

This function writes characteristic value to a server using ATT "write" request, i.e. the server will send a response after writing the value.

## **Parameters**

session	GATT session.
characteristic	Characteristic declaration.
value	Characteristic value.
len	The length of the value.

## Returns

- TRUE if a request has been sent.
- FALSE otherwise. No events will be generated.

3.25.3.27 bt\_bool bt\_gatt\_exchange\_mtu ( bt\_gatt\_client\_session\_t \* session, bt\_uint mtu )

## Exchange MTU.

This function informs the server about the client's MTU. In response to the "exchange MTU" request the server sends its MTU to the client.

### **Parameters**

session	GATT session.
mtu	Client's MTU.

## Returns

- TRUE if a request has been sent.
- FALSE otherwise. No events will be generated.

3.25.3.28 bt\_gatt\_client\_mgr\_t\* bt\_gatt\_get\_client\_mgr ( void )

Return a pointer to an instance of the GATT client manager.

This function returns a pointer to an instance of the GATT client manager. There is only one instance of the manager allocated by the stack.

### Returns

• A pointer to the GATT client manager.

3.26 Configuration 233

## 3.26 Configuration

This module describes parameters used to configure GATT client.

This module describes parameters used to configure GATT client.

dotstack is customized using a configuration file. The configuration file tailors the dotstack to the application being built. It has to have the structure shown below.

```
#include "cdbt/bt/bt std.h"
// HCI and L2CAP must always be present
// SDP is required only if stack is running in dual mode. This is the default mode.
// To run the stack in single mode (i.e. only BLE is supported) a BT_BLE_SINGLE_MODE symbol
// must be defined:
// #define BT_BLE_SINGLE_MODE
// HCI configuration parameters
#define HCI_MAX_CMD_BUFFERS
#define HCI_MAX_DATA_BUFFERS
#define HCI_MAX_HCI_CONNECTIONS
#define HCI_RX_BUFFER_LEN
#define HCI_TX_BUFFER_LEN
#define HCI L2CAP BUFFER LEN
#define HCI_MAX_CMD_PARAM_LEN
                                       . . .
// L2CAP configuration parameters
#define L2CAP_MAX_CMD_BUFFERS
#define L2CAP_MAX_FRAME_BUFFERS
#define L2CAP MAX PSMS
#define L2CAP_MAX_CHANNELS
// SDP configuration parameters
#define SDP_MAX_SEARCH_RESULT_LEN
#define SDP_MAX_ATTRIBUTE_RESULT_LEN ...
// Depending on protocols and profiles used below goes configuration parameters
// for each used module. E.g., to use and configure GATT,
// the following values must be defined:
#define BT_INCLUDE_ATT
                                    // tells dotstack to compile in ATT support
#define ATT_CLIENT_MAX_SESSIONS
#define BT_INCLUDE_GATT_CLIENT
                                    // tells dotstack to compile in GATT support
// By default all GATT client functions can be used by the application. In most cases only a subset of the
      API will be used.
// To to save quite significant amount of code space unused functions can be disabled. To disable a
      function the correspondign symbol
// must be defined in this configuration file. For example do disable bt_gatt_client_read_by_char_uuid the
      GATT_NO_READ_BY_CHAR_UUID
// must be defined:
    #define GATT_NO_READ_BY_CHAR_UUID
#include "cdbt/bt/bt oem config.h"
```

- · GATT NO DISCOVER ALL SERVICES Disable bt gatt client discover all services ex()
- GATT\_NO\_DISCOVER\_BY\_SERVICE\_UUID Disable bt\_gatt\_client\_discover\_by\_service\_uuid()
- GATT\_NO\_FIND\_INCLUDED\_SERVICES Disable bt\_gatt\_client\_find\_included\_services\_ex()
- GATT\_NO\_DISCOVER\_ALL\_CHARS Disable bt\_gatt\_client\_discover\_all\_chars\_ex()
- GATT\_NO\_DISCOVER\_CHAR\_BY\_UUID Disable bt\_gatt\_client\_discover\_char\_by\_uuid()
- GATT\_NO\_READ\_CHAR\_VALUE Disable bt\_gatt\_client\_read\_char\_value()
- GATT\_NO\_READ\_BY\_CHAR\_UUID Disable bt\_gatt\_client\_read\_by\_char\_uuid()
- GATT\_NO\_READ\_MULTIPLE\_CHAR\_VALUES Disable bt\_gatt\_client\_read\_multiple\_char\_values()
- GATT\_NO\_READ\_CHAR\_DESCRIPTOR Disable bt\_gatt\_client\_read\_char\_descriptor()
- GATT\_NO\_READ\_CHAR\_LONG\_VALUE Disable bt\_gatt\_client\_read\_char\_long\_value()
- GATT\_NO\_READ\_CHAR\_LONG\_DESCRIPTOR Disable bt\_gatt\_client\_read\_char\_long\_descriptor()

- GATT\_NO\_WRITE\_CHAR\_VALUE Disable bt\_gatt\_client\_write\_char\_value()
- GATT\_NO\_WRITE\_CHAR\_DESCRIPTOR Disable bt\_gatt\_client\_write\_char\_descriptor()
- GATT\_NO\_WRITE\_CHAR\_LONG\_VALUE Disable bt\_gatt\_client\_write\_char\_long\_value()
- GATT\_NO\_WRITE\_CHAR\_LONG\_DESCRIPTOR Disable bt\_gatt\_client\_write\_char\_long\_descriptor()
- GATT\_NO\_DISCOVER\_CHAR\_DESCRIPTORS Disable bt\_gatt\_client\_discover\_char\_descriptors()

3.27 Security Manager 235

# 3.27 Security Manager

This module describes functions and data structures used to implement Security Manager.

This module describes functions and data structures used to implement Security Manager.

## 3.28 OEM - HCI Communication Interface

### **Modules**

· HCI UART (H4) transport protocol

## **Typedefs**

- typedef void(\* bt\_oem\_send\_callback\_fp) (void)
   Send callback.
- typedef void(\* bt\_oem\_recv\_callback\_fp) (bt\_uint len)
   Receive callback.

### **Functions**

- void bt\_oem\_send (const bt\_byte \*buffer, bt\_uint len, bt\_oem\_send\_callback\_fp callback)
   Send data.
- void bt\_oem\_recv (bt\_byte \*buffer, bt\_uint len, bt\_oem\_recv\_callback\_fp callback)

  \*\*Receive data.\*

## 3.28.1 Detailed Description

The DotStack library provides several HCI transport protocols (e.g., H4, 3-wire protocol). However, the code that actually moves octets of data between the CPU and HCI controller is application specific.

This module declares an interface that allows DotStack to communicate with the HCI controller. The application has to implement this interface.

The interface consist of the following functions that must be implemented by the application:

- bt\_oem\_send()
- bt\_oem\_recv()

## 3.28.2 Typedef Documentation

3.28.2.1 typedef void(\* bt\_oem\_recv\_callback\_fp) (bt\_uint len)

Receive callback.

This callback function is called when a receive operation initiated by bt\_oem\_recv() has completed.

### **Parameters**

len	Number of received bytes. The value of this parameter should always be the same as the
	number of bytes requested in a call to bt_oem_recv().

3.28.2.2 typedef void(\* bt\_oem\_send\_callback\_fp) (void)

Send callback.

This callback function is called when a send operation initiated by bt\_oem\_send() has completed.

## 3.28.3 Function Documentation

3.28.3.1 void bt\_oem\_recv ( bt\_byte \* buffer, bt\_uint len, bt\_oem\_recv\_callback\_fp callback )

## Receive data.

This function is called by the HCI layer when it needs more data from the HCI controller. Implementation of this function must receive the specified number of bytes from the HCI controller and call the provided callback function.

### **Parameters**

buffer	Pointer to a buffer for the received data. The buffer must be long enough to accommodate
	the number of bytes specified by the

len parameter.

## **Parameters**

len	Number of bytes to receive.
callback	A callback function that must be called when the requested number of bytes have been re-
	ceived.

3.28.3.2 void bt\_oem\_send ( const bt\_byte \* buffer, bt\_uint len, bt\_oem\_send\_callback\_fp callback )

## Send data.

This function is called by the HCI layer when it needs to send data to the HCI controller. Implementation of this function must send the specified number of bytes to the HCI controller and call the provided callback function.

buffer	Pointer to the data to be sent .
len	Number of bytes to send.
callback	A callback function that must be called when all data have been sent.

## 3.29 OEM - Timer Interface

## **Typedefs**

typedef void(\* bt\_timer\_callback\_fp) (void)

Timer callback.

## **Functions**

void bt\_oem\_timer\_set (bt\_uint timerId, bt\_ulong milliseconds, bt\_timer\_callback\_fp callback)

• void bt\_oem\_timer\_clear (bt\_uint timerId)

Clear timer.

## 3.29.1 Detailed Description

DotStack requires a facility to measure various time intervals. This module declares an interface that the application has to implement in order to provide DotStack with such functionality.

The minimum required timer resolution is 100 milliseconds.

The maximum number of timers is defined by the BT\_TIMER\_MAX constant.

Timer ID values used in the interface are from 0 to BT\_TIMER\_MAX-1.

The interface consists of the following function:

- bt\_oem\_timer\_set()
- bt\_oem\_timer\_clear()

## 3.29.2 Typedef Documentation

3.29.2.1 typedef void(\* bt\_timer\_callback\_fp) (void)

Timer callback.

This callback is called when a timer expires.

## 3.29.3 Function Documentation

3.29.3.1 void bt\_oem\_timer\_clear ( bt\_uint timerId )

Clear timer.

This function must be implemented by the application. When this function is called the application must clear the specified timer. If it is already expired and a callback is currently pending, the application should also take measures to cancel the callback.

timerId	ID of the timer to clear.	

3.29 OEM - Timer Interface 239

3.29.3.2 void bt\_oem\_timer\_set ( bt\_uint timerId, bt\_ulong milliseconds, bt\_timer\_callback\_fp callback )

## Set timer.

This function must be implemented by the application. When it is called, the application must set the specified timer. When the timer expires, the application must call the passed callback function. The function must not wait until the timer expires. It must set the timer and exit immediately.

timerId	ID of the timer to set.
milliseconds	Timer interval in milliseconds
callback	Timer expiration callback function.

## 3.30 OEM - Non-volatile Storage Interface

## **Functions**

bt\_uint bt\_oem\_storage\_get\_capacity (void)

Get non-volatile storage capacity.

void bt\_oem\_storage\_start (void)

Begin a sequence of non-volatile storage operations.

void bt\_oem\_storage\_stop (void)

End a sequence of non-volatile storage operations.

- void bt\_oem\_storage\_write (bt\_int addr, const bt\_byte \*data, bt\_int len, bt\_storage\_callback\_fp callback)
   Write to non-volatile storage.
- void bt\_oem\_storage\_read (bt\_int addr, bt\_byte \*buffer, bt\_int len, bt\_storage\_callback\_fp callback)

  Read from the non-volatile storage.

## 3.30.1 Detailed Description

DotStack requires a non-volatile storage for storing link keys. This module declares an interface for accessing such storage. The application must provide implementations of all functions of this interface.

### 3.30.2 Function Documentation

3.30.2.1 bt\_uint bt\_oem\_storage\_get\_capacity ( void )

Get non-volatile storage capacity.

Implementation of this function must return the capacity of its non-volatile storage.

3.30.2.2 void bt\_oem\_storage\_read ( bt\_int addr, bt\_byte \* buffer, bt\_int len, bt\_storage\_callback\_fp callback )

Read from the non-volatile storage.

This function is called by the stack to read from the non-volatile storage. This function must be implemented by the application. When this function is called the application must start a read operation. When the number of bytes specified by the len parameter is read, the application must call the callback function specified by the callback parameter. The application does not have to read the whole number of bytes during the call to this function. It may complete reading later and then call the completion callback. The stack guarantees that the destination data buffer will be available until the application calls the completion callback.

## Parameters

addr	The non-volatile storage address where to read data from.
buffer	The receiving buffer.
len	The number of bytes to read.
callback	The completion callback function.

3.30.2.3 void bt\_oem\_storage\_start ( void )

Begin a sequence of non-volatile storage operations.

DotStack calls this function when it starts a sequence of non-volatile storage operations. When the sequence is finished, DotStack will call bt\_oem\_storage\_stop().

3.30.2.4 void bt\_oem\_storage\_stop (void )

End a sequence of non-volatile storage operations.

DotStack calls this function when it finishes executing a sequence of non-volatile storage operations.

3.30.2.5 void bt\_oem\_storage\_write ( bt\_int addr, const bt\_byte \* data, bt\_int len, bt\_storage\_callback\_fp callback )

Write to non-volatile storage.

This function is called by the stack to write data to the non-volatile storage. This function must be implemented by the application. When this function is called, the application must start writing specified data to the non-volatile storage. When all data has been written, the application must call the callback function passed in the callback parameter. The application does not have to complete the write operation during the call to this function. It may complete the operation later and then call the callback function. In this case, the application does not have to store the data in an internal buffer. The stack guarantees that the passed data will be present until the completion callback is called by the application.

addr	The persitent storage address where to write data to.
data	Pointer to data.
len	Data length.
callback	The completion callback function.

## 3.31 OEM - Logging Interface

## **Functions**

void bt\_oem\_log\_write (const char \*msg)
 Output log message.

## 3.31.1 Detailed Description

This module declares an interface that DotStack is using to output its debug and diagnostic information. The application must implement this interface.

Though the simplest implementation may consis of just empty stubs it is strongly encouraged to provide a useful implementation that allows for viewing and capturing of the data passed through this interface.

## 3.31.2 Function Documentation

3.31.2.1 void bt\_oem\_log\_write ( const char \* msg )

Output log message.

DotStack calls this function to output its debug information. Implementation should output or store the specified message to whatever device or medium where it can be examined and analyzed.

# 3.32 Vendor specific extensions to HCI

## **Modules**

• CSR

## 3.32.1 Detailed Description

This module defines functions and data structures used to access and control various capabilities of CSR's controllers.

## 3.33 CSR

## **Functions**

void btx csr init (void)

Initialize CSR support layer.

void btx\_csr\_autobaud (btx\_csr\_autobaud\_buffer\_t \*buffer, btx\_csr\_autobaud\_callback\_fp callback, void \*callback\_param)

Configure controller's UART speed.

void btx\_csr\_bc7\_sel\_host\_interface\_h4 (btx\_csr\_autobaud\_buffer\_t \*buffer, bt\_byte interval, btx\_csr\_
 autobaud\_callback\_fp callback, void \*callback\_param)

Configure controller's UART speed and host interface.

void btx\_csr\_exec\_script (const btx\_csr\_script\_t \*script, btx\_csr\_exec\_script\_buffer\_t \*buffer, btx\_csr\_
 exec\_script\_callback\_fp callback, void \*callback\_param)

Patch controller's firmware.

void btx\_csr\_set\_ps\_vars (const bt\_uint \*ps\_vars, btx\_csr\_set\_ps\_vars\_buffer\_t \*buffer, btx\_csr\_set\_ps\_
 vars\_callback\_fp callback, void \*callback\_param)

Write PS variables.

 void btx\_csr\_set\_ps\_vars\_ex (const bt\_uint \*ps\_vars, btx\_csr\_set\_ps\_vars\_buffer\_t \*buffer, bt\_uint store, btx\_csr\_set\_ps\_vars\_callback\_fp callback, void \*callback\_param)

Write PS variables.

bt\_bool btx\_csr\_warm\_reset (void)

Warm reset.

• bt\_bool btx\_csr\_warm\_reset\_ex (bt\_hci\_cmd\_callback\_fp callback, void \*callback\_param)

Warm reset.

• bt\_bool btx\_csr\_enable\_tx (bt\_bool enable, bt\_hci\_cmd\_callback\_fp callback, void \*callback\_param)

Enable/disable transmitter.

• bt\_bool btx\_csr\_get\_cached\_temperature (btx\_csr\_get\_var\_callback\_fp callback, void \*callback\_param)

Get chip's cached temperature.

bt\_bool btx\_csr\_get\_rssi\_acl (bt\_hci\_hconn\_t hconn, btx\_csr\_get\_var\_callback\_fp callback, void \*callback → \_param)

Get RSSI.

• void btx\_csr\_patch\_controller (const btx\_csr\_get\_script\_fp \*scripts, bt\_int script\_count, btx\_csr\_exec\_ 
script\_buffer\_t \*buffer, btx\_csr\_exec\_script\_callback\_fp callback, void \*callback\_param)

Patch controller's firmware.

• const btx\_csr\_script\_t \* btx\_csr\_get\_script\_\_PB\_27\_R20\_BC6ROM\_A04 (void)

Return script for patching BlueCore 6.

const btx\_csr\_script\_t \* btx\_csr\_get\_script\_\_PB\_90\_REV6 (void)

Return script for patching CSR8810 (BlueCore 7)

const btx\_csr\_script\_t \* btx\_csr\_get\_script\_\_PB\_101\_CSR8811\_CSP28\_UART (void)

Return script for patching CSR8x11 A06 (BlueCore 7)

• const btx\_csr\_script\_t \* btx\_csr\_get\_script\_\_PB\_109\_CSR8811\_REV16 (void)

Return script for patching CSR8x11 A08 (BlueCore 7)

• const btx\_csr\_script\_t \* btx\_csr\_get\_script\_\_dsp\_script\_\_PB\_109\_DSP\_rev8 (void)

Return script for patching DSP in CSR8x11 A08 (BlueCore 7)

• const btx\_csr\_script\_t \* btx\_csr\_get\_script\_\_PB\_173\_CSR8X11\_REV1 (void)

Return script for patching CSR8x11 A12 (BlueCore 7)

3.33 CSR 245

## 3.33.1 Detailed Description

## 3.33.2 Function Documentation

3.33.2.1 void btx\_csr\_autobaud ( btx\_csr\_autobaud\_buffer\_t \* buffer, btx\_csr\_autobaud\_callback\_fp callback, void \* callback\_param )

Configure controller's UART speed.

This function makes the controller auto-configure its UART speed. The host transport must be set to H4. This function works only with BC6 controllers.

3.33.2.2 void btx\_csr\_bc7\_sel\_host\_interface\_h4 ( btx\_csr\_autobaud\_buffer\_t \* buffer, bt\_byte interval, btx\_csr\_autobaud\_callback\_fp callback, void \* callback\_param )

Configure controller's UART speed and host interface.

This function makes the controller auto-configure its UART speed and select H4 as host interface. PS\_KEY\_HO 
ST\_INTERFACE must not be set. PS\_KEY\_UART\_BITRATE must be set to 0. This function works only with BC7 controllers.

3.33.2.3 bt\_bool btx\_csr\_enable\_tx ( bt\_bool enable, bt\_hci\_cmd\_callback\_fp callback, void \* callback\_param )

Enable/disable transmitter.

### **Parameters**

enable	Specifies whether the transmitter should be enable or disabled.
callback	The callback function that will be called after the command has completed.
callback_param	A pointer to arbitrary data to be passed to the callback callback.

3.33.2.4 void btx\_csr\_exec\_script ( const btx\_csr\_script\_t \* script, btx\_csr\_exec\_script\_buffer\_t \* buffer, btx\_csr\_exec\_script\_callback\_fp callback, void \* callback\_param )

Patch controller's firmware.

This function executes a script that patches the controller's firmware. The <code>script</code> must point to a structure that contain a complete patch script for a particular controller model and revision. If the revision specified in the script and revision read from the controller are the same <code>btx\_csr\_patch\_controller()</code> loads the script to the controller and calls the <code>callback</code> with the first parameter TRUE. Otherwise the <code>callback</code> is called with the first parameter FALSE.

If support for multiple firmware revisions is neede use <a href="https://bx.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm.ncbi.nlm

### **Parameters**

script	Array of patch scripts.
buffer	A buffer for storing temporary data needed for script execution.
callback	The callback function that will be called when the script has been executed.
callback_param	A pointer to arbitrary data to be passed to the callback callback

3.33.2.5 bt\_bool btx\_csr\_get\_cached\_temperature ( btx\_csr\_get\_var\_callback\_fp callback, void \* callback\_param )

Get chip's cached temperature.

### **Parameters**

callback	The callback function that will be called after the command has completed.
callback_param	A pointer to arbitrary data to be passed to the callback callback.

3.33.2.6 bt\_bool btx\_csr\_get\_rssi\_acl ( bt\_hci\_hconn\_t hconn, btx\_csr\_get\_var\_callback\_fp callback, void \* callback\_param )

### Get RSSI.

This function retrieves the RSSI for a given HCI ACL handle.

#### **Parameters**

hconn	ACL connection handle.
callback	The callback function that will be called after the command has completed.
callback_param	A pointer to arbitrary data to be passed to the callback callback.

3.33.2.7 void btx\_csr\_init ( void )

Initialize CSR support layer.

This function initializes all internal variables of the CSR support layer. CSR controllers use vendor specific event (0xFF) to carry the BCCMD protocol. They also do not report number of completed packets for BCCMD commands. This function installs a vendor specific event handler that makes sure that callback are called on corresponding vendor specific commands and the number of free command buffers in the controller is kept correct.

3.33.2.8 void btx\_csr\_patch\_controller ( const btx\_csr\_get\_script\_fp \* scripts, bt\_int script\_count, btx\_csr\_exec\_script\_buffer\_t \* buffer, btx\_csr\_exec\_script\_callback\_fp callback, void \* callback\_param )

Patch controller's firmware.

This function executes a script that patches the controller's firmware. Each entry of the <code>scripts</code> array must be a complete patch script for a particular controller model and revision. <code>btx\_csr\_patch\_controller()</code> reads the revision number from the controller then tries to find the corresponding script in the <code>scripts</code>. If there is a matching script it is loaded to the controller and <code>callback</code> is called with the first parameter TRUE. If no suitable script found <code>callback</code> is called with the first parameter FALSE.

## **Parameters**

scripts	Array of patch scripts.
script_count	The number of scripts in scripts.
buffer	A buffer for storing temporary data needed for script execution.
callback	The callback function that will be called when the script has been executed.
callback_param	A pointer to arbitrary data to be passed to the callback callback

3.33.2.9 void btx\_csr\_set\_ps\_vars ( const bt\_uint \* ps\_vars, btx\_csr\_set\_ps\_vars\_buffer\_t \* buffer, btx\_csr\_set\_ps\_vars\_callback\_fp callback, void \* callback\_param )

Write PS variables.

ps_vars	PS values

3.33 CSR 247

buffer	A buffer for storing temporary data during function execution.
callback	The callback function that will be called when all PS values have been sent to the controller
	or error occurred.
callback_param	A pointer to arbitrary data to be passed to the callback callback

3.33.2.10 void btx\_csr\_set\_ps\_vars\_ex ( const bt\_uint \* ps\_vars, btx\_csr\_set\_ps\_vars\_buffer\_t \* buffer, bt\_uint store, btx\_csr\_set\_ps\_vars\_callback\_fp callback, void \* callback\_param )

### Write PS variables.

### **Parameters**

ps_vars	PS values
buffer	A buffer for storing temporary data during function execution.
store	
callback	The callback function that will be called when all PS values have been sent to the controller
	or error occurred.
callback_param	A pointer to arbitrary data to be passed to the callback callback

3.33.2.11 bt\_bool btx\_csr\_warm\_reset ( void )

## Warm reset.

This function performs warm reset of the controller. All patches and configuration parameters sent to the controller before warm reset are kept intact.

3.33.2.12 bt\_bool btx\_csr\_warm\_reset\_ex ( bt\_hci\_cmd\_callback\_fp callback, void \* callback\_param )

## Warm reset.

This function performs warm reset of the controller. All patches and configuration parameters sent to the controller before warm reset are kept intact. Since the controller does not respond to the warm reset command as it starts resetting immediately upon receiving the command, the callback is called right after the command packet has been transmitted to the controller.

callback	The callback function that will be called after the warm reset command has been sent to the
	controller.
callback_param	A pointer to arbitrary data to be passed to the callback callback.

# **Chapter 4**

# **Data Structure Documentation**

- 4.1 bt\_spp\_port\_t::\_bt\_spp\_port\_flags\_t Struct Reference
- 4.2 bt\_a2dp\_aac\_config\_t Struct Reference
- 4.3 bt a2dp event t Union Reference

Parameter to an application callback.

```
#include <a2dp.h>
```

## **Data Fields**

- bt\_a2dp\_evt\_open\_and\_start\_stream\_completed\_t open\_and\_start\_stream\_completed Valid if event is A2DP\_EVT\_OPEN\_AND\_START\_STREAM\_COMPLETED.
- bt\_avdtp\_evt\_ctrl\_channel\_connected\_t ctrl\_channel\_connected Valid if event is A2DP\_EVT\_CTRL\_CHANNEL\_CONNECTED.
- bt\_avdtp\_evt\_ctrl\_channel\_disconnected\_t ctrl\_channel\_disconnected Valid if event is A2DP\_EVT\_CTRL\_CHANNEL\_DISCONNECTED.
- bt\_avdtp\_evt\_discover\_completed\_t discover\_completed

Valid if event is A2DP EVT DISCOVER COMPLETED.

- bt\_avdtp\_evt\_sep\_info\_received\_t sep\_info\_received
  - Valid if event is A2DP\_EVT\_SEP\_INFO\_RECEIVED.
- bt\_avdtp\_evt\_get\_sep\_capabilities\_completed\_t get\_sep\_capabilities\_completed Valid if event is A2DP\_EVT\_GET\_SEP\_CAPABILITIES\_COMPLETED.
- bt\_avdtp\_evt\_sep\_capabilities\_received\_t sep\_capabilities\_received Valid if event is A2DP\_EVT\_SEP\_CAPABILITIES\_RECEIVED.
- bt\_avdtp\_evt\_set\_stream\_configuration\_completed\_t set\_stream\_configuration\_completed Valid if event is A2DP\_EVT\_SET\_STREAM\_CONFIGURATION\_COMPLETED.
- bt\_avdtp\_evt\_get\_stream\_configuration\_completed\_t get\_stream\_configuration\_completed
   Valid if event is A2DP\_EVT\_GET\_STREAM\_CONFIGURATION\_COMPLETED.
- bt\_avdtp\_evt\_stream\_reconfigure\_completed\_t stream\_reconfigure\_completed

Valid if event is A2DP\_EVT\_STREAM\_RECONFIGURE\_COMPLETED.

- bt\_avdtp\_evt\_open\_stream\_completed\_t open\_stream\_completed Valid if event is A2DP\_EVT\_OPEN\_STREAM\_COMPLETED.
- bt\_avdtp\_evt\_start\_stream\_completed\_t start\_stream\_completed Valid if event is A2DP\_EVT\_START\_STREAM\_COMPLETED.

- bt\_avdtp\_evt\_close\_stream\_completed\_t close\_stream\_completed Valid if event is A2DP\_EVT\_CLOSE\_STREAM\_COMPLETED.
- bt\_avdtp\_evt\_suspend\_stream\_completed\_t suspend\_stream\_completed

Valid if event is A2DP\_EVT\_SUSPEND\_STREAM\_COMPLETED.

• bt\_avdtp\_evt\_stream\_security\_control\_completed\_t security\_control\_completed

Valid if event is A2DP\_EVT\_STREAM\_SECURITY\_CONTROL\_COMPLETED.

bt\_avdtp\_evt\_set\_stream\_configuration\_requested\_t set\_stream\_configuration\_requested

Valid if event is A2DP\_EVT\_SET\_STREAM\_CONFIGURATION\_REQUESTED.

• bt\_avdtp\_evt\_reconfigure\_stream\_requested\_t reconfigure\_stream\_requested

Valid if event is A2DP\_EVT\_RECONFIGURE\_STREAM\_REQUESTED.

• bt\_avdtp\_evt\_open\_stream\_requested\_t open\_stream\_requested

Valid if event is A2DP\_EVT\_OPEN\_STREAM\_REQUESTED.

bt\_avdtp\_evt\_start\_stream\_requested\_t start\_stream\_requested

Valid if event is A2DP EVT START STREAM REQUESTED.

• bt\_avdtp\_evt\_suspend\_stream\_requested\_t suspend\_stream\_requested

Valid if event is A2DP EVT SUSPEND STREAM REQUESTED.

• bt\_avdtp\_evt\_close\_stream\_requested\_t close\_stream\_requested

Valid if event is A2DP\_EVT\_CLOSE\_STREAM\_REQUESTED.

• bt\_avdtp\_evt\_abort\_stream\_requested\_t abort\_stream\_requested

Valid if event is A2DP\_EVT\_ABORT\_STREAM\_REQUESTED.

· bt avdtp evt stream configured t stream configured

Valid if event is A2DP\_EVT\_STREAM\_CONFIGURED.

• bt\_avdtp\_evt\_stream\_reconfigured\_t stream\_reconfigured

Valid if event is A2DP\_EVT\_STREAM\_RECONFIGURED.

• bt\_avdtp\_evt\_stream\_opened\_t stream\_opened

Valid if event is A2DP\_EVT\_STREAM\_OPENED.

• bt\_avdtp\_evt\_stream\_started\_t stream\_started

Valid if event is A2DP\_EVT\_STREAM\_STARTED.

bt\_avdtp\_evt\_stream\_suspended\_t stream\_suspended

Valid if event is A2DP\_EVT\_STREAM\_SUSPENDED.

bt\_avdtp\_evt\_stream\_closed\_t stream\_closed

Valid if event is A2DP\_EVT\_STREAM\_CLOSED.

bt\_avdtp\_evt\_stream\_aborted\_t stream\_aborted

Valid if event is A2DP\_EVT\_STREAM\_ABORTED.

## 4.3.1 Detailed Description

Parameter to an application callback.

This union is used to pass event specific data to the A2DP consumer. Which member of the union points to a valid structure depends on the event reported to the consumer. In general, each event has a corresponding member in the union.

## 4.4 bt\_a2dp\_evt\_open\_and\_start\_stream\_completed\_t Struct Reference

Parameter to A2DP EVT OPEN AND START STREAM COMPLETED event.

#include <a2dp.h>

## **Data Fields**

bt\_byte failed\_cmd

ID of the failed command.

• bt\_byte err\_code

The result of the request.

· bt\_byte strm\_handle

Stream handle.

• bt\_avdtp\_evt\_set\_stream\_configuration\_completed\_t set\_config

If "set configuration" request failed this member contains description of an error returned by the remote party.

• bt\_avdtp\_evt\_open\_stream\_completed\_t open

If open stream" request failed this member contains description of an error returned by the remote party.

bt\_avdtp\_evt\_start\_stream\_completed\_t start

If "start stream" request failed this member contains description of an error returned by the remote party.

## 4.4.1 Detailed Description

Parameter to A2DP\_EVT\_OPEN\_AND\_START\_STREAM\_COMPLETED event.

A pointer to this structure is passed to the A2DP application callback as a valid member of the bt\_a2dp\_event \_t union - bt\_a2dp\_event\_t::open\_and\_start\_stream\_completed - when A2DP completed a "open & start stream" request.

### 4.4.2 Field Documentation

## 4.4.2.1 bt\_byte err\_code

The result of the request.

- If the remote accepted all 3 requests sent during execution of the "open & start" request err\_code == AVDTP\_ERROR\_SUCCESS.
- Otherwise err\_code == the error code returned by the remote party.

### 4.4.2.2 bt byte failed cmd

ID of the failed command.

This can be one of the following values:

- AVDTP\_CMD\_SET\_CONFIGURATION
- AVDTP\_CMD\_OPEN
- AVDTP\_CMD\_START

## 4.5 bt\_a2dp\_mgr\_t Struct Reference

## A2DP manager.

#include <a2dp.h>

## **Data Fields**

· bt\_byte state

Manager state.

bt\_avdtp\_mgr\_t \* avdtp\_mgr

AVDTP manager.

• bt\_avdtp\_stream\_t \* connection\_stream

Pointer to a stream being open with bt\_a2dp\_open\_and\_start\_stream.

bt\_a2dp\_mgr\_callback\_fp callback

Pointer to a function which a A2DP consumer must register in order to be notified of various events.

void \* callback param

Pointer to arbitrary data to be passed to the callback.

## 4.5.1 Detailed Description

## A2DP manager.

A structure that glues all pieces together. There is only one instance of this structure allocated by dotstack. A pointer to the instance can be get with bt\_a2dp\_get\_mgr.

### 4.5.2 Field Documentation

4.5.2.1 bt\_byte state

Manager state.

This value can be one of the following values:

- A2DP\_MANAGER\_STATE\_IDLE
- · A2DP MANAGER STATE CONNECTING
- 4.6 bt a2dp mpeg config t Struct Reference
- 4.7 bt\_a2dp\_sbc\_config\_t Struct Reference
- 4.8 bt\_a2dp\_sbc\_packet\_info\_t Struct Reference
- 4.9 bt\_att\_attribute\_t Struct Reference
- 4.10 bt\_att\_client\_evt\_conn\_param\_update\_completed\_t Struct Reference

Parameter to ATT\_CLIENT\_EVT\_CONN\_PARAM\_UPDATE\_COMPLETED event.

```
#include <att_client.h>
```

## **Data Fields**

• bt\_byte status

Operation status.

· bt\_int hci\_status

HCI command status.

## 4.10.1 Detailed Description

Parameter to ATT\_CLIENT\_EVT\_CONN\_PARAM\_UPDATE\_COMPLETED event.

A pointer to this structure is passed to the ATT client application callback after the new connection parameters have been set.

## 4.11 bt\_att\_client\_evt\_conn\_param\_update\_t Struct Reference

Parameter to ATT CLIENT EVT CONN PARAM UPDATE REQUEST event.

```
#include <att_client.h>
```

## **Data Fields**

· bt uint min interval

Minimum connection interval expressed in 1.25ms units.

• bt\_uint max\_interval

Maximum connection interval expressed in 1.25ms units.

bt\_uint slave\_latency

Slave latency expressed as number of connection events.

bt\_uint supervision\_timeout

Link supervision timeout expressed in 10ms units.

## 4.11.1 Detailed Description

Parameter to ATT\_CLIENT\_EVT\_CONN\_PARAM\_UPDATE\_REQUEST event.

A pointer to this structure is passed to the ATT client application callback when client received update connection parameters request.

## 4.12 bt\_att\_client\_evt\_execute\_write\_response\_t Struct Reference

Parameter to ATT\_CLIENT\_EVT\_EXECUTE\_WRITE\_RESPONSE event.

```
#include <att_client.h>
```

### **Data Fields**

• bt\_byte status

Operation status.

bt\_uint err\_handle

The attribute handle that generated error response.

## 4.12.1 Detailed Description

Parameter to ATT\_CLIENT\_EVT\_EXECUTE\_WRITE\_RESPONSE event.

A pointer to this structure is passed to the ATT client application callback when client received a response (either positive or negative) to a "execute write" request.

## 4.13 bt\_att\_client\_evt\_find\_by\_type\_value\_response\_t Struct Reference

Parameter to ATT CLIENT EVT FIND BY TYPE VALUE RESPONSE event.

```
#include <att_client.h>
```

### **Data Fields**

• bt\_byte status

Operation status.

• bt\_uint start\_handle

First attribute handle found.

· bt uint end handle

Last attribute handle found.

· bt uint count

The number of attribute handles found.

## 4.13.1 Detailed Description

Parameter to ATT CLIENT EVT FIND BY TYPE VALUE RESPONSE event.

A pointer to this structure is passed to the ATT client application callback when client received a response (either positive or negative) to a "find by type value info" request.

## 4.14 bt\_att\_client\_evt\_info\_response\_t Struct Reference

Parameter to ATT\_CLIENT\_EVT\_FIND\_INFO\_RESPONSE event.

```
#include <att client.h>
```

## **Data Fields**

• bt byte status

Operation status.

bt\_uint start\_handle

First attribute handle found.

bt\_uint end\_handle

Last attribute handle found.

· bt uint count

The number of attribute handles found.

## 4.14.1 Detailed Description

Parameter to ATT CLIENT EVT FIND INFO RESPONSE event.

A pointer to this structure is passed to the ATT client application callback when client received a response (either positive or negative) to a "find info" request.

## 4.15 bt att client evt mtu response t Struct Reference

Parameter to ATT\_CLIENT\_EVT\_EXCHANGE\_MTU\_RESPONSE event.

```
#include <att_client.h>
```

## **Data Fields**

bt\_byte status

Operation status.

• bt uint mtu

Server's MTU.

## 4.15.1 Detailed Description

Parameter to ATT\_CLIENT\_EVT\_EXCHANGE\_MTU\_RESPONSE event.

A pointer to this structure is passed to the ATT client application callback when client received a response (either positive or negative) to a "exchange MTU" request.

## 4.16 bt att client evt prepare write response t Struct Reference

Parameter to ATT\_CLIENT\_EVT\_PREPARE\_WRITE\_RESPONSE event.

```
#include <att_client.h>
```

## **Data Fields**

bt\_byte status

Operation status.

· bt uint err handle

The attribute handle that generated error response.

## 4.16.1 Detailed Description

Parameter to ATT\_CLIENT\_EVT\_PREPARE\_WRITE\_RESPONSE event.

A pointer to this structure is passed to the ATT client application callback when client received a response (either positive or negative) to a "prepare write" request.

## 4.17 bt\_att\_client\_evt\_read\_blob\_response\_t Struct Reference

Parameter to ATT\_CLIENT\_EVT\_READ\_BLOB\_RESPONSE event.

```
#include <att_client.h>
```

## **Data Fields**

bt\_byte status

Operation status.

bt\_uint handle

Attribute handle.

bt\_uint offset

The offset of the first octet.

• const bt\_byte \* value

Part of the attribute value.

• bt uint len

Part of the attribute value length.

## 4.17.1 Detailed Description

Parameter to ATT\_CLIENT\_EVT\_READ\_BLOB\_RESPONSE event.

A pointer to this structure is passed to the ATT client application callback when client received a response (either positive or negative) to a "read blob" request.

## 4.18 bt\_att\_client\_evt\_read\_by\_group\_type\_response\_t Struct Reference

Parameter to ATT CLIENT EVT READ BY GROUP TYPE RESPONSE event.

```
#include <att_client.h>
```

## **Data Fields**

· bt byte status

Operation status.

• bt\_uint start\_handle

First attribute handle found.

· bt\_uint end\_handle

Last attribute handle found.

bt\_uint count

The number of attribute handles found.

## 4.18.1 Detailed Description

Parameter to ATT CLIENT EVT READ BY GROUP TYPE RESPONSE event.

A pointer to this structure is passed to the ATT client application callback when client received a response (either positive or negative) to a "read by group type" request.

## 4.19 bt att client evt read by type response t Struct Reference

Parameter to ATT\_CLIENT\_EVT\_READ\_BY\_TYPE\_RESPONSE event.

```
#include <att_client.h>
```

## **Data Fields**

• bt\_byte status

Operation status.

bt\_uint start\_handle

First attribute handle found.

bt\_uint end\_handle

Last attribute handle found.

· bt uint count

The number of attribute handles found.

## 4.19.1 Detailed Description

Parameter to ATT\_CLIENT\_EVT\_READ\_BY\_TYPE\_RESPONSE event.

A pointer to this structure is passed to the ATT client application callback when client received a response (either positive or negative) to a "read by type" request.

# 4.20 bt\_att\_client\_evt\_read\_multiple\_response\_t Struct Reference

Parameter to ATT\_CLIENT\_EVT\_READ\_MULTIPLE\_RESPONSE event.

```
#include <att_client.h>
```

#### **Data Fields**

• bt\_byte status

Operation status.

• const bt\_byte \* values

Attribute values.

bt\_uint len

The length of attribute values.

### 4.20.1 Detailed Description

Parameter to ATT\_CLIENT\_EVT\_READ\_MULTIPLE\_RESPONSE event.

A pointer to this structure is passed to the ATT client application callback when client received a response (either positive or negative) to a "read multiple" request.

# 4.21 bt\_att\_client\_evt\_read\_response\_t Struct Reference

Parameter to ATT\_CLIENT\_EVT\_READ\_RESPONSE event.

```
#include <att_client.h>
```

### **Data Fields**

• bt\_byte status

Operation status.

· bt\_uint handle

Attribute handle.

• const bt\_byte \* value

Attribute value.

bt\_uint len

Attribute value length.

### 4.21.1 Detailed Description

Parameter to ATT\_CLIENT\_EVT\_READ\_RESPONSE event.

A pointer to this structure is passed to the ATT client application callback when client received a response (either positive or negative) to a "read" request.

## 4.22 bt att client evt t Union Reference

Parameter to ATT client application callback.

#include <att\_client.h>

#### **Data Fields**

• bt\_att\_client\_evt\_mtu\_response\_t mtu

Valid if event is ATT\_CLIENT\_EVT\_EXCHANGE\_MTU\_RESPONSE.

bt\_att\_client\_evt\_info\_response\_t find\_info

Valid if event is ATT\_CLIENT\_EVT\_FIND\_INFO\_RESPONSE.

bt\_att\_client\_evt\_find\_by\_type\_value\_response\_t find\_by\_type\_value

Valid if event is ATT\_CLIENT\_EVT\_FIND\_BY\_TYPE\_VALUE\_RESPONSE.

bt\_att\_client\_evt\_read\_by\_type\_response\_t read\_by\_type

Valid if event is ATT\_CLIENT\_EVT\_READ\_BY\_TYPE\_RESPONSE.

bt\_att\_client\_evt\_read\_response\_t read

Valid if event is ATT\_CLIENT\_EVT\_READ\_RESPONSE.

bt\_att\_client\_evt\_read\_blob\_response\_t read\_blob

Valid if event is ATT\_CLIENT\_EVT\_READ\_BLOB\_RESPONSE.

bt\_att\_client\_evt\_read\_multiple\_response\_t read\_multiple

Valid if event is ATT\_CLIENT\_EVT\_READ\_MULTIPLE\_RESPONSE.

bt\_att\_client\_evt\_read\_by\_group\_type\_response\_t read\_by\_group\_type

Valid if event is ATT\_CLIENT\_EVT\_READ\_BY\_GROUP\_TYPE\_RESPONSE.

bt\_att\_client\_evt\_write\_response\_t write

Valid if event is ATT\_CLIENT\_EVT\_WRITE\_RESPONSE.

• bt\_att\_client\_evt\_prepare\_write\_response\_t prepare\_write

Valid if event is ATT CLIENT EVT PREPARE WRITE RESPONSE.

bt\_att\_client\_evt\_execute\_write\_response\_t execute\_write

Valid if event is ATT\_CLIENT\_EVT\_EXECUTE\_WRITE\_RESPONSE.

bt\_att\_client\_evt\_value\_notification\_t value\_notification

Valid if event is ATT\_CLIENT\_EVT\_VALUE\_NOTIFICATION.

bt\_att\_client\_evt\_value\_indication\_t value\_indication

Valid if event is ATT\_CLIENT\_EVT\_VALUE\_INDICATION.

bt\_att\_client\_evt\_conn\_param\_update\_t conn\_param\_update

Valid if event is ATT\_CLIENT\_EVT\_CONN\_PARAM\_UPDATE\_REQUEST.

bt\_att\_client\_evt\_conn\_param\_update\_completed\_t conn\_param\_update\_completed

Valid if event is ATT\_CLIENT\_EVT\_CONN\_PARAM\_UPDATE\_COMPLETED.

### 4.22.1 Detailed Description

Parameter to ATT client application callback.

This union is used to pass event specific data to the ATT client consumer. Which member of the union points to a valid structure depends on the event reported to the consumer. In general, each event has a corresponding member in the union.

## 4.23 bt att client evt value indication t Struct Reference

Parameter to ATT\_CLIENT\_EVT\_VALUE\_INDICATION event.

#include <att\_client.h>

bt\_uint handle

Attribute handle.

• const bt\_byte \* value

Attribute value.

• bt uint len

Attribute value length.

## 4.23.1 Detailed Description

Parameter to ATT CLIENT EVT VALUE INDICATION event.

A pointer to this structure is passed to the ATT client application callback when client received attribute value indication.

# 4.24 bt\_att\_client\_evt\_value\_notification\_t Struct Reference

Parameter to ATT\_CLIENT\_EVT\_VALUE\_NOTIFICATION event.

```
#include <att_client.h>
```

## **Data Fields**

· bt uint handle

Attribute handle.

• const bt\_byte \* value

Attribute value.

bt\_uint len

Attribute value length.

### 4.24.1 Detailed Description

Parameter to ATT\_CLIENT\_EVT\_VALUE\_NOTIFICATION event.

A pointer to this structure is passed to the ATT client application callback when client received attribute value notification.

# 4.25 bt\_att\_client\_evt\_write\_response\_t Struct Reference

Parameter to ATT\_CLIENT\_EVT\_WRITE\_RESPONSE event.

```
#include <att_client.h>
```

### **Data Fields**

· bt byte status

Operation status.

• bt\_uint handle

The attribute handle.

## 4.25.1 Detailed Description

Parameter to ATT\_CLIENT\_EVT\_WRITE\_RESPONSE event.

A pointer to this structure is passed to the ATT client application callback when client received a response (either positive or negative) to a "write" request.

- 4.26 bt\_att\_client\_mgr\_t Struct Reference
- 4.27 bt\_att\_client\_session\_t Struct Reference
- 4.28 bt\_att\_client\_uuid\_t Struct Reference

UUID.

#include <gatt\_client.h>

### **Data Fields**

• bt\_byte len

UUID length.

• bt\_uint uuid16

16-bit UUID.

• bt\_uuid\_t uuid128

128-bit UUID.

## 4.28.1 Detailed Description

UUID.

This structure is used to hold a UUID - 16 or 128 bit.

## 4.28.2 Field Documentation

4.28.2.1 bt\_byte len

UUID length.

This can be one of the following values:

- ATT\_CLIENT\_UUID\_LEN\_16
- ATT\_CLIENT\_UUID\_LEN\_128
- 4.29 bt\_att\_enum\_context\_t Struct Reference
- 4.30 bt att evt attr indication sent t Struct Reference

Parameter to ATT\_SERVER\_EVT\_ATTR\_VALUE\_INDICATED event.

#include <att.h>

· bt\_byte status

Error code.

• bt\_att\_session\_t \* session

ATT session.

· bt\_uint handle

Attribute handle.

## 4.30.1 Detailed Description

Parameter to ATT\_SERVER\_EVT\_ATTR\_VALUE\_INDICATED event.

A pointer to this structure is passed to the ATT application callback when a value indication has bee sent to the client.

## 4.31 bt\_att\_evt\_attr\_notification\_sent\_t Struct Reference

Parameter to ATT\_SERVER\_EVT\_ATTR\_VALUE\_NOTIFIED event.

```
#include <att.h>
```

#### **Data Fields**

• bt\_byte status

Error code.

• bt\_att\_session\_t \* session

ATT session.

• bt\_uint handle

Attribute handle.

## 4.31.1 Detailed Description

Parameter to ATT\_SERVER\_EVT\_ATTR\_VALUE\_NOTIFIED event.

A pointer to this structure is passed to the ATT application callback when a value notification has bee sent to the client.

# 4.32 bt\_att\_evt\_attr\_value\_changed\_t Struct Reference

Parameter to ATT\_SERVER\_EVT\_ATTR\_VALUE\_CHANGED\_BY\_SERVER and ATT\_SERVER\_EVT\_ATTR\_  $\hookleftarrow$  VALUE\_CHANGED\_BY\_CLIENT events.

```
#include <att.h>
```

### **Data Fields**

bt\_att\_attribute\_t \* attr

ATT attribute.

## 4.32.1 Detailed Description

Parameter to ATT\_SERVER\_EVT\_ATTR\_VALUE\_CHANGED\_BY\_SERVER and ATT\_SERVER\_EVT\_ATTR\_ ← VALUE CHANGED BY CLIENT events.

A pointer to this structure is passed to the ATT application callback when the value of an attribute has been changed locally on the server or by a client.

# 4.33 bt\_att\_evt\_attr\_value\_read\_t Struct Reference

Parameter to ATT\_SERVER\_EVT\_ATTR\_VALUE\_READ event.

```
#include <att.h>
```

### **Data Fields**

• bt\_att\_attribute\_t \* attr

ATT attribute.

bt\_uint value\_offset

Offset from which value of the attribute has been read.

## 4.33.1 Detailed Description

Parameter to ATT SERVER EVT ATTR VALUE READ event.

A pointer to this structure is passed to the ATT application callback when the value of an attribute has been read by a client.

# 4.34 bt\_att\_evt\_authorization\_requested\_t Struct Reference

Parameter to ATT\_SERVER\_EVT\_AUTHORIZATION\_REQUESTED event.

```
#include <att.h>
```

### **Data Fields**

• bt\_att\_session\_t \* session

ATT session.

• bt\_byte opcode

Operation code.

bt\_att\_attribute\_t \* attr

ATT attribute.

## 4.34.1 Detailed Description

Parameter to ATT\_SERVER\_EVT\_AUTHORIZATION\_REQUESTED event.

A pointer to this structure is passed to the ATT application callback if authorization is required in order to access the attribute's value.

# 4.35 bt\_att\_evt\_data\_received\_t Struct Reference

## 4.36 bt\_att\_evt\_session\_connected\_t Struct Reference

Parameter to ATT\_SERVER\_EVT\_SESSION\_CONNECTED event.

```
#include <att.h>
```

## **Data Fields**

 bt\_att\_session\_t \* session ATT session.

## 4.36.1 Detailed Description

Parameter to ATT\_SERVER\_EVT\_SESSION\_CONNECTED event.

A pointer to this structure is passed to the ATT application callback when a client connected to the server.

# 4.37 bt\_att\_evt\_session\_disconnected\_t Struct Reference

Parameter to ATT\_SERVER\_EVT\_SESSION\_DISCONNECTED event.

```
#include <att.h>
```

### **Data Fields**

 bt\_att\_session\_t \* session ATT session.

## 4.37.1 Detailed Description

Parameter to ATT\_SERVER\_EVT\_SESSION\_DISCONNECTED event.

A pointer to this structure is passed to the ATT application callback when a client disconnected from the server.

# 4.38 bt\_att\_evt\_tran\_timeout\_t Struct Reference

Parameter to ATT\_SERVER\_EVT\_TRAN\_TIMEOUT event.

```
#include <att.h>
```

## **Data Fields**

 bt\_att\_session\_t \* session ATT session.

## 4.38.1 Detailed Description

Parameter to ATT\_SERVER\_EVT\_TRAN\_TIMEOUT event.

A pointer to this structure is passed to the ATT application callback if operation has timed out.

# 4.39 bt\_att\_find\_by\_type\_value\_response\_t Struct Reference

Structure to store response to a "find by type value" request.

```
#include <att_client.h>
```

#### **Data Fields**

• bt\_uint handle

Found Attribute Handle.

· bt\_uint group\_end\_handle

Group End Handle.

### 4.39.1 Detailed Description

Structure to store response to a "find by type value" request.

An array of this structures is passed to bt\_att\_client\_find\_by\_type\_value(). It is application's responsibility to allocate memory for the array. When response is received the actual number of valid entries in the array is stored in bt\_catt\_client\_evt\_find\_by\_type\_value\_response\_t::count.

# 4.40 bt\_att\_find\_info\_response\_t Struct Reference

Structure to store response to a "find info" request.

```
#include <att_client.h>
```

## **Data Fields**

· bt\_byte format

Length of the attribute UUID.

bt\_uint handle

Attribute handle.

```
    union {
        bt_uint uuid16
            16-bit UUID
        bt_uuid_t uuid128
            128-bit UUID
    } type
```

Attribute UUID.

## 4.40.1 Detailed Description

Structure to store response to a "find info" request.

An array of this structures is passed to bt\_att\_client\_find\_info(). It is application's responsibility to allocate memory for the array. When response is received the actual number of valid entries in the array is stored in bt\_att\_client\_cevt\_info\_response\_t::count.

### 4.40.2 Field Documentation

4.40.2.1 bt\_byte format

Length of the attribute UUID.

- 0x01 = 16-bit UUID
- 0x02 = 128-bit UUID

# 4.41 bt\_att\_found\_attribyte\_t Struct Reference

# 4.42 bt\_att\_listener\_t Struct Reference

# 4.43 bt\_att\_mgr\_t Struct Reference

### ATT manager.

```
#include <att.h>
```

## 4.43.1 Detailed Description

## ATT manager.

A structure that glues all pieces together. There is only one instance of this structure allocated by dotstack. A pointer to the instance can be get with bt\_att\_get\_mgr().

# 4.44 bt\_att\_read\_by\_group\_type\_response\_t Struct Reference

Structure to store response to a "read by group type" request.

```
#include <att_client.h>
```

## **Data Fields**

· bt uint handle

Attribute handle.

• bt\_uint end\_group\_handle

Group End Handle.

· const bt byte \* value

Attribute value.

• bt\_byte len

Attribute value length.

# 4.44.1 Detailed Description

Structure to store response to a "read by group type" request.

An array of this structures is passed to bt\_att\_client\_read\_by\_group\_type(). It is application's responsibility to allocate memory for the array. When response is received the actual number of valid entries in the array is stored in bt\_att\_client\_evt\_read\_by\_group\_type\_response\_t::count.

# 4.45 bt\_att\_read\_by\_type\_response\_t Struct Reference

Structure to store response to a "read by type" request.

```
#include <att_client.h>
```

#### **Data Fields**

· bt uint handle

Attribute handle.

• const bt\_byte \* value

Attribute value.

• bt\_byte len

Attribute value length.

### 4.45.1 Detailed Description

Structure to store response to a "read by type" request.

An array of this structures is passed to bt\_att\_client\_read\_by\_type(). It is application's responsibility to allocate memory for the array. When response is received the actual number of valid entries in the array is stored in bt\_catt\_client\_evt\_read\_by\_type\_response\_t::count.

- 4.46 bt\_att\_session\_evt\_authorization\_requested\_t Struct Reference
- 4.47 bt\_att\_session\_evt\_packet\_sent\_t Struct Reference
- 4.48 bt\_att\_session\_t Struct Reference
- 4.49 bt\_av\_add\_to\_now\_playing\_t Struct Reference

Parameter to AVRCP\_EVT\_ADD\_TO\_NOW\_PLAYING\_COMPLETED event.

```
#include <avrcp_command.h>
```

### **Data Fields**

· bt av response theader

Common response header.

· bt\_byte status

The result of the request.

### 4.49.1 Detailed Description

Parameter to AVRCP\_EVT\_ADD\_TO\_NOW\_PLAYING\_COMPLETED event.

A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt\_avrcp\_event\_t union - bt\_avrcp\_event\_t::add\_to\_now\_playing\_status - when a local device received a response to a "add to now playing" request.

# 4.50 bt\_av\_battery\_status\_of\_ct\_t Struct Reference

Parameter to AVRCP\_EVT\_BATTERY\_STATUS\_OF\_CT\_RECEIVED event.

#include <avrcp\_command.h>

### **Data Fields**

· bt\_av\_response\_t header

Common response header.

• bt\_byte status

Battery status.

## 4.50.1 Detailed Description

Parameter to AVRCP\_EVT\_BATTERY\_STATUS\_OF\_CT\_RECEIVED event.

A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt\_avrcp\_event tunion - bt\_avrcp\_event\_t::battery\_status\_of\_ct - when a local device received a "battery status of controller" command.

#### 4.50.2 Field Documentation

4.50.2.1 bt\_byte status

Battery status.

This can be one of the following values:

- AVC\_BATTERY\_STATUS\_NORMAL
- AVC\_BATTERY\_STATUS\_WARNING
- AVC\_BATTERY\_STATUS\_CRITICAL
- AVC\_BATTERY\_STATUS\_EXTERNAL
- AVC\_BATTERY\_STATUS\_FULL\_CHARGE

# 4.51 bt\_av\_capability\_company\_id\_t Struct Reference

Parameter to AVRCP\_EVT\_COMPANY\_ID\_LIST\_RECEIVED event.

#include <avrcp\_command.h>

### **Data Fields**

• bt\_av\_response\_t header

Common response header.

bt\_byte count

The number of supported company ids.

• bt\_ulong \* company\_id\_list

List of supported company ids.

## 4.51.1 Detailed Description

Parameter to AVRCP\_EVT\_COMPANY\_ID\_LIST\_RECEIVED event.

A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt\_avrcp\_event\_t union - bt\_avrcp\_event\_t::company\_id - when a local device received a response to a "get company id" request.

# 4.52 bt\_av\_capability\_event\_id\_t Struct Reference

Parameter to AVRCP\_EVT\_EVENT\_ID\_LIST\_RECEIVED event.

```
#include <avrcp_command.h>
```

#### **Data Fields**

· bt\_av\_response\_t header

Common response header.

· bt byte count

The number of supported events ids.

bt\_byte \* event\_id\_list

List of supported event ids.

## 4.52.1 Detailed Description

Parameter to AVRCP\_EVT\_EVENT\_ID\_LIST\_RECEIVED event.

A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt\_avrcp\_event\_t union - bt\_avrcp\_event\_t::supported\_event\_id - when a local device received a response to a "get supported events" request.

# 4.53 bt\_av\_command\_t Struct Reference

# 4.54 bt\_av\_displayable\_character\_set\_t Struct Reference

Parameter to AVRCP EVT DISPLAYABLE CHARACTER SET RECEIVED event.

```
#include <avrcp_command.h>
```

### **Data Fields**

· bt av response theader

Common response header.

bt\_byte count

The number of supported characters sets by the controller.

• bt uint \* charset list

List of supported characters sets by the controller.

## 4.54.1 Detailed Description

Parameter to AVRCP\_EVT\_DISPLAYABLE\_CHARACTER\_SET\_RECEIVED event.

A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt\_avrcp\_event \_\_t union - bt\_avrcp\_event\_t::displayable\_character\_set - when a local device received a "displayable chracter set command" request.

# 4.55 bt\_av\_element\_attribute\_t Struct Reference

Media element attribute.

```
#include <avrcp_command.h>
```

### **Data Fields**

bt ulong id

Attribute Id.

· bt uint charset

Charcater set.

bt uint len

Value length.

• bt\_byte \* value

Attribute value.

### 4.55.1 Detailed Description

Media element attribute.

This structure is used to store media element attribute.

## 4.56 bt\_av\_element\_attributes\_t Struct Reference

Parameter to AVRCP\_EVT\_GET\_ELEMENT\_ATTRIBUTES\_RECEIVED event.

```
#include <avrcp_command.h>
```

### **Data Fields**

bt\_av\_response\_t header

Common response header.

• bt\_byte count

The number of attributes returned.

• bt\_av\_element\_attribute\_t \* attr\_list

List of attribute values.

### 4.56.1 Detailed Description

Parameter to AVRCP\_EVT\_GET\_ELEMENT\_ATTRIBUTES\_RECEIVED event.

A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt\_avrcp\_event\_t union - bt\_avrcp\_event\_t::element\_attributes - when a local device received a response to a "get media element attributes" request.

# 4.57 bt\_av\_element\_id\_t Struct Reference

### Media element UID.

#include <avrcp\_command.h>

#### **Data Fields**

• bt\_ulong id\_lo

4 least significant bytes of UID.

bt\_ulong id\_hi

4 most significant bytes of UID.

## 4.57.1 Detailed Description

Media element UID.

This structure is used to store media element UID.

# 4.58 bt\_av\_get\_element\_attributes\_t Struct Reference

Parameter to AVRCP\_EVT\_ELEMENT\_ATTRIBUTES\_REQUESTED event.

```
#include <avrcp_command.h>
```

### **Data Fields**

• bt\_av\_response\_t header

Common response header.

• bt\_av\_element\_id\_t id

Media element UID.

bt\_byte attributes

Bitmask that defines attributes requested.

## 4.58.1 Detailed Description

Parameter to AVRCP\_EVT\_ELEMENT\_ATTRIBUTES\_REQUESTED event.

A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt\_avrcp\_event\_t union - bt\_avrcp\_event\_t::get\_element\_attributes - when a local device received a "get element attributes" request.

## 4.58.2 Field Documentation

4.58.2.1 bt\_byte attributes

Bitmask that defines attributes requested.

This can be a combination of the following values:

- AVC\_MEDIA\_ATTR\_FLAG\_TITLE
- AVC\_MEDIA\_ATTR\_FLAG\_ARTIST
- AVC\_MEDIA\_ATTR\_FLAG\_ALBUM

- AVC\_MEDIA\_ATTR\_FLAG\_NUMBER
- AVC\_MEDIA\_ATTR\_FLAG\_TOTAL\_NUMBER
- AVC\_MEDIA\_ATTR\_FLAG\_GENRE
- AVC\_MEDIA\_ATTR\_FLAG\_PLAYING\_TIME

# 4.59 bt\_av\_notification\_addressed\_player\_changed\_t Struct Reference

Parameter to AVRCP\_EVT\_ADDRESSED\_PLAYER\_CHANGED event.

```
#include <avrcp_command.h>
```

#### **Data Fields**

· bt\_uint player\_id

New player Id.

· bt\_uint uid\_counter

UID counter.

## 4.59.1 Detailed Description

Parameter to AVRCP\_EVT\_ADDRESSED\_PLAYER\_CHANGED event.

A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt\_avrcp\_event\_t union - bt\_avrcp\_event\_t::notification::params:addressed\_player - when a local device received a "addressed player changed" notification.

# 4.60 bt\_av\_notification\_app\_setting\_changed\_t Struct Reference

# 4.61 bt\_av\_notification\_battery\_status\_t Struct Reference

Parameter to AVRCP EVT BATT STATUS CHANGED event.

```
#include <avrcp_command.h>
```

### **Data Fields**

bt\_byte status

New battery status. This can be one of the following values:

## 4.61.1 Detailed Description

Parameter to AVRCP\_EVT\_BATT\_STATUS\_CHANGED event.

A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt\_avrcp\_event—tunion - bt\_avrcp\_event\_t::notification::params:battery\_status - when a local device received a "battery status changed" notification.

### 4.61.2 Field Documentation

### 4.61.2.1 bt\_byte status

New battery status. This can be one of the following values:

- AVC BATTERY STATUS NORMAL
- · AVC BATTERY STATUS WARNING
- AVC\_BATTERY\_STATUS\_CRITICAL
- AVC\_BATTERY\_STATUS\_EXTERNAL
- · AVC BATTERY STATUS FULL CHARGE

## 4.62 bt\_av\_notification\_playback\_pos\_changed\_t Struct Reference

Parameter to AVRCP EVT PLAYBACK POS CHANGED event.

```
#include <avrcp_command.h>
```

### **Data Fields**

bt\_ulong position
 New playback position.

### 4.62.1 Detailed Description

Parameter to AVRCP\_EVT\_PLAYBACK\_POS\_CHANGED event.

A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt\_avrcp\_event—tunion - bt\_avrcp\_event\_t::notification::params:playback\_pos - when a local device received a "playback position changed" notification.

# 4.63 bt\_av\_notification\_playback\_status\_changed\_t Struct Reference

Parameter to AVRCP\_EVT\_PLAYBACK\_STATUS\_CHANGED event.

```
#include <avrcp_command.h>
```

### **Data Fields**

bt\_av\_response\_t header

Common response header.

• bt\_byte status

Play status.

## 4.63.1 Detailed Description

Parameter to AVRCP\_EVT\_PLAYBACK\_STATUS\_CHANGED event.

A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt\_avrcp\_event\_t union - bt\_avrcp\_event\_t::notification::params:play\_status - when a local device received a "play status changed" notification.

### 4.63.2 Field Documentation

4.63.2.1 bt\_byte status

Play status.

This can be on of the following values:

- AVC\_PLAY\_STATUS\_STOPPED
- AVC\_PLAY\_STATUS\_PLAYING
- AVC\_PLAY\_STATUS\_PAUSED
- AVC\_PLAY\_STATUS\_FW\_SEEK
- AVC\_PLAY\_STATUS\_REV\_SEEK
- AVC\_PLAY\_STATUS\_ERROR

## 4.64 bt\_av\_notification\_system\_status\_changed\_t Struct Reference

Parameter to AVRCP EVT SYSTEM STATUS CHANGED event.

#include <avrcp\_command.h>

### **Data Fields**

· bt byte status

New system status.

### 4.64.1 Detailed Description

Parameter to AVRCP\_EVT\_SYSTEM\_STATUS\_CHANGED event.

A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt\_avrcp\_event \_t union - bt\_avrcp\_event\_t::notification::params:system\_status - when a local device received a "system status changed" notification.

## 4.65 bt av notification t Struct Reference

Parameter to the following events:

#include <avrcp\_command.h>

## **Data Fields**

bt\_av\_response\_t header

Common response header.

• bt\_av\_notification\_playback\_status\_changed\_t play\_status

Valid if notification is AVRCP\_EVT\_PLAYBACK\_STATUS\_CHANGED.

bt\_av\_notification\_track\_changed\_t track

Valid if notification is AVRCP\_EVT\_TRACK\_CHANGED.

bt\_av\_notification\_playback\_pos\_changed\_t playback\_pos

Valid if notification is AVRCP\_EVT\_PLAYBACK\_POS\_CHANGED.

- · bt\_av\_notification\_battery\_status\_t battery\_status
  - Valid if notification is AVRCP\_EVT\_BATT\_STATUS\_CHANGED.
- bt\_av\_notification\_system\_status\_changed\_t system\_status
- Valid if notification is AVRCP\_EVT\_SYSTEM\_STATUS\_CHANGED.

   bt\_av\_notification\_addressed\_player\_changed\_t addressed\_player
  - Valid if notification is AVRCP\_EVT\_ADDRESSED\_PLAYER\_CHANGED.
- bt av\_notification\_uids\_changed\_t uids
  - Valid if notification is AVRCP\_EVT\_UIDS\_CHANGED.
- bt\_av\_notification\_volume\_changed\_t volume
  - Valid if notification is AVRCP\_EVT\_VOLUME\_CHANGED.
- bt av notification app setting changed tapp setting

Valid if notification is AVRCP\_EVT\_PLAYER\_APPLICATION\_SETTING\_CHANGED.

## 4.65.1 Detailed Description

Parameter to the following events:

- AVRCP\_EVT\_PLAYBACK\_STATUS\_CHANGED
- AVRCP\_EVT\_TRACK\_CHANGED
- AVRCP\_EVT\_PLAYBACK\_POS\_CHANGED
- AVRCP\_EVT\_BATT\_STATUS\_CHANGED
- · AVRCP EVT SYSTEM STATUS CHANGED
- AVRCP\_EVT\_NOW\_PLAYING\_CONTENT\_CHANGED
- AVRCP\_EVT\_AVAILABLE\_PLAYERS\_CHANGED
- AVRCP\_EVT\_ADDRESSED\_PLAYER\_CHANGED
- AVRCP\_EVT\_UIDS\_CHANGED
- AVRCP\_EVT\_VOLUME\_CHANGED

A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt\_avrcp\_event \_\_t union - bt\_avrcp\_event\_t::notification - when a local device received one of the following notifications from the target:

- · Play status changed
- · Track changed changed
- · Playback position changed
- Battery status changed
- System status changed
- · Addressed player changed
- · UIDs changed
- · Volume changed
- Player application setting changed The notification code defines which member of the bt\_av\_notification\_t 

  ∷params union is valid
- AVRCP\_EVT\_PLAYBACK\_STATUS\_CHANGED bt\_av\_notification\_playback\_status\_← changed\_t play\_status

- AVRCP\_EVT\_TRACK\_CHANGED bt\_av\_notification\_track\_changed\_t track
- AVRCP\_EVT\_PLAYBACK\_POS\_CHANGED bt\_av\_notification\_playback\_pos\_changed ← \_t playback\_pos
- AVRCP\_EVT\_BATT\_STATUS\_CHANGED bt\_av\_notification\_battery\_status\_t battery\_
   status
- AVRCP\_EVT\_SYSTEM\_STATUS\_CHANGED bt\_av\_notification\_system\_status\_changed ← \_t system\_status
- AVRCP\_EVT\_ADDRESSED\_PLAYER\_CHANGED bt\_av\_notification\_addressed\_player ← \_\_changed\_t addressed\_player
- AVRCP\_EVT\_UIDS\_CHANGED bt\_av\_notification\_uids\_changed\_t uids
- AVRCP\_EVT\_VOLUME\_CHANGED bt\_av\_notification\_volume\_changed\_t volume
- AVRCP\_EVT\_PLAYER\_APPLICATION\_SETTING\_CHANGED bt\_av\_notification\_app\_ 
  setting\_changed\_t app\_setting

## 4.66 bt av notification track changed t Struct Reference

Parameter to AVRCP\_EVT\_TRACK\_CHANGED event.

#include <avrcp\_command.h>

#### **Data Fields**

 bt\_av\_element\_id\_t id New track UID.

## 4.66.1 Detailed Description

Parameter to AVRCP EVT TRACK CHANGED event.

A pointer to this structure is passed to the AVRCP application callback as a valid member of the <a href="bt\_avrcp\_event\_t">bt\_avrcp\_event\_t</a>::notification::params:track - when a local device received a "track changed" notification.

## 4.67 bt\_av\_notification\_uids\_changed\_t Struct Reference

Parameter to AVRCP EVT UIDS CHANGED event.

#include <avrcp\_command.h>

## **Data Fields**

bt\_uint uid\_counter
 UID counter.

### 4.67.1 Detailed Description

Parameter to AVRCP\_EVT\_UIDS\_CHANGED event.

A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt\_avrcp\_event\_t union - bt\_avrcp\_event\_t::notification::params:uids - when a local device received a "UIDs changed" notification.

# 4.68 bt\_av\_notification\_volume\_changed\_t Struct Reference

Parameter to AVRCP\_EVT\_VOLUME\_CHANGED event.

#include <avrcp\_command.h>

### **Data Fields**

• bt byte volume

Volume.

## 4.68.1 Detailed Description

Parameter to AVRCP\_EVT\_VOLUME\_CHANGED event.

A pointer to this structure is passed to the AVRCP application callback as a valid member of the <a href="bt\_avrcp\_event\_t">bt\_avrcp\_event\_t</a>::notification::params:volume - when a local device received a "UIDs changed" notification.

# 4.69 bt\_av\_play\_item\_t Struct Reference

Parameter to AVRCP\_EVT\_PLAY\_ITEM\_COMPLETED event.

#include <avrcp\_command.h>

### **Data Fields**

· bt\_av\_response\_t header

Common response header.

bt\_byte status

The result of the request.

## 4.69.1 Detailed Description

Parameter to AVRCP\_EVT\_PLAY\_ITEM\_COMPLETED event.

A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt\_avrcp\_event\_t union - bt\_avrcp\_event\_t::play\_item\_status - when a local device received a response to a "play item" request.

# 4.70 bt\_av\_play\_status\_t Struct Reference

Parameter to AVRCP\_EVT\_GET\_PLAY\_STATUS\_RECEIVED event.

#include <avrcp\_command.h>

### **Data Fields**

bt\_av\_response\_t header

Common response header.

• bt\_ulong song\_length

Current track length.

bt\_ulong song\_position

Current track position.

bt\_byte play\_status

Playback status.

### 4.70.1 Detailed Description

Parameter to AVRCP\_EVT\_GET\_PLAY\_STATUS\_RECEIVED event.

A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt\_avrcp\_event\_t union - bt\_avrcp\_event\_t::play\_status - when a local device received a response to a "get play status" request.

### 4.70.2 Field Documentation

4.70.2.1 bt\_byte play\_status

Playback status.

This can be one of the following values:

- AVC\_PLAY\_STATUS\_STOPPED
- AVC\_PLAY\_STATUS\_PLAYING
- AVC\_PLAY\_STATUS\_PAUSED
- AVC\_PLAY\_STATUS\_FW\_SEEK
- AVC\_PLAY\_STATUS\_REV\_SEEK
- AVC\_PLAY\_STATUS\_ERROR

## 4.71 bt av player setting current values t Struct Reference

Parameter to AVRCP\_EVT\_PLAYER\_CURRENT\_SETTING\_VALUES\_RECEIVED event.

#include <avrcp\_command.h>

## **Data Fields**

bt\_av\_response\_t header

Common response header.

· bt byte count

The number of player setting attribute ids to be returned from the target.

• bt byte \* setting id list

List of player setting attribute ids to be returned from the target.

bt\_byte \* setting\_value\_id\_list

List of current player setting attribute value ids.

### 4.71.1 Detailed Description

Parameter to AVRCP\_EVT\_PLAYER\_CURRENT\_SETTING\_VALUES\_RECEIVED event.

A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt\_avrcp\_event\_t union - bt\_avrcp\_event\_t::player\_setting\_current\_values - when a local device received a response to a "get current player setting attribute values" request.

# 4.72 bt\_av\_player\_setting\_values\_t Struct Reference

Parameter to AVRCP EVT PLAYER SETTING VALUES RECEIVED event.

#include <avrcp\_command.h>

### **Data Fields**

• bt\_av\_response\_t header

Common response header.

bt byte count

The number of supported player setting attribute value ids.

bt\_byte \* setting\_value\_id\_list

List of supported player setting attribute value ids.

### 4.72.1 Detailed Description

Parameter to AVRCP EVT PLAYER SETTING VALUES RECEIVED event.

A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt\_avrcp\_event\_t union - bt\_avrcp\_event\_t::player\_setting\_values - when a local device received a response to a "get player setting attribute values" request.

# 4.73 bt\_av\_player\_setting\_values\_text\_t Struct Reference

Parameter to AVRCP\_EVT\_PLAYER\_SETTING\_VALUES\_TEXT\_RECEIVED event.

#include <avrcp\_command.h>

## **Data Fields**

• bt\_av\_response\_t header

Common response header.

bt\_byte count

The number of player setting attribute value ids for which displayable text is requested.

bt\_byte \* setting\_value\_id\_list

List of player setting attribute value ids for which displayable text is requested.

bt\_av\_player\_text\_t \* setting\_value\_text\_list

List of player setting attribute values displayable text.

### 4.73.1 Detailed Description

Parameter to AVRCP EVT PLAYER SETTING VALUES TEXT RECEIVED event.

A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt\_avrcp\_event\_t union - bt\_avrcp\_event\_t::player\_setting\_values\_text - when a local device received a response to a "get player setting attribute values displayable text" request.

# 4.74 bt\_av\_player\_settings\_t Struct Reference

Parameter to AVRCP\_EVT\_PLAYER\_SETTING\_ATTRIBUTES\_RECEIVED event.

#include <avrcp\_command.h>

bt\_av\_response\_t header

Common response header.

· bt byte count

The number of supported player setting attribute ids.

bt byte \* setting id list

List of supported player setting attribute ids.

### 4.74.1 Detailed Description

Parameter to AVRCP\_EVT\_PLAYER\_SETTING\_ATTRIBUTES\_RECEIVED event.

A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt\_avrcp\_event \_t union - bt\_avrcp\_event\_t::player\_settings - when a local device received a response to a "get supported player setting attributes" request.

## 4.75 bt\_av\_player\_settings\_text\_t Struct Reference

Parameter to AVRCP\_EVT\_PLAYER\_SETTING\_ATTRIBUTES\_TEXT\_RECEIVED event.

#include <avrcp\_command.h>

### **Data Fields**

bt\_av\_response\_t header

Common response header.

bt\_byte count

The number of player setting attribute ids for which displayable text is requested.

bt byte \* setting id list

List of player setting attribute ids for which displayable text is requested.

bt\_av\_player\_text\_t \* setting\_text\_list

List of player setting attributes displayable text.

### 4.75.1 Detailed Description

Parameter to AVRCP\_EVT\_PLAYER\_SETTING\_ATTRIBUTES\_TEXT\_RECEIVED event.

A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt\_avrcp\_event\_t union - bt\_avrcp\_event\_t::player\_settings\_text - when a local device received a response to a "get player setting attributes displayable text" request.

# 4.76 bt\_av\_player\_text\_t Struct Reference

# 4.77 bt\_av\_register\_notification\_t Struct Reference

Parameter to AVRCP\_EVT\_REGISTER\_NOTIFICATION\_REQUESTED event.

#include <avrcp\_command.h>

bt\_av\_response\_t header

Common response header.

bt\_byte event\_id

Event Id.

bt\_ulong playback\_pos

Playback position. Used only if event\_id is AVC\_EVENT\_PLAYBACK\_POS\_CHANGED.

## 4.77.1 Detailed Description

Parameter to AVRCP\_EVT\_REGISTER\_NOTIFICATION\_REQUESTED event.

A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt\_avrcp\_event\_t union - bt\_avrcp\_event\_t::register\_notification - when a local device received a "register notification" request.

### 4.77.2 Field Documentation

4.77.2.1 bt\_byte event\_id

Event Id.

This can be one of the following values:

- AVC\_EVENT\_PLAYBACK\_STATUS\_CHANGED
- AVC\_EVENT\_TRACK\_CHANGED
- AVC\_EVENT\_TRACK\_REACHED\_END
- AVC\_EVENT\_TRACK\_REACHED\_START
- AVC\_EVENT\_PLAYBACK\_POS\_CHANGED
- AVC\_EVENT\_BATT\_STATUS\_CHANGED
- AVC\_EVENT\_SYSTEM\_STATUS\_CHANGED
- AVC\_EVENT\_PLAYER\_APPLICATION\_SETTING\_CHANGED
- AVC\_EVENT\_NOW\_PLAYING\_CONTENT\_CHANGED
- AVC\_EVENT\_AVAILABLE\_PLAYERS\_CHANGED
- AVC\_EVENT\_ADDRESSED\_PLAYER\_CHANGED
- AVC\_EVENT\_UIDS\_CHANGED
- AVC\_EVENT\_VOLUME\_CHANGED

# 4.78 bt\_av\_response\_t Struct Reference

AV/C response header.

#include <avrcp\_command.h>

• bt\_avrcp\_channel\_t \* channel

AVRCP channel.

• bt\_byte ctype

Response type.

· bt\_byte tran\_id

Transaction Id.

## 4.78.1 Detailed Description

AV/C response header.

This structure is used to store fields present in every AV/C response.

### 4.78.2 Field Documentation

4.78.2.1 bt\_byte ctype

Response type.

This can be one of the following values:

- AVC\_RESPONSE\_NOT\_IMPLEMENTED
- AVC\_RESPONSE\_ACCEPTED
- AVC\_RESPONSE\_REJECTED
- AVC\_RESPONSE\_IN\_TRANSITION
- AVC\_RESPONSE\_STABLE
- AVC\_RESPONSE\_IMPLEMENTED
- AVC\_RESPONSE\_CHANGED
- AVC\_RESPONSE\_INTERIM
- AVC\_RESPONSE\_TIMEOUT

# 4.79 bt\_av\_set\_absolute\_volume\_t Struct Reference

Parameter to AVRCP\_EVT\_SET\_ABSOLUTE\_VOLUME\_COMPLETED event.

```
#include <avrcp_command.h>
```

### **Data Fields**

• bt\_av\_response\_t header

Common response header.

• bt\_byte volume

Volume.

## 4.79.1 Detailed Description

Parameter to AVRCP\_EVT\_SET\_ABSOLUTE\_VOLUME\_COMPLETED event.

A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt\_avrcp\_event\_t union - bt\_avrcp\_event\_t::absolute\_volume - when a local device received a response to a "set absolute volume" request.

# 4.80 bt\_av\_set\_addressed\_player\_t Struct Reference

Parameter to AVRCP\_EVT\_SET\_ADDRESSED\_PLAYER\_COMPLETED event.

```
#include <avrcp_command.h>
```

#### **Data Fields**

· bt\_av\_response\_t header

Common response header.

bt\_byte status

The result of changing the addressed player.

## 4.80.1 Detailed Description

Parameter to AVRCP\_EVT\_SET\_ADDRESSED\_PLAYER\_COMPLETED event.

A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt\_avrcp\_event\_t union - bt\_avrcp\_event\_t::addressed\_player - when a local device received a response to a "set addressed player" request.

# 4.81 bt\_avctp\_channel\_t Struct Reference

AVCTP channel description.

```
#include <avctp.h>
```

### 4.81.1 Detailed Description

AVCTP channel description.

This structure is used to hold information about an AVCTP channel.

## 4.82 bt\_avctp\_event\_t Union Reference

Parameter to an application callback.

```
#include <avctp.h>
```

## **Data Fields**

- bt\_avctp\_evt\_channel\_connected\_t channel\_connected
   Valid if event is AVCTP\_EVT\_CHANNEL\_CONNECTED.
- bt\_avctp\_evt\_channel\_disconnected\_t channel\_disconnected

Valid if event is AVCTP\_EVT\_CHANNEL\_DISCONNECTED.

bt\_avctp\_evt\_connection\_failed\_t connection\_failed

Valid if event is AVCTP\_EVT\_CONNECTION\_FAILED.

• bt\_avctp\_evt\_command\_received\_t command\_received

Valid if event is AVCTP\_EVT\_COMMAND\_RECEIVED.

• bt\_avctp\_evt\_response\_received\_t response\_received

Valid if event is AVCTP\_EVT\_RESPONE\_RECEIVED.

bt\_avctp\_evt\_command\_sent\_t command\_sent

Valid if event is AVCTP EVT COMMAND SENT.

bt\_avctp\_evt\_response\_sent\_t response\_sent
 Valid if event is AVCTP\_EVT\_RESPONSE\_SENT.

• bt\_avctp\_evt\_command\_cancelled\_t command\_cancelled

Valid if event is AVCTP\_EVT\_COMMAND\_CANCELLED.

• bt\_avctp\_evt\_response\_cancelled\_t response\_cancelled

Valid if event is AVCTP\_EVT\_RESPONSE\_CANCELLED.

## 4.82.1 Detailed Description

Parameter to an application callback.

This union is used to pass event specific data to the AVCTP consumer. Which member of the union points to a valid structure depends on the event reported to the consumer. In general, each event has a corresponding member in the union.

## 4.83 bt\_avctp\_evt\_channel\_connected\_t Struct Reference

Parameter to AVCTP\_EVT\_CHANNEL\_CONNECTED event.

```
#include <avctp.h>
```

### **Data Fields**

 bt\_avctp\_channel\_t \* channel AVCTP channel.

### 4.83.1 Detailed Description

Parameter to AVCTP\_EVT\_CHANNEL\_CONNECTED event.

A pointer to this structure is passed to the AVCTP application callback as a valid member of the bt\_avctp\_event\_t union - bt\_avctp\_event\_t::channel\_connected - when a channel between two devices has been established.

## 4.84 bt\_avctp\_evt\_channel\_disconnected\_t Struct Reference

 $Parameter\ to\ AVCTP\_EVT\_CHANNEL\_DISCONNECTED\ event.$ 

```
#include <avctp.h>
```

### **Data Fields**

 bt\_avctp\_channel\_t \* channel AVCTP channel.

## 4.84.1 Detailed Description

Parameter to AVCTP\_EVT\_CHANNEL\_DISCONNECTED event.

A pointer to this structure is passed to the AVCTP application callback as a valid member of the bt\_avctp\_event\_t union - bt\_avctp\_event\_t::channel\_disconnected - when a channel between two devices has been terminated.

## 4.85 bt\_avctp\_evt\_command\_cancelled\_t Struct Reference

Parameter to AVCTP\_EVT\_COMMAND\_CANCELLED event.

```
#include <avctp.h>
```

### **Data Fields**

bt\_avctp\_channel\_t \* channel

AVCTP channel.

bt\_avctp\_message\_t \* command

AVCTP command message.

### 4.85.1 Detailed Description

Parameter to AVCTP\_EVT\_COMMAND\_CANCELLED event.

A pointer to this structure is passed to the AVCTP application callback as a valid member of the bt\_avctp\_event\_t union - bt\_avctp\_event\_t::command\_cancelled - when sending a command message has been canceled.

# 4.86 bt\_avctp\_evt\_command\_received\_t Struct Reference

Parameter to AVCTP\_EVT\_COMMAND\_RECEIVED event.

```
#include <avctp.h>
```

#### **Data Fields**

bt\_avctp\_channel\_t \* channel

AVCTP channel.

bt\_avctp\_message\_t \* command

AVCTP command message.

### 4.86.1 Detailed Description

Parameter to AVCTP EVT COMMAND RECEIVED event.

A pointer to this structure is passed to the AVCTP application callback as a valid member of the bt\_avctp\_event\_t union - bt\_avctp\_event\_t::command\_received - when a local device received a command message.

## 4.87 bt avctp evt command sent t Struct Reference

Parameter to AVCTP\_EVT\_COMMAND\_SENT event.

```
#include <avctp.h>
```

```
    bt_avctp_channel_t * channel
    AVCTP channel.
```

bt\_avctp\_message\_t \* command

AVCTP command message.

## 4.87.1 Detailed Description

Parameter to AVCTP\_EVT\_COMMAND\_SENT event.

A pointer to this structure is passed to the AVCTP application callback as a valid member of the bt\_avctp\_event\_t union - bt\_avctp\_event\_t::command\_sent - when a local device finished sending a command message.

## 4.88 bt\_avctp\_evt\_connection\_failed\_t Struct Reference

Parameter to AVCTP\_EVT\_CONNECTION\_FAILED event.

```
#include <avctp.h>
```

### **Data Fields**

 bt\_avctp\_channel\_t \* channel AVCTP channel.

## 4.88.1 Detailed Description

Parameter to AVCTP EVT CONNECTION FAILED event.

A pointer to this structure is passed to the AVCTP application callback as a valid member of the <a href="mailto:bt\_avctp\_event\_t::connection\_failed">bt\_avctp\_event\_t::connection\_failed</a> - when a channel between two devices could not be established.

## 4.89 bt\_avctp\_evt\_response\_cancelled\_t Struct Reference

Parameter to AVCTP\_EVT\_RESPONSE\_CANCELLED event.

```
#include <avctp.h>
```

### **Data Fields**

bt\_avctp\_channel\_t \* channel

AVCTP channel.

• bt avctp message t \* response

AVCTP response message.

## 4.89.1 Detailed Description

Parameter to AVCTP\_EVT\_RESPONSE\_CANCELLED event.

A pointer to this structure is passed to the AVCTP application callback as a valid member of the bt\_avctp\_event\_t union - bt\_avctp\_event\_t::response\_cancelled - when sending a response message has been canceled.

# 4.90 bt\_avctp\_evt\_response\_received\_t Struct Reference

Parameter to AVCTP\_EVT\_RESPONSE\_RECEIVED event.

```
#include <avctp.h>
```

### **Data Fields**

- bt\_avctp\_channel\_t \* channel
  - AVCTP channel.
- bt\_avctp\_message\_t \* response

AVCTP response message.

### 4.90.1 Detailed Description

Parameter to AVCTP\_EVT\_RESPONSE\_RECEIVED event.

A pointer to this structure is passed to the AVCTP application callback as a valid member of the bt\_avctp\_event\_t union - bt\_avctp\_event\_t::response\_received - when a local device received a response message.

# 4.91 bt\_avctp\_evt\_response\_sent\_t Struct Reference

Parameter to AVCTP\_EVT\_RESPONSE\_SENT event.

```
#include <avctp.h>
```

### **Data Fields**

- bt\_avctp\_channel\_t \* channel
  - AVCTP channel.
- bt\_avctp\_message\_t \* response

AVCTP response message.

## 4.91.1 Detailed Description

Parameter to AVCTP\_EVT\_RESPONSE\_SENT event.

A pointer to this structure is passed to the AVCTP application callback as a valid member of the bt\_avctp\_event\_t union - bt\_avctp\_event\_t::response\_sent - when a local device finished sending a response message.

## 4.92 bt avctp message t Struct Reference

AVCTP message description.

```
#include <avctp.h>
```

### 4.92.1 Detailed Description

AVCTP message description.

This structure is used to hold information about an AVCTP message.

# 4.93 bt\_avctp\_mgr\_t Struct Reference

## AVCTP manager.

#include <avctp.h>

### **Data Fields**

• bt\_byte state

Manager state.

bt\_byte flags

Additional manager state.

bt\_avctp\_channel\_t \* channels

List of available AVCTP channels.

• bt\_avctp\_transport\_t \* transports

List of available AVCTP transports.

bt\_avctp\_mgr\_callback\_fp callback

Pointer to a function used to notify the AVCTP consumer about various events.

void \* callback\_param

Pointer to arbitrary data to be passed to the callback.

• bt\_avctp\_channel\_t \* opening\_channel

Pointer to a channle being open.

## 4.93.1 Detailed Description

## AVCTP manager.

A structure that glues all pieces together. There is only one instance of this structure allocated by dotstack. A pointer to the instance can be get with bt\_avctp\_get\_mgr().

### 4.93.2 Field Documentation

4.93.2.1 bt\_byte flags

Additional manager state.

This value can be a combination of the following values:

AVDTP\_MANAGER\_FLAG\_SENDING\_MEDIA\_PACKET

4.93.2.2 bt\_byte state

Manager state.

This value can be one of the following values:

- · AVCTP MANAGER STATE IDLE
- AVCTP\_MANAGER\_STATE\_CONNECTING

## 4.94 bt\_avctp\_transport\_t Struct Reference

AVCTP transport description.

#include <avctp.h>

## 4.94.1 Detailed Description

AVCTP transport description.

This structure is used to hold information about an AVCTP transport.

- 4.95 bt\_avdtp\_codec\_op\_decode\_t Struct Reference
- 4.96 bt avdtp codec op encode t Struct Reference
- 4.97 bt\_avdtp\_codec\_op\_param\_t Union Reference

Parameter to a codec handler.

```
#include <avdtp.h>
```

#### **Data Fields**

- bt\_avdtp\_codec\_op\_parse\_config\_t parse
   Valid if operation is AVDTP\_CODEC\_OPCODE\_PARSE\_CONFIG.
- bt\_avdtp\_codec\_op\_serialize\_config\_t serialize

Valid if operation is AVDTP\_CODEC\_OPCODE\_SERIALIZE\_CONFIG.

• bt\_avdtp\_codec\_op\_parse\_packet\_t parse\_packet

This member is currently not used.

### 4.97.1 Detailed Description

Parameter to a codec handler.

This union is used to pass operation specific data to a codec handler. Which member of the union points to a valid structure depends on the operation.

# 4.98 bt\_avdtp\_codec\_op\_parse\_config\_t Struct Reference

Parameter to AVDTP\_CODEC\_OPCODE\_PARSE\_CONFIG operation.

```
#include <avdtp.h>
```

### **Data Fields**

void \* codec\_config

A pointer to a structure defined by the AVDTP consumer where codec's configuration will be stored by the handler.

• bt\_byte codec\_config\_max\_size

The maximum size of a buffer pointed to by codec\_config field.

• bt\_byte \* buffer

A pointer to a buffer holding codec's configuration in OTA format.

bt\_int buffer\_len

The length of the buffer.

bt\_int offset

The buffer points to a complete packet received from the remoted device.

## 4.98.1 Detailed Description

Parameter to AVDTP\_CODEC\_OPCODE\_PARSE\_CONFIG operation.

A pointer to this structure is passed to the codec handler as a valid member of the bt\_avdtp\_codec\_op\_param \_\_t union - bt\_avdtp\_codec\_op\_param\_t::parse - when ADVDTP needs to parse codec's capabilities/configuration received from the remote device.

#### 4.98.2 Field Documentation

4.98.2.1 void\* codec\_config

A pointer to a structure defined by the AVDTP consumer where codec's configuration will be stored by the handler.

The format of the structure is totally up to the AVDTP consumer. The dotstack defines such structures for SBC, MPEG-1,2 and MPEG-2,4 AAC:

- SBC: bt a2dp sbc config t (defined in a2dp sbc codec.h)
- MPEG-1,2: bt\_a2dp\_mpeg\_config\_t (defined in a2dp\_mpeg\_codec.h)
- MPEG-2,4 AAC: bt\_a2dp\_aac\_config\_t (defined in a2dp\_aac\_codec.h)

### 4.98.2.2 bt\_int offset

The buffer points to a complete packet received from the remoted device.

The offset points to a location in the buffer where codec's configuration starts.

## 4.99 bt\_avdtp\_codec\_op\_parse\_packet\_t Struct Reference

## 4.100 bt\_avdtp\_codec\_op\_serialize\_config\_t Struct Reference

Parameter to AVDTP\_CODEC\_OPCODE\_SERIALIZE\_CONFIG operation.

```
#include <avdtp.h>
```

### **Data Fields**

void \* codec\_config

A pointer to a structure defined by the AVDTP consumer where codec's configuration will be read from by the handler.

• bt byte \* buffer

A pointer to a buffer where the handler has to write codec's configuration in OTA format.

bt\_int buffer\_len

The length of the buffer.

bt\_int offset

The buffer points to a complete packet that will be sent to the remote device.

### 4.100.1 Detailed Description

Parameter to AVDTP\_CODEC\_OPCODE\_SERIALIZE\_CONFIG operation.

A pointer to this structure is passed to the codec handler as a valid member of the bt\_avdtp\_codec\_op\_param\_t union - bt\_avdtp\_codec\_op\_param\_t::serialize - when ADVDTP needs to serialize codec's capabilities/configuration for sending to the remote device.

### 4.100.2 Field Documentation

4.100.2.1 void\* codec\_config

A pointer to a structure defined by the AVDTP consumer where codec's configuration will be read from by the handler.

The format of the structure is totally up to the AVDTP consumer. The dotstack defines such structures for SBC, MPEG-1,2 and MPEG-2,4 AAC:

- SBC: bt\_a2dp\_sbc\_config\_t (defined in a2dp\_sbc\_codec.h)
- MPEG-1,2: bt\_a2dp\_mpeg\_config\_t (defined in a2dp\_mpeg\_codec.h)
- MPEG-2,4 AAC: bt\_a2dp\_aac\_config\_t (defined in a2dp\_aac\_codec.h)

### 4.100.2.2 bt\_int offset

The buffer points to a complete packet that will be sent to the remote device.

The offset points to a location in the buffer where coded's configuration has to be written.

# 4.101 bt\_avdtp\_codec\_t Struct Reference

### Codec handler description.

```
#include <avdtp.h>
```

### **Data Fields**

• bt byte codec type

Codec type.

• bt\_avdtp\_codec\_handler\_fp codec\_handler

A pointer to a codec handler.

# 4.101.1 Detailed Description

Codec handler description.

### 4.101.2 Field Documentation

4.101.2.1 bt\_byte codec\_type

## Codec type.

The codec\_type can be one of the following values:

- AVDTP\_CODEC\_TYPE\_SBC: SBC
- AVDTP\_CODEC\_TYPE\_MPEG1\_2\_AUDIO: MPEG-1,2 (used in MP3 files)
- AVDTP CODEC TYPE MPEG2 4 AAC: MPEG-2,4 AAC (used in Apple products)
- AVDTP CODEC TYPE ATRAC: ATRAC (used in Sony products)
- AVDTP\_CODEC\_TYPE\_NON\_A2DP: Non-A2DP Codec

## 4.102 bt avdtp event t Union Reference

Parameter to an application callback.

#include <avdtp.h>

#### **Data Fields**

- bt\_avdtp\_evt\_ctrl\_channel\_connected\_t ctrl\_channel\_connected
  - Valid if event is AVDTP EVT CTRL CHANNEL CONNECTED.
- bt\_avdtp\_evt\_ctrl\_channel\_disconnected\_t ctrl\_channel\_disconnected
  - Valid if event is AVDTP\_EVT\_CTRL\_CHANNEL\_DISCONNECTED.
- · bt avdtp evt discover completed t discover completed
  - Valid if event is AVDTP\_EVT\_DISCOVER\_COMPLETED.
- bt\_avdtp\_evt\_sep\_info\_received\_t sep\_info\_received
  - Valid if event is AVDTP EVT SEP INFO RECEIVED.
- bt\_avdtp\_evt\_get\_sep\_capabilities\_completed\_t get\_sep\_capabilities\_completed
  - Valid if event is AVDTP\_EVT\_GET\_SEP\_CAPABILITIES\_COMPLETED.
- bt\_avdtp\_evt\_sep\_capabilities\_received\_t sep\_capabilities\_received
  - Valid if event is AVDTP EVT SEP CAPABILITIES RECEIVED.
- bt\_avdtp\_evt\_set\_stream\_configuration\_completed\_t set\_stream\_configuration\_completed
   Valid if event is AVDTP\_EVT\_SET\_STREAM\_CONFIGURATION\_COMPLETED.
- bt\_avdtp\_evt\_get\_stream\_configuration\_completed\_t get\_stream\_configuration\_completed Valid if event is AVDTP\_EVT\_GET\_STREAM\_CONFIGURATION\_COMPLETED.
- $\bullet \ \ bt\_avdtp\_evt\_stream\_reconfigure\_completed\_t\ stream\_reconfigure\_completed\\$ 
  - Valid if event is AVDTP\_EVT\_STREAM\_RECONFIGURE\_COMPLETED.
- bt\_avdtp\_evt\_open\_stream\_completed\_t open\_stream\_completed
  - Valid if event is AVDTP\_EVT\_OPEN\_STREAM\_COMPLETED.
- bt\_avdtp\_evt\_start\_stream\_completed\_t start\_stream\_completed
  - Valid if event is AVDTP\_EVT\_START\_STREAM\_COMPLETED.
- bt\_avdtp\_evt\_close\_stream\_completed\_t close\_stream\_completed
  - Valid if event is AVDTP\_EVT\_CLOSE\_STREAM\_COMPLETED.
- bt\_avdtp\_evt\_suspend\_stream\_completed\_t suspend\_stream\_completed
  - Valid if event is AVDTP\_EVT\_SUSPEND\_STREAM\_COMPLETED.
- bt\_avdtp\_evt\_stream\_security\_control\_completed\_t security\_control\_completed
  - Valid if event is AVDTP\_EVT\_STREAM\_SECURITY\_CONTROL\_COMPLETED.
- bt\_avdtp\_evt\_set\_stream\_configuration\_requested\_t set\_stream\_configuration\_requested
  - Valid if event is AVDTP\_EVT\_SET\_STREAM\_CONFIGURATION\_REQUESTED.
- bt\_avdtp\_evt\_reconfigure\_stream\_requested\_t reconfigure\_stream\_requested
  - Valid if event is AVDTP\_EVT\_RECONFIGURE\_STREAM\_REQUESTED.
- bt\_avdtp\_evt\_open\_stream\_requested\_t open\_stream\_requested
  - Valid if event is AVDTP\_EVT\_OPEN\_STREAM\_REQUESTED.
- bt\_avdtp\_evt\_start\_stream\_requested\_t start\_stream\_requested
  - Valid if event is AVDTP\_EVT\_START\_STREAM\_REQUESTED.
- bt avdtp evt suspend stream requested t suspend stream requested
  - Valid if event is AVDTP\_EVT\_SUSPEND\_STREAM\_REQUESTED.
- bt\_avdtp\_evt\_close\_stream\_requested\_t close\_stream\_requested
  - Valid if event is AVDTP EVT CLOSE STREAM REQUESTED.
- bt\_avdtp\_evt\_abort\_stream\_requested\_t abort\_stream\_requested
  - Valid if event is AVDTP EVT ABORT STREAM REQUESTED.
- bt\_avdtp\_evt\_delay\_report\_completed\_t delay\_report\_completed

Valid if event is AVDTP\_EVT\_DELAYREPORT\_COMPLETED.

• bt\_avdtp\_evt\_stream\_configured\_t stream\_configured

Valid if event is AVDTP\_EVT\_STREAM\_CONFIGURED.

bt\_avdtp\_evt\_stream\_reconfigured\_t stream\_reconfigured

Valid if event is AVDTP\_EVT\_STREAM\_RECONFIGURED.

• bt\_avdtp\_evt\_stream\_opened\_t stream\_opened

Valid if event is AVDTP\_EVT\_STREAM\_OPENED.

• bt\_avdtp\_evt\_stream\_started\_t stream\_started

Valid if event is AVDTP\_EVT\_STREAM\_STARTED.

• bt\_avdtp\_evt\_stream\_suspended\_t stream\_suspended

Valid if event is AVDTP EVT STREAM SUSPENDED.

bt\_avdtp\_evt\_stream\_closed\_t stream\_closed

Valid if event is AVDTP\_EVT\_STREAM\_CLOSED.

bt\_avdtp\_evt\_stream\_aborted\_t stream\_aborted

Valid if event is AVDTP\_EVT\_STREAM\_ABORTED.

bt\_avdtp\_evt\_media\_packet\_received\_t media\_packet\_received

Valid if event is AVDTP\_EVT\_MEDIA\_PACKET\_RECEIVED.

bt\_avdtp\_evt\_media\_packet\_sent\_t media\_packet\_sent

Valid if event is AVDTP\_EVT\_MEDIA\_PACKET\_SENT.

• bt\_avdtp\_evt\_media\_packet\_send\_failed\_t media\_packet\_send\_failed

Valid if event is AVDTP\_EVT\_MEDIA\_PACKET\_SEND\_FAILED.

## 4.102.1 Detailed Description

Parameter to an application callback.

This union is used to pass event specific data to the AVDTP consumer. Which member of the union points to a valid structure depends on the event reported to the consumer. In general, each event has a corresponding member in the union.

# 4.103 bt\_avdtp\_evt\_abort\_stream\_requested\_t Struct Reference

Parameter to AVDTP\_EVT\_ABORT\_STREAM\_REQUESTED event.

```
#include <avdtp.h>
```

#### **Data Fields**

• bt byte err code

The result to be sent to the remote party.

bt\_byte strm\_handle

The handle of a stream to abort.

### 4.103.1 Detailed Description

Parameter to AVDTP\_EVT\_ABORT\_STREAM\_REQUESTED event.

A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt\_avdtp\_event\_t union - bt\_avdtp\_event\_t::abort\_stream\_requested - when AVDTP received a "abort stream" request.

### 4.103.2 Field Documentation

4.103.2.1 bt\_byte err\_code

The result to be sent to the remote party.

• If local device accepts the configuration requested by the remote device it should set err\_code to AVDT ← P\_ERROR\_SUCCESS. Otherwise it should set err\_code to one of the AVDTP\_ERROR\_constants.

# 4.104 bt\_avdtp\_evt\_close\_stream\_completed\_t Struct Reference

Parameter to AVDTP\_EVT\_CLOSE\_STREAM\_COMPLETED event.

```
#include <avdtp.h>
```

### **Data Fields**

• bt\_byte err\_code

The result of the request.

bt\_byte strm\_handle

Stream handle.

## 4.104.1 Detailed Description

Parameter to AVDTP\_EVT\_CLOSE\_STREAM\_COMPLETED event.

A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt\_avdtp\_event\_t union - bt\_avdtp\_event\_t::close\_stream\_completed - when AVDTP received a response to a "close stream" request.

### 4.104.2 Field Documentation

```
4.104.2.1 bt_byte err_code
```

The result of the request.

- If the remote accepted the request err\_code == AVDTP\_ERROR\_SUCCESS.
- Otherwise err\_code == the error code returned by the remote party.

## 4.105 bt\_avdtp\_evt\_close\_stream\_requested\_t Struct Reference

Parameter to AVDTP EVT CLOSE STREAM REQUESTED event.

```
#include <avdtp.h>
```

## **Data Fields**

• bt\_byte err\_code

The result to be sent to the remote party.

• bt\_byte strm\_handle

The handle of a stream to close.

## 4.105.1 Detailed Description

Parameter to AVDTP\_EVT\_CLOSE\_STREAM\_REQUESTED event.

A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt\_avdtp\_event\_t union - bt\_avdtp\_event\_t::close\_stream\_requested - when AVDTP received a "close stream" request.

#### 4.105.2 Field Documentation

```
4.105.2.1 bt_byte err_code
```

The result to be sent to the remote party.

If local device accepts the configuration requested by the remote device it should set err\_code to AVDT
 —
 P ERROR SUCCESS. Otherwise it should set err\_code to one of the AVDTP ERROR constants.

# 4.106 bt avdtp evt ctrl channel connected t Struct Reference

```
Parameter to AVDTP_EVT_CTRL_CHANNEL_CONNECTED event.
```

```
#include <avdtp.h>
```

### **Data Fields**

 bt\_bdaddr\_t \* bdaddr the address of a remote device

## 4.106.1 Detailed Description

Parameter to AVDTP\_EVT\_CTRL\_CHANNEL\_CONNECTED event.

A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt\_avdtp\_cevent\_t union - bt\_avdtp\_event\_t::ctrl\_channel\_connected - when a control channel between two devices has been established.

## 4.107 bt\_avdtp\_evt\_ctrl\_channel\_disconnected\_t Struct Reference

Parameter to AVDTP\_EVT\_CTRL\_CHANNEL\_DISCONNECTED event.

```
#include <avdtp.h>
```

#### **Data Fields**

 bt\_bdaddr\_t \* bdaddr the address of a remote device

### 4.107.1 Detailed Description

Parameter to AVDTP\_EVT\_CTRL\_CHANNEL\_DISCONNECTED event.

A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt\_avdtp\_event \_\_t union - bt\_avdtp\_event\_t::ctrl\_channel\_disconnected - when a control channel between two devices has been terminated.

# 4.108 bt\_avdtp\_evt\_delay\_report\_completed\_t Struct Reference

Parameter to AVDTP\_EVT\_DELAYREPORT\_COMPLETED event.

#include <avdtp.h>

### **Data Fields**

· bt byte err code

The result of the request.

### 4.108.1 Detailed Description

Parameter to AVDTP\_EVT\_DELAYREPORT\_COMPLETED event.

A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt\_avdtp\_event\_t union - bt\_avdtp\_event\_t::delay\_report\_completed - when AVDTP received a response to a "delay report" request.

### 4.108.2 Field Documentation

4.108.2.1 bt\_byte err\_code

The result of the request.

- If the remote accepted the request err\_code == AVDTP\_ERROR\_SUCCESS.
- Otherwise err\_code == the error code returned by the remote party.

## 4.109 bt\_avdtp\_evt\_discover\_completed\_t Struct Reference

Parameter to AVDTP\_EVT\_DISCOVER\_COMPLETED event.

#include <avdtp.h>

## **Data Fields**

· bt byte err code

The result of discovering.

## 4.109.1 Detailed Description

Parameter to AVDTP\_EVT\_DISCOVER\_COMPLETED event.

A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt\_avdtp\_event\_t union - bt\_avdtp\_event\_t::discover\_completed - when AVDTP completed discovering SEPs available on a remote device.

#### 4.109.2 Field Documentation

4.109.2.1 bt\_byte err\_code

The result of discovering.

- If the remote accepted the request err\_code == AVDTP\_ERROR\_SUCCESS.
- Otherwise err code == the error code returned by the remote party.

# 4.110 bt\_avdtp\_evt\_get\_sep\_capabilities\_completed\_t Struct Reference

Parameter to AVDTP\_EVT\_GET\_SEP\_CAPABILITIES\_COMPLETED event.

#include <avdtp.h>

#### **Data Fields**

· bt\_byte err\_code

The result of the request.

### 4.110.1 Detailed Description

Parameter to AVDTP\_EVT\_GET\_SEP\_CAPABILITIES\_COMPLETED event.

A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt\_avdtp\_covent\_t union - bt\_avdtp\_event\_t::get\_sep\_capabilities\_completed - when AVDTP received a response to a "get SEP capabilities" request. AVDTP\_EVT\_GET\_SEP\_CAPABILITIES\_COMPLETED only informs the status of the request - success or failure. In case of success another event - AVDTP\_EVT\_SEP\_CAPABILITIES\_RECEIVED - is generate with a pointer to a structure that holds actual SEP's capabilities.

#### 4.110.2 Field Documentation

4.110.2.1 bt\_byte err\_code

The result of the request.

- If the remote accepted the request err\_code == AVDTP ERROR SUCCESS.
- Otherwise err\_code == the error code returned by the remote party.

## 4.111 bt\_avdtp\_evt\_get\_stream\_configuration\_completed\_t Struct Reference

Parameter to AVDTP EVT SET STREAM CONFIGURATION COMPLETED event.

#include <avdtp.h>

## 4.111.1 Detailed Description

Parameter to AVDTP\_EVT\_SET\_STREAM\_CONFIGURATION\_COMPLETED event.

A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt\_avdtp\_event \_\_t union - bt\_avdtp\_event\_t::get\_stream\_configuration\_completed - when AVDTP received a response to a "get stream configuration" request. AVDTP\_EVT\_SET\_STREAM\_CONFIGURATION\_COMPLETED only informs the status of the request - success or failure. In case of success another event - AVDTP\_EVT\_STREAM\_CONFIGU ← RATION\_RECEIVED - is generate with a pointer to a structure that hold actual stream's configuration.

# 4.112 bt\_avdtp\_evt\_media\_packet\_received\_t Struct Reference

Parameter to AVDTP\_EVT\_MEDIA\_PACKET\_RECEIVED event.

```
#include <avdtp.h>
```

#### **Data Fields**

· bt byte strm handle

The handle of a stream which received a packet.

bt\_media\_packet\_t \* packet

A pointer to a media packet buffer.

## 4.112.1 Detailed Description

Parameter to AVDTP\_EVT\_MEDIA\_PACKET\_RECEIVED event.

A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt\_avdtp\_event \_\_t union - bt\_avdtp\_event\_t::media\_packet\_received - when ADVDTP received a media packet from the remote device.

# 4.113 bt\_avdtp\_evt\_media\_packet\_send\_failed\_t Struct Reference

Parameter to AVDTP\_EVT\_MEDIA\_PACKET\_SEND\_FAILED event.

```
#include <avdtp.h>
```

## **Data Fields**

• bt\_byte strm\_handle

The handle of a stream which received a packet.

bt\_media\_packet\_t \* packet

A pointer to a media packet buffer.

## 4.113.1 Detailed Description

Parameter to AVDTP\_EVT\_MEDIA\_PACKET\_SEND\_FAILED event.

A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt\_avdtp\_event\_t union - bt\_avdtp\_event\_t::media\_packet\_send\_failed - when ADVDTP failed to send a media packet to the remote device.

## 4.114 bt\_avdtp\_evt\_media\_packet\_sent\_t Struct Reference

Parameter to AVDTP\_EVT\_MEDIA\_PACKET\_SENT event.

```
#include <avdtp.h>
```

• bt\_byte strm\_handle

The handle of a stream which received a packet.

bt\_media\_packet\_t \* packet

A pointer to a media packet buffer.

## 4.114.1 Detailed Description

Parameter to AVDTP\_EVT\_MEDIA\_PACKET\_SENT event.

A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt\_avdtp\_event\_t union - bt\_avdtp\_event\_t::media\_packet\_sent - when ADVDTP sent a media packet to the remote device.

# 4.115 bt\_avdtp\_evt\_open\_stream\_completed\_t Struct Reference

Parameter to AVDTP\_EVT\_OPEN\_STREAM\_COMPLETED event.

```
#include <avdtp.h>
```

#### **Data Fields**

· bt\_byte err\_code

The result of the request.

• bt\_byte strm\_handle

Stream handle.

## 4.115.1 Detailed Description

Parameter to AVDTP\_EVT\_OPEN\_STREAM\_COMPLETED event.

A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt\_avdtp\_event\_t union - bt\_avdtp\_event\_t::open\_stream\_completed - when AVDTP received a response to a "open stream" request.

## 4.115.2 Field Documentation

4.115.2.1 bt\_byte err\_code

The result of the request.

- If the remote accepted the request err\_code == AVDTP\_ERROR\_SUCCESS.
- Otherwise err\_code == the error code returned by the remote party.

# 4.116 bt\_avdtp\_evt\_open\_stream\_requested\_t Struct Reference

Parameter to AVDTP\_EVT\_OPEN\_STREAM\_REQUESTED event.

```
#include <avdtp.h>
```

· bt\_byte err\_code

The result to be sent to the remote party.

· bt byte strm handle

The handle of a stream to open.

### 4.116.1 Detailed Description

Parameter to AVDTP EVT OPEN STREAM REQUESTED event.

A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt\_avdtp\_event\_t union - bt\_avdtp\_event\_t::open\_stream\_requested - when AVDTP received a "open stream" request.

#### 4.116.2 Field Documentation

```
4.116.2.1 bt_byte err_code
```

The result to be sent to the remote party.

If local device accepts the configuration requested by the remote device it should set err\_code to AVDT
 —
 P ERROR SUCCESS. Otherwise it should set err\_code to one of the AVDTP ERROR constants.

# 4.117 bt\_avdtp\_evt\_reconfigure\_stream\_requested\_t Struct Reference

Parameter to AVDTP\_EVT\_RECONFIGURE\_STREAM\_REQUESTED event.

```
#include <avdtp.h>
```

## **Data Fields**

bt\_byte err\_code

The result to be sent to the remote party.

bt\_byte err\_category

If local device cannot accept the request it should set <code>err\_category</code> to the value of the forst Service Category that failed.

• bt avdtp sep t \* sep

Description of the local SEP.

• bt\_avdtp\_sep\_capabilities\_t \* config

Stream configuration requested by the remote party.

• bt\_byte strm\_handle

Stream handle.

### 4.117.1 Detailed Description

Parameter to AVDTP\_EVT\_RECONFIGURE\_STREAM\_REQUESTED event.

A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt\_avdtp\_event\_t union - bt\_avdtp\_event\_t::reconfigure\_stream\_requested - when AVDTP received a "change stream configuration" request.

### 4.117.2 Field Documentation

4.117.2.1 bt\_byte err\_code

The result to be sent to the remote party.

• If local device accepts the configuration requested by the remote device it should set err\_code to AVDT P ERROR SUCCESS. Otherwise it should set err\_code to one of the AVDTP ERROR constants.

# 4.118 bt\_avdtp\_evt\_sep\_capabilities\_received\_t Struct Reference

Parameter to AVDTP\_EVT\_SEP\_CAPABILITIES\_RECEIVED and AVDTP\_EVT\_STREAM\_CONFIGURATION\_← RECEIVED events.

```
#include <avdtp.h>
```

#### **Data Fields**

• bt\_avdtp\_sep\_capabilities\_t \* caps

SEP capabilities or stream configuration.

## 4.118.1 Detailed Description

Parameter to AVDTP\_EVT\_SEP\_CAPABILITIES\_RECEIVED and AVDTP\_EVT\_STREAM\_CONFIGURATION\_← RECEIVED events.

A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt\_avdtp\_event \_\_t union - bt\_avdtp\_event\_t::sep\_capabilities\_received - when AVDTP received a positive response to a "get SEP capabilities" or "get stream configuration" request.

## 4.119 bt avdtp evt sep info received t Struct Reference

Parameter to AVDTP\_EVT\_SEP\_INFO\_RECEIVED event.

```
#include <avdtp.h>
```

## **Data Fields**

bt\_byte sep\_id

SEP ID.

• bt\_byte sep\_type

SEP type. This can be either AVDTP\_SEP\_TYPE\_SOURCE or AVDTP\_SEP\_TYPE\_SINK.

· bt\_bool in\_use

A flag indicating if a SEP is already being used.

• bt\_byte media\_type

Type of media supported by this SEP.

## 4.119.1 Detailed Description

Parameter to AVDTP\_EVT\_SEP\_INFO\_RECEIVED event.

A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt\_avdtp\_event\_t union - bt\_avdtp\_event\_t::sep\_info\_received - when AVDTP received positive result to a "discover" request. AVD $\leftarrow$  TP\_EVT\_SEP\_INFO\_RECEIVED is generated for every SEP received from the remote device.

#### 4.119.2 Field Documentation

4.119.2.1 bt\_byte media\_type

Type of media supported by this SEP.

This can be on of the following values:

- AVDTP\_MEDIA\_TYPE\_AUDIO
- AVDTP\_MEDIA\_TYPE\_VIDEO
- · AVDTP MEDIA TYPE MULTIMEDIA

# 4.120 bt\_avdtp\_evt\_set\_stream\_configuration\_completed\_t Struct Reference

Parameter to AVDTP\_EVT\_SET\_STREAM\_CONFIGURATION\_COMPLETED event.

#include <avdtp.h>

### **Data Fields**

bt\_byte err\_code

The result of the request.

bt\_byte strm\_handle

Stream handle.

bt\_byte svc\_category

The value of the first Service Category to fail if the remote rejected the request.

## 4.120.1 Detailed Description

Parameter to AVDTP\_EVT\_SET\_STREAM\_CONFIGURATION\_COMPLETED event.

A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt\_avdtp\_event\_t union - bt\_avdtp\_event\_t::set\_stream\_configuration\_completed - when AVDTP received a response to a "set stream configuration" request.

### 4.120.2 Field Documentation

4.120.2.1 bt\_byte err\_code

The result of the request.

- If the remote accepted the request err\_code == AVDTP\_ERROR\_SUCCESS.
- Otherwise err\_code == the error code returned by the remote party.

# 4.121 bt\_avdtp\_evt\_set\_stream\_configuration\_requested\_t Struct Reference

Parameter to AVDTP\_EVT\_SET\_STREAM\_CONFIGURATION\_REQUESTED event.

#include <avdtp.h>

## **Data Fields**

• bt byte err code

The result to be sent to the remote party.

bt\_byte err\_category

If local device cannot accept the request it should set <code>err\_category</code> to the value of the forst Service Category that failed.

bt\_avdtp\_sep\_t \* sep

Description of the local SEP.

• bt\_byte int\_sep\_id

The ID of the remote SEP.

bt\_avdtp\_sep\_capabilities\_t \* config

Stream configuration requested by the remote party.

• bt byte strm handle

Stream handle.

### 4.121.1 Detailed Description

Parameter to AVDTP\_EVT\_SET\_STREAM\_CONFIGURATION\_REQUESTED event.

A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt\_avdtp\_event\_t union - bt\_avdtp\_event\_t::set\_stream\_configuration\_requested - when AVDTP received a "set stream configuration" request.

#### 4.121.2 Field Documentation

4.121.2.1 bt\_byte err\_code

The result to be sent to the remote party.

• If local device accepts the configuration requested by the remote device it should set err\_code to AVDT PERROR\_SUCCESS. Otherwise it should set err\_code to one of the AVDTP\_ERROR\_constants.

## 4.122 bt avdtp evt start stream completed t Struct Reference

Parameter to AVDTP\_EVT\_START\_STREAM\_COMPLETED event.

#include <avdtp.h>

## **Data Fields**

· bt\_byte err\_code

The result of the request.

• bt\_byte strm\_handle

Stream handle.

## 4.122.1 Detailed Description

Parameter to AVDTP\_EVT\_START\_STREAM\_COMPLETED event.

A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt\_avdtp\_event\_t union - bt\_avdtp\_event\_t::start\_stream\_completed - when AVDTP received a response to a "start stream" request.

### 4.122.2 Field Documentation

4.122.2.1 bt byte err code

The result of the request.

- If the remote accepted the request err\_code == AVDTP ERROR SUCCESS.
- Otherwise err\_code == the error code returned by the remote party.

# 4.123 bt\_avdtp\_evt\_start\_stream\_requested\_t Struct Reference

Parameter to AVDTP EVT START STREAM REQUESTED event.

```
#include <avdtp.h>
```

#### **Data Fields**

· bt byte err code

The result to be sent to the remote party.

bt\_byte strm\_handle

The handle of a stream to start.

## 4.123.1 Detailed Description

Parameter to AVDTP\_EVT\_START\_STREAM\_REQUESTED event.

A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt\_avdtp\_event\_t union - bt\_avdtp\_event\_t::start\_stream\_requested - when AVDTP received a "start stream" request.

# 4.123.2 Field Documentation

4.123.2.1 bt\_byte err\_code

The result to be sent to the remote party.

If local device accepts the configuration requested by the remote device it should set err\_code to AVDT

P\_ERROR\_SUCCESS. Otherwise it should set err\_code to one of the AVDTP\_ERROR\_constants.

## 4.124 bt\_avdtp\_evt\_stream\_aborted\_t Struct Reference

Parameter to AVDTP\_EVT\_STREAM\_ABORTED event.

```
#include <avdtp.h>
```

· bt\_byte strm\_handle

The handle of a stream that has been aborted.

## 4.124.1 Detailed Description

Parameter to AVDTP EVT STREAM ABORTED event.

A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt\_avdtp\_event \_\_t union - bt\_avdtp\_event\_t::stream\_closed - to notify the AVDTP consumer that a stream has been successfully aborted.

# 4.125 bt\_avdtp\_evt\_stream\_closed\_t Struct Reference

Parameter to AVDTP EVT STREAM CLOSED event.

```
#include <avdtp.h>
```

### **Data Fields**

bt\_byte strm\_handle

The handle of a stream that has been closed.

## 4.125.1 Detailed Description

Parameter to AVDTP\_EVT\_STREAM\_CLOSED event.

A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt\_avdtp\_event \_\_t union - bt\_avdtp\_event\_t::stream\_closed - to notify the AVDTP consumer that a stream has been successfully closed.

# 4.126 bt\_avdtp\_evt\_stream\_configured\_t Struct Reference

Parameter to AVDTP\_EVT\_STREAM\_CONFIGURED event.

```
#include <avdtp.h>
```

## **Data Fields**

bt\_byte strm\_handle

The handle of a stream that has been configured.

### 4.126.1 Detailed Description

Parameter to AVDTP\_EVT\_STREAM\_CONFIGURED event.

A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt\_avdtp\_event\_t union - bt\_avdtp\_event\_t::stream\_configured - to notify the AVDTP consumer that a stream configuration has been successfully completed.

# 4.127 bt\_avdtp\_evt\_stream\_opened\_t Struct Reference

Parameter to AVDTP EVT STREAM OPENED event.

#include <avdtp.h>

#### **Data Fields**

· bt\_byte strm\_handle

The handle of a stream that has been opened.

## 4.127.1 Detailed Description

Parameter to AVDTP\_EVT\_STREAM\_OPENED event.

A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt\_avdtp\_event\_t union - bt\_avdtp\_event\_t::stream\_opened - to notify the AVDTP consumer that a stream has been successfully opened.

# 4.128 bt\_avdtp\_evt\_stream\_reconfigure\_completed\_t Struct Reference

Parameter to AVDTP\_EVT\_STREAM\_RECONFIGURE\_COMPLETED event.

#include <avdtp.h>

### **Data Fields**

bt\_byte err\_code

The result of the request.

bt\_byte strm\_handle

Stream handle.

bt\_byte svc\_category

The value of the first Service Category to fail if the remote rejected the request.

# 4.128.1 Detailed Description

Parameter to AVDTP EVT STREAM RECONFIGURE COMPLETED event.

A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt\_avdtp\_event\_t union - bt\_avdtp\_event\_t::stream\_reconfigure\_completed - when AVDTP received a response to a "change stream configuration" request.

## 4.128.2 Field Documentation

4.128.2.1 bt\_byte err\_code

The result of the request.

- If the remote accepted the request err\_code == AVDTP\_ERROR\_SUCCESS.
- Otherwise err\_code == the error code returned by the remote party.

# 4.129 bt\_avdtp\_evt\_stream\_reconfigured\_t Struct Reference

Parameter to AVDTP\_EVT\_STREAM\_RECONFIGURED event.

```
#include <avdtp.h>
```

#### **Data Fields**

· bt byte strm handle

The handle of a stream that has been reconfigured.

## 4.129.1 Detailed Description

Parameter to AVDTP\_EVT\_STREAM\_RECONFIGURED event.

A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt\_avdtp\_event\_t union - bt\_avdtp\_event\_t::stream\_reconfigured - to notify the AVDTP consumer that a stream configuration has been successfully changed.

# 4.130 bt\_avdtp\_evt\_stream\_security\_control\_completed\_t Struct Reference

Parameter to AVDTP\_EVT\_STREAM\_SECURITY\_CONTROL\_COMPLETED event.

```
#include <avdtp.h>
```

#### **Data Fields**

• bt\_byte err\_code

The result of the request.

## 4.130.1 Detailed Description

Parameter to AVDTP\_EVT\_STREAM\_SECURITY\_CONTROL\_COMPLETED event.

A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt\_avdtp\_event\_t union - bt\_avdtp\_event\_t::security\_control\_completed - when AVDTP received a response to a "exchange content protection control data" request.

### 4.130.2 Field Documentation

4.130.2.1 bt\_byte err\_code

The result of the request.

- If the remote accepted the request err\_code == AVDTP\_ERROR\_SUCCESS.
- Otherwise err\_code == the error code returned by the remote party.

## 4.131 bt avdtp evt stream started t Struct Reference

Parameter to AVDTP\_EVT\_STREAM\_STARTED event.

#include <avdtp.h>

· bt byte strm handle

The handle of a stream that has been started.

## 4.131.1 Detailed Description

Parameter to AVDTP\_EVT\_STREAM\_STARTED event.

A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt\_avdtp\_event—tunion - bt\_avdtp\_event\_t::stream\_started - to notify the AVDTP consumer that a stream has been successfully started.

# 4.132 bt\_avdtp\_evt\_stream\_suspended\_t Struct Reference

Parameter to AVDTP\_EVT\_STREAM\_SUSPENDED event.

```
#include <avdtp.h>
```

#### **Data Fields**

· bt\_byte strm\_handle

The handle of a stream that has been suspended.

### 4.132.1 Detailed Description

Parameter to AVDTP\_EVT\_STREAM\_SUSPENDED event.

A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt\_avdtp\_event\_t union - bt\_avdtp\_event\_t::stream\_suspended - to notify the AVDTP consumer that a stream has been successfully suspended.

# 4.133 bt\_avdtp\_evt\_suspend\_stream\_completed\_t Struct Reference

Parameter to AVDTP\_EVT\_SUSPEND\_STREAM\_COMPLETED event.

```
#include <avdtp.h>
```

### **Data Fields**

bt\_byte err\_code

The result of the request.

• bt\_byte strm\_handle

Stream handle.

### 4.133.1 Detailed Description

Parameter to AVDTP\_EVT\_SUSPEND\_STREAM\_COMPLETED event.

A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt\_avdtp\_event\_t union - bt\_avdtp\_event\_t::suspend\_stream\_completed - when AVDTP received a response to a "suspend stream" request.

### 4.133.2 Field Documentation

4.133.2.1 bt\_byte err\_code

The result of the request.

- If the remote accepted the request err\_code == AVDTP\_ERROR\_SUCCESS.
- Otherwise err\_code == the error code returned by the remote party.

# 4.134 bt\_avdtp\_evt\_suspend\_stream\_requested\_t Struct Reference

Parameter to AVDTP\_EVT\_SUSPEND\_STREAM\_REQUESTED event.

```
#include <avdtp.h>
```

### **Data Fields**

· bt\_byte err\_code

The result to be sent to the remote party.

• bt\_byte strm\_handle

The handle of a stream to suspend.

## 4.134.1 Detailed Description

Parameter to AVDTP\_EVT\_SUSPEND\_STREAM\_REQUESTED event.

A pointer to this structure is passed to the AVDTP application callback as a valid member of the bt\_avdtp\_event\_t union - bt\_avdtp\_event\_t::suspend\_stream\_requested - when AVDTP received a "suspend stream" request.

### 4.134.2 Field Documentation

```
4.134.2.1 bt_byte err_code
```

The result to be sent to the remote party.

• If local device accepts the configuration requested by the remote device it should set err\_code to AVDT ← P\_ERROR\_SUCCESS. Otherwise it should set err\_code to one of the AVDTP\_ERROR\_constants.

# 4.135 bt\_avdtp\_mgr\_t Struct Reference

## AVDTP manager.

```
#include <avdtp.h>
```

### **Data Fields**

· bt\_byte state

Manager state.

• bt\_byte flags

Additional manager state.

• bt\_avdtp\_sep\_t \* seps

Pointer to a buffer of SEPs available for allocating with bt\_avdtp\_register\_sep.

· bt byte next sep id

Holds ID of the next SEP to be allocated. Every time bt\_avdtp\_register\_sep is called this value is increased by 1.

• bt\_avdtp\_stream\_t \* streams

Pointer to a buffer of streams available for allocating with bt\_avdtp\_create\_stream.

bt byte next stream handle

Holds ID of the next stream to be allocated. Every time bt\_avdtp\_create\_stream is called this value is increased by 1.

bt avdtp control channel t \* control channels

Pointer to a buffer of available control channles.

• bt\_avdtp\_transport\_channel\_t \* transport\_channels

Pointer to a buffer of available transport channles.

bt\_avdtp\_codec\_t \* codecs

A list of supported codecs.

bt\_avdtp\_mgr\_callback\_fp callback

Pointer to a function which a AVDTP consumer must register in order to be notified of various events.

void \* callback\_param

Pointer to arbitrary data to be passed to the callback.

bt\_avdtp\_control\_cmd\_t \* pending\_command

If local device wants to send a command to a remote device but control channel does not exists this member holds a pointer to the command until the channel is created.

bt avdtp stream t \* opening strm

Holds a pointer to a stream being opened by a remote device.

### 4.135.1 Detailed Description

AVDTP manager.

A structure that glues all pieces together. There is only one instance of this structure allocated by dotstack. A pointer to the instance can be get with bt\_avdtp\_get\_mgr().

## 4.135.2 Field Documentation

4.135.2.1 bt\_byte flags

Additional manager state.

This value can be a combination of the following values:

AVDTP\_MANAGER\_FLAG\_SENDING\_MEDIA\_PACKET

4.135.2.2 bt\_byte state

Manager state.

This value can be one of the following values:

- AVDTP\_MANAGER\_STATE\_IDLE
- AVDTP\_MANAGER\_STATE\_CONNECTING

# 4.136 bt\_avdtp\_sep\_capabilities\_t Struct Reference

### SEP capabilities.

#include <avdtp.h>

### **Data Fields**

· bt\_byte categories

Defines service capabilities exposed by a SEP to a remote party.

bt\_byte media\_type

Type of media supported by this SEP.

• bt\_byte codec\_type

Codec type supported by this SEP.

void \* codec\_config

Pointer to a buffer that holds codec specific capabilities.

• bt\_uint cp\_type

Type of content protection supported by this SEP.

• bt\_byte \* cp\_specific

Pointer to a buffer holding content protection specific data.

• bt\_byte cp\_specific\_len

Length of the content protection specific data.

• bt\_byte recovery\_type

Type of recovery supported by this SEP.

· bt\_byte max\_recovery\_window

Recovery window size.

• bt\_byte max\_parity\_code\_packets

Maximum number of parity codec packets.

bt\_byte header\_compression\_options

Header compression configuration.

bt\_byte multiplexing\_options

Multiplexing configuration.

· bt\_byte tsid\_media

ID of the media transport session.

bt\_byte tcid\_media

ID of the media transport channel.

bt\_byte tsid\_reporting

ID of the reporting transport session.

• bt\_byte tcid\_reporting

ID of the reporting transport channel.

bt\_byte tsid\_recovery

ID of the recovery transport session.

bt\_byte tcid\_recovery

ID of the recovery transport channel.

# 4.136.1 Detailed Description

## SEP capabilities.

This structure is used to hold SEP capabilities.

### 4.136.2 Field Documentation

### 4.136.2.1 bt\_byte categories

Defines service capabilities exposed by a SEP to a remote party.

This can be a combination of the following values:

- AVDTP\_SEP\_CAPABILITY\_FLAG\_MEDIA\_TRANSPORT
- AVDTP\_SEP\_CAPABILITY\_FLAG\_REPORTING
- · AVDTP SEP CAPABILITY FLAG RECOVERY
- AVDTP\_SEP\_CAPABILITY\_FLAG\_CONTENT\_PROTECTION
- AVDTP\_SEP\_CAPABILITY\_FLAG\_HEADER\_COMPRESSION
- AVDTP\_SEP\_CAPABILITY\_FLAG\_MULTIPLEXING
- AVDTP\_SEP\_CAPABILITY\_FLAG\_MEDIA\_CODEC

### 4.136.2.2 bt\_byte codec\_type

Codec type supported by this SEP.

This can be on of the following values:

- AVDTP\_CODEC\_TYPE\_SBC
- AVDTP\_CODEC\_TYPE\_MPEG1\_2\_AUDIO
- AVDTP\_CODEC\_TYPE\_MPEG2\_4\_AAC
- AVDTP\_CODEC\_TYPE\_ATRAC
- AVDTP\_CODEC\_TYPE\_NON\_A2DP

### 4.136.2.3 bt\_byte\* cp\_specific

Pointer to a buffer holding content protection specific data.

Note

Content protection is currently not supported by dotstack.

```
4.136.2.4 bt_byte cp_specific_len
```

Length of the content protection specific data.

Note

Content protection is currently not supported by dotstack.

## 4.136.2.5 bt\_uint cp\_type

Type of content protection supported by this SEP.

Note

Content protection is currently not supported by dotstack.

4.136.2.6 bt\_byte header\_compression\_options

Header compression configuration.

Note

Header compression is currently not supported by dotstack.

4.136.2.7 bt\_byte max\_parity\_code\_packets

Maximum number of parity codec packets.

Note

Recovery is currently not supported by dotstack.

4.136.2.8 bt\_byte max\_recovery\_window

Recovery window size.

Note

Recovery is currently not supported by dotstack.

4.136.2.9 bt\_byte media\_type

Type of media supported by this SEP.

This can be on of the following values:

- AVDTP\_MEDIA\_TYPE\_AUDIO
- AVDTP\_MEDIA\_TYPE\_VIDEO
- AVDTP\_MEDIA\_TYPE\_MULTIMEDIA

4.136.2.10 bt\_byte multiplexing\_options

Multiplexing configuration.

Note

Multiplexing is currently not supported by dotstack.

4.136.2.11 bt\_byte recovery\_type

Type of recovery supported by this SEP.

Note

Recovery is currently not supported by dotstack.

# 4.137 bt\_avdtp\_sep\_t Struct Reference

## SEP description.

```
#include <avdtp.h>
```

### **Data Fields**

• bt\_avdtp\_sep\_t \* next\_sep

Pointer to next SEP.

• bt\_byte id

ID of the SEP.

• bt\_byte type

Type of the SEP.

const bt\_avdtp\_sep\_capabilities\_t \* caps

SEP capabilities.

• bt\_byte state

State of the SEP buffer.

## 4.137.1 Detailed Description

SEP description.

This structure is used to hold information about SEPs available on a local device.

## 4.137.2 Field Documentation

4.137.2.1 bt\_byte state

State of the SEP buffer.

This can be one of the following values:

- AVDTP\_SEP\_STATE\_FREE
- AVDTP\_SEP\_STATE\_IDLE

4.137.2.2 bt\_byte type

Type of the SEP.

This can be one of the following values:

- AVDTP\_SEP\_TYPE\_SOURCE
- AVDTP\_SEP\_TYPE\_SINK

# 4.138 bt\_avdtp\_stream\_t Struct Reference

Stream description.

#include <avdtp.h>

• struct \_bt\_avdtp\_stream\_t \* next\_stream

Pointer to next stream.

bt\_byte handle

Stream handle. This values is allocated by dotstack and is used to manipulate the stream by the AVDTP consumer.

· bt\_byte state

State of the stream.

· bt byte flags

Additional stream state.

struct \_bt\_avdtp\_sep\_t \* sep

Local SEP this stream is connected to.

• bt\_avdtp\_sep\_capabilities\_t \* config

Current SEP configuration.

• bt\_byte remote\_seid

ID of the remote SEP.

• bt\_bdaddr\_t remote\_addr

BT address of the remote device.

bt\_avdtp\_transport\_session\_t sessions [AVDTP\_MAX\_STREAM\_TRANSPORT\_SESSION]

List of transport session available/active on this stream.

• struct \_bt\_avdtp\_mgr\_t \* mgr

AVDTP manager this stream belongs to.

bt\_byte cur\_channel\_index

This value is currenlty not used.

• bt\_byte \* listen\_sep\_list

A list of SEPs this channel this channel is listening on (i.e. can accept incoming connection requests).

bt\_byte listen\_sep\_count

 $\textit{The number of SEPs in } \verb|listen_sep_list|.$ 

• bt\_queue\_element\_t \* media\_rx\_queue

A list of media packet buffer for receiving incoming packets.

bt\_queue\_element\_t \* media\_tx\_queue

A list of media packet buffer to be sent to a remote device.

# 4.138.1 Detailed Description

Stream description.

This structure is used to hold information about streams available on a local device.

### 4.138.2 Field Documentation

4.138.2.1 bt\_byte flags

Additional stream state.

This value can be one of the following values:

AVDTP STREAM FLAG LISTENING

4.138.2.2 bt\_avdtp\_transport\_session\_t sessions[AVDTP\_MAX\_STREAM\_TRANSPORT\_SESSION]

List of transport session available/active on this stream.

Note

There can be up to 3 (AVDTP\_MAX\_STREAM\_TRANSPORT\_SESSION == 3) transport sessions on a stream - media, reporting and recovery. dotstack supports only media sessions so the other two are never used.

4.138.2.3 bt\_byte state

State of the stream.

This value can be one of the following values:

- AVDTP\_STREAM\_STATE\_IDLE
- AVDTP\_STREAM\_OPENING\_TRANSPORT\_CHANNELS
- AVDTP\_STREAM\_CLOSING\_TRANSPORT\_CHANNELS
- AVDTP\_STREAM\_STATE\_CONFIGURED
- · AVDTP STREAM STATE OPEN
- · AVDTP STREAM STATE STREAMING
- AVDTP STREAM STATE CLOSING
- AVDTP\_STREAM\_STATE\_ABORTING

# 4.139 bt\_avdtp\_transport\_channel\_t Struct Reference

Transport channel description.

#include <avdtp.h>

#### **Data Fields**

- struct \_bt\_avdtp\_transport\_channel\_t \* next\_channel
  - Pointer to next channel.
- bt\_byte id

ID of the channel.

bt\_byte type

Type of the channel.

bt\_l2cap\_channel\_t \* l2cap\_channel

L2CAp channel used to transfer this AVDTP channel's data.

• bt\_byte ref\_count

Channel's reference count.

bt\_byte connect\_ref\_count

This value is currently not used.

# 4.139.1 Detailed Description

Transport channel description.

This structure is used to hold information about transport channels available on a local device.

## 4.139.2 Field Documentation

4.139.2.1 bt\_byte ref\_count

Channel's reference count.

This value is intended for use with shared channels. When ref count reaches 0 the channel is closed.

Note

Shared channles (i.e. multiplexing) is currently not supported by dotstack

4.139.2.2 bt\_byte type

Type of the channel.

This can be one of the following values:

- AVDTP TRANSPORT CHANNEL TYPE DEDICATED
- AVDTP\_TRANSPORT\_CHANNEL\_TYPE\_SHARED
   Note

Shared channles (i.e. multiplexing) is currently not supported by dotstack

# 4.140 bt\_avdtp\_transport\_session\_t Struct Reference

Transport session description.

```
#include <avdtp.h>
```

## **Data Fields**

• bt\_byte id

ID of the transport session.

bt\_byte type

Type of the transport session.

bt\_avdtp\_transport\_channel\_t \* transport\_channel

Transport channel used for this transport session.

## 4.140.1 Detailed Description

Transport session description.

This structure is used to hold information about transport sessions available on a local device.

## 4.140.2 Field Documentation

4.140.2.1 bt\_byte type

Type of the transport session.

This can be one of the following values:

AVDTP\_TRANSPORT\_SESSION\_TYPE\_MEDIA

- AVDTP\_TRANSPORT\_SESSION\_TYPE\_REPORTING
- AVDTP TRANSPORT SESSION TYPE RECOVERY

## 4.141 bt avrcp channel t Struct Reference

AVRCP channel description.

#include <avrcp.h>

### 4.141.1 Detailed Description

AVRCP channel description.

This structure is used to hold information about an AVRCP channel.

# 4.142 bt\_avrcp\_device\_t Struct Reference

# 4.143 bt\_avrcp\_event\_t Union Reference

Parameter to an application callback.

#include <avrcp.h>

### **Data Fields**

- bt\_avrcp\_evt\_channel\_connected\_t channel\_connected
- Valid if event is AVRCP\_EVT\_CONTROL\_CHANNEL\_CONNECTED.

   bt\_avrcp\_evt\_channel\_disconnected\_t channel\_disconnected

Valid if event is AVRCP\_EVT\_CONTROL\_CHANNEL\_DISCONNECTED.

- · bt avrcp evt connection failed t connection failed
  - Valid if event is AVRCP\_EVT\_CONTROL\_CONNECTION\_FAILED.
- bt\_avrcp\_evt\_panel\_response\_received\_t panel\_response\_received
   Valid if event is AVRCP\_EVT\_PANEL\_RESPONSE\_RECEIVED.
- bt\_avrcp\_evt\_search\_completed\_t device\_search

Valid if event is AVRCP\_EVT\_SEARCH\_COMPLETED.

bt\_av\_capability\_company\_id\_t company\_id

Valid if event is AVRCP\_EVT\_COMPANY\_ID\_LIST\_RECEIVED.

- bt\_av\_capability\_event\_id\_t supported\_event\_id
  - Valid if event is AVRCP EVT EVENT ID LIST RECEIVED.
- bt\_av\_player\_settings\_t player\_settings

Valid if event is AVRCP\_EVT\_PLAYER\_SETTING\_ATTRIBUTES\_RECEIVED.

bt\_av\_player\_setting\_values\_t player\_setting\_values

Valid if event is AVRCP\_EVT\_PLAYER\_SETTING\_VALUES\_RECEIVED.

- bt\_av\_player\_setting\_current\_values\_t player\_setting\_current\_values
  - Valid if event is AVRCP\_EVT\_PLAYER\_CURRENT\_SETTING\_VALUES\_RECEIVED.
- bt\_av\_player\_settings\_text\_t player\_settings\_text

Valid if event is AVRCP\_EVT\_PLAYER\_SETTING\_ATTRIBUTES\_TEXT\_RECEIVED.

• bt\_av\_player\_setting\_values\_text\_t player\_setting\_values\_text

Valid if event is AVRCP\_EVT\_PLAYER\_SETTING\_VALUES\_TEXT\_RECEIVED.

• bt\_av\_element\_attributes\_t element\_attributes

Valid if event is AVRCP\_EVT\_GET\_ELEMENT\_ATTRIBUTES\_RECEIVED.

bt\_av\_play\_status\_t play\_status

Valid if event is AVRCP\_EVT\_GET\_PLAY\_STATUS\_RECEIVED.

bt\_av\_set\_absolute\_volume\_t absolute\_volume

Valid if event is AVRCP\_EVT\_SET\_ABSOLUTE\_VOLUME\_COMPLETED.

bt\_av\_set\_addressed\_player\_t addressed\_player

Valid if event is AVRCP\_EVT\_SET\_ADDRESSED\_PLAYER\_COMPLETED.

bt\_av\_play\_item\_t play\_item\_status

Valid if event is AVRCP\_EVT\_PLAY\_ITEM\_COMPLETED.

bt\_av\_add\_to\_now\_playing\_t add\_to\_now\_playing\_status

Valid if event is AVRCP\_EVT\_ADD\_TO\_NOW\_PLAYING\_COMPLETED.

bt\_av\_notification\_t notification

Valid if event is on of the following:

bt\_avrcp\_evt\_register\_events\_completed\_t register\_events

Valid if event AVRCP\_EVT\_REGISTER\_NOTIFICATIONS\_COMPLETED.

bt\_avrcp\_evt\_panel\_command\_received\_t panel\_command\_received

Valid if event is AVRCP\_EVT\_PANEL\_COMMAND\_RECEIVED.

bt\_av\_battery\_status\_of\_ct\_t battery\_status\_of\_ct

Valid if event is AVRCP\_EVT\_BATTERY\_STATUS\_OF\_CT\_RECEIVED.

bt\_av\_displayable\_character\_set\_t displayable\_character\_set

Valid if event is AVRCP\_EVT\_DISPLAYABLE\_CHARACTER\_SET\_RECEIVED.

bt\_av\_get\_element\_attributes\_t get\_element\_attributes

Valid if event is AVRCP\_EVT\_ELEMENT\_ATTRIBUTES\_REQUESTED.

bt\_av\_register\_notification\_t register\_notification

Valid if event is AVRCP\_EVT\_REGISTER\_NOTIFICATION\_REQUESTED.

## 4.143.1 Detailed Description

Parameter to an application callback.

This union is used to pass event specific data to the AVRCP consumer. Which member of the union points to a valid structure depends on the event reported to the consumer. In general, each event has a corresponding member in the union.

## 4.143.2 Field Documentation

4.143.2.1 bt\_av\_notification\_t notification

Valid if event is on of the following:

- AVRCP\_EVT\_PLAYBACK\_STATUS\_CHANGED
- AVRCP\_EVT\_TRACK\_CHANGED
- AVRCP EVT TRACK REACHED END
- AVRCP EVT TRACK REACHED START
- AVRCP\_EVT\_PLAYBACK\_POS\_CHANGED
- AVRCP\_EVT\_BATT\_STATUS\_CHANGED
- AVRCP\_EVT\_SYSTEM\_STATUS\_CHANGED

- AVRCP\_EVT\_PLAYER\_APPLICATION\_SETTING\_CHANGED
- · AVRCP EVT NOW PLAYING CONTENT CHANGED
- AVRCP\_EVT\_AVAILABLE\_PLAYERS\_CHANGED
- AVRCP\_EVT\_ADDRESSED\_PLAYER\_CHANGED
- · AVRCP EVT UIDS CHANGED
- AVRCP\_EVT\_VOLUME\_CHANGED

# 4.144 bt\_avrcp\_evt\_channel\_connected\_t Struct Reference

Parameter to AVRCP\_EVT\_CONTROL\_CHANNEL\_CONNECTED event.

```
#include <avrcp.h>
```

### **Data Fields**

 bt\_avrcp\_channel\_t \* channel AVRCP channel.

## 4.144.1 Detailed Description

Parameter to AVRCP\_EVT\_CONTROL\_CHANNEL\_CONNECTED event.

A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt\_avrcp\_event\_t union - bt\_avrcp\_event\_t::channel\_connected - when a control channel between two devices has been established.

# 4.145 bt\_avrcp\_evt\_channel\_disconnected\_t Struct Reference

Parameter to AVRCP EVT CONTROL CHANNEL DISCONNECTED event.

```
#include <avrcp.h>
```

## **Data Fields**

 bt\_avrcp\_channel\_t \* channel AVRCP channel.

### 4.145.1 Detailed Description

Parameter to AVRCP\_EVT\_CONTROL\_CHANNEL\_DISCONNECTED event.

A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt\_avrcp\_cevent\_t union - bt\_avrcp\_event\_t::channel\_disconnected - when a control channel between two devices has been terminated.

## 4.146 bt avrcp evt connection failed t Struct Reference

Parameter to AVRCP\_EVT\_CONTROL\_CONNECTION\_FAILED event.

```
#include <avrcp.h>
```

bt\_avrcp\_channel\_t \* channel
 AVRCP channel.

## 4.146.1 Detailed Description

Parameter to AVRCP\_EVT\_CONTROL\_CONNECTION\_FAILED event.

A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt\_avrcp\_event\_t union - bt\_avrcp\_event\_t::connection\_failed - when a local device failed to create a control channel between two AVRCP entities.

# 4.147 bt\_avrcp\_evt\_panel\_command\_received\_t Struct Reference

Parameter to AVRCP\_EVT\_PANEL\_COMMAND\_RECEIVED event.

```
#include <avrcp.h>
```

### **Data Fields**

• bt\_avrcp\_channel\_t \* channel

AVRCP channel.

• bt\_byte ctype

Command type.

• bt\_byte button\_status

Button status.

• bt\_byte opcode

Operation Id.

bt\_byte \* params

Operation parameters.

bt\_byte params\_len

Length of the operation parameters.

## 4.147.1 Detailed Description

Parameter to AVRCP\_EVT\_PANEL\_COMMAND\_RECEIVED event.

A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt\_avrcp\_event\_t union - bt\_avrcp\_event\_t::panel\_command\_received - when a local device received a PASS THROUGH command.

# 4.148 bt\_avrcp\_evt\_panel\_response\_received\_t Struct Reference

Parameter to AVRCP\_EVT\_PANEL\_RESPONSE\_RECEIVED event.

#include <avrcp.h>

bt\_avrcp\_channel\_t \* channel

AVRCP channel.

· bt byte ctype

Response type.

• bt\_byte button\_status

Button status.

• bt\_byte opcode

Operation Id.

bt\_byte \* params

Operation parameters.

• bt\_byte params\_len

Length of the operation parameters.

## 4.148.1 Detailed Description

Parameter to AVRCP\_EVT\_PANEL\_RESPONSE\_RECEIVED event.

A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt\_avrcp\_event\_t union - bt\_avrcp\_event\_t::panel\_response\_received - when a local device received a response to a PASS THRO UGH command.

# 4.149 bt\_avrcp\_evt\_register\_events\_completed\_t Struct Reference

### **Data Fields**

 bt\_avrcp\_channel\_t \* channel AVRCP channel.

## 4.150 bt\_avrcp\_evt\_search\_completed\_t Struct Reference

Parameter to AVRCP\_EVT\_SEARCH\_COMPLETED event.

```
#include <avrcp.h>
```

### **Data Fields**

• bt\_avrcp\_device\_t \* devices

list of found devices

• bt\_byte count

the number of found devices

## 4.150.1 Detailed Description

Parameter to AVRCP\_EVT\_SEARCH\_COMPLETED event.

A pointer to this structure is passed to the AVRCP application callback as a valid member of the bt\_avrcp\_event\_t union - bt\_avrcp\_event\_t::device\_search - when searching for nearby devices has finished.

# 4.151 bt\_avrcp\_mgr\_t Struct Reference

## AVRCP manager.

#include <avrcp.h>

### **Data Fields**

· bt\_byte state

Manager state.

bt\_byte flags

Additional manager state.

· bt\_ulong company\_id

The 24-bit unique ID obtained from the IEEE Registration Authority Committee.

• bt\_uint supported\_events

Events supported by the target.

• bt\_ulong song\_length

Current song length.

• bt\_ulong song\_position

Current song poistion.

• bt\_byte play\_status

Current playback status.

• bt\_byte volume

Current volumne.

• bt\_byte battery\_status

Current battery statusThis can be one of the following values:

• bt\_byte system\_status

Current system status.

• bt\_av\_element\_id\_t current\_track\_id

Current track id.

· bt\_uint uid\_counter

The number of media items in the target.

• bt\_avrcp\_channel\_t \* channels

List of available AVRCP channels.

• bt\_avrcp\_mgr\_callback\_fp callback

Pointer to a function used to notify the AVRCP consumer about various events.

void \* callback\_param

 $\textit{Pointer to arbitrary data to be passed to the $\it callback}.$ 

# 4.151.1 Detailed Description

## AVRCP manager.

A structure that glues all pieces together. There is only one instance of this structure allocated by dotstack. A pointer to the instance can be get with  $bt\_avrcp\_get\_mgr()$ .

### 4.151.2 Field Documentation

### 4.151.2.1 bt\_byte battery\_status

Current battery statusThis can be one of the following values:

- AVC\_BATTERY\_STATUS\_NORMAL
- · AVC BATTERY STATUS WARNING
- · AVC BATTERY STATUS CRITICAL
- AVC\_BATTERY\_STATUS\_EXTERNAL
- AVC\_BATTERY\_STATUS\_FULL\_CHARGE

### 4.151.2.2 bt\_ulong company\_id

The 24-bit unique ID obtained from the IEEE Registration Authority Committee.

If the vendor of a TG device does not have the unique ID, the value 0xFFFFFF may be used.

### 4.151.2.3 bt\_byte flags

Additional manager state.

This value can be a combination of the following values:

AVRCP\_MANAGER\_FLAG\_SEARCHING

### 4.151.2.4 bt\_byte state

Manager state.

This value can be one of the following values:

- AVRCP\_MANAGER\_STATE\_IDLE
- AVRCP\_MANAGER\_STATE\_CONNECTING
- AVRCP\_MANAGER\_STATE\_DISCONNECTING

### 4.151.2.5 bt\_uint supported\_events

Events supported by the target.

This value can be a combination of the following values:

- AVC\_EVENT\_FLAG\_PLAYBACK\_STATUS\_CHANGED
- AVC\_EVENT\_FLAG\_TRACK\_CHANGED
- AVC\_EVENT\_FLAG\_TRACK\_REACHED\_END
- AVC\_EVENT\_FLAG\_TRACK\_REACHED\_START
- AVC\_EVENT\_FLAG\_PLAYBACK\_POS\_CHANGED
- AVC\_EVENT\_FLAG\_BATT\_STATUS\_CHANGED

- AVC\_EVENT\_FLAG\_SYSTEM\_STATUS\_CHANGED
- · AVC EVENT FLAG PLAYER APPLICATION SETTING CHANGED
- AVC\_EVENT\_FLAG\_NOW\_PLAYING\_CONTENT\_CHANGED
- · AVC EVENT FLAG AVAILABLE PLAYERS CHANGED
- AVC\_EVENT\_FLAG\_ADDRESSED\_PLAYER\_CHANGED
- AVC\_EVENT\_FLAG\_UIDS\_CHANGED
- AVC\_EVENT\_FLAG\_VOLUME\_CHANGED

# 4.152 bt\_cp\_header\_t Struct Reference

# 4.153 bt\_gatt\_client\_char\_declaration\_t Struct Reference

## Characteristic Declaration.

```
#include <gatt_client.h>
```

### **Data Fields**

• bt\_byte props

Characteristic properties.

bt\_uint start\_handle

First characteristic attribute handle.

• bt\_uint end\_handle

Last characteristic attribute handle.

· bt uint value handle

Characteristic value attribute handle.

• bt\_att\_client\_uuid\_t uuid

Characteristic UUID.

## 4.153.1 Detailed Description

Characteristic Declaration.

This structure is used to hold an characteristic declaration.

# 4.154 bt\_gatt\_client\_char\_descriptor\_t Struct Reference

### Characteristic Descriptor Declaration.

```
#include <gatt_client.h>
```

## **Data Fields**

• bt\_uint handle

Characteristic descriptor attribute handle.

• bt\_att\_client\_uuid\_t uuid

Characteristic descriptor UUID.

## 4.154.1 Detailed Description

Characteristic Descriptor Declaration.

This structure is used to hold an characteristic descriptor declaration.

# 4.155 bt gatt client char value t Struct Reference

Characteristic Value.

```
#include <gatt_client.h>
```

### **Data Fields**

bt uint handle

Characteristic value attribute handle.

• bt\_byte \* value

Characteristic value.

bt uint len

Characteristic value length.

## 4.155.1 Detailed Description

Characteristic Value.

This structure is used to hold an characteristic value.

## 4.156 bt\_gatt\_client\_evt\_conn\_param\_update\_completed\_t Struct Reference

Parameter to GATT\_CLIENT\_EVT\_CONN\_PARAM\_UPDATE\_COMPLETED event.

```
#include <gatt_client.h>
```

## **Data Fields**

• bt\_byte status

Operation status.

· bt\_int hci\_status

HCI command status.

## 4.156.1 Detailed Description

Parameter to GATT CLIENT EVT CONN PARAM UPDATE COMPLETED event.

A pointer to this structure is passed to the GATT client application callback after the new connection parameters have been set.

# 4.157 bt\_gatt\_client\_evt\_conn\_param\_update\_t Struct Reference

Parameter to GATT\_CLIENT\_EVT\_CONN\_PARAM\_UPDATE\_REQUEST event.

```
#include <gatt_client.h>
```

bt\_uint min\_interval

Minimum connection interval expressed in 1.25ms units.

• bt\_uint max\_interval

Maximum connection interval expressed in 1.25ms units.

· bt uint slave latency

Slave latency expressed as number of connection events.

bt\_uint supervision\_timeout

Link supervision timeout expressed in 10ms units.

## 4.157.1 Detailed Description

Parameter to GATT\_CLIENT\_EVT\_CONN\_PARAM\_UPDATE\_REQUEST event.

A pointer to this structure is passed to the GATT client application callback when client received update connection parameters request.

# 4.158 bt\_gatt\_client\_evt\_discover\_all\_chars\_completed\_t Struct Reference

Parameter to GATT\_CLIENT\_EVT\_DISCOVER\_ALL\_CHARS\_COMPLETED event.

```
#include <gatt_client.h>
```

### **Data Fields**

· bt uint count

The number of characteristic declarations found.

bt\_gatt\_client\_char\_declaration\_t \* chars

An array of characteristic declarations.

## 4.158.1 Detailed Description

Parameter to GATT\_CLIENT\_EVT\_DISCOVER\_ALL\_CHARS\_COMPLETED event.

A pointer to this structure is passed to the GATT client application callback when "discover all characteristics of a service" operation has completed.

### 4.158.2 Field Documentation

4.158.2.1 bt\_gatt\_client\_char\_declaration\_t\* chars

An array of characteristic declarations.

This is the same array that is passed to bt\_gatt\_client\_discover\_all\_chars\_ex().

# 4.159 bt\_gatt\_client\_evt\_discover\_all\_services\_completed\_t Struct Reference

Parameter to GATT\_CLIENT\_EVT\_DISCOVER\_ALL\_SERVICES\_COMPLETED event.

```
#include <gatt_client.h>
```

· bt uint count

The number of found services.

bt gatt client service definition t \* services

An array of service definitions.

## 4.159.1 Detailed Description

Parameter to GATT\_CLIENT\_EVT\_DISCOVER\_ALL\_SERVICES\_COMPLETED event.

A pointer to this structure is passed to the GATT client application callback when "discover all services" operation has completed.

### 4.159.2 Field Documentation

```
4.159.2.1 bt_gatt_client_service_definition_t* services
```

An array of service definitions.

This is the same array that is passed to bt\_gatt\_client\_discover\_all\_services\_ex().

# 4.160 bt\_gatt\_client\_evt\_discover\_char\_by\_uuid\_completed\_t Struct Reference

Parameter to GATT\_CLIENT\_EVT\_DISCOVER\_CHAR\_BY\_UUID\_COMPLETED event.

```
#include <gatt_client.h>
```

#### **Data Fields**

· bt uint count

The number of characteristic declarations found.

bt\_gatt\_client\_char\_declaration\_t \* chars

An array of characteristic declarations.

## 4.160.1 Detailed Description

Parameter to GATT\_CLIENT\_EVT\_DISCOVER\_CHAR\_BY\_UUID\_COMPLETED event.

A pointer to this structure is passed to the GATT client application callback when "discover characteristics by UUID" operation has completed.

## 4.160.2 Field Documentation

4.160.2.1 bt\_gatt\_client\_char\_declaration\_t\* chars

An array of characteristic declarations.

This is the same array that is passed to  $bt_gatt_client_discover_char_by_uuid()$  or  $bt_gatt_client_discover_char_by_uuid_80()$ .

# 4.161 bt\_gatt\_client\_evt\_discover\_descriptors\_completed\_t Struct Reference

Parameter to GATT\_CLIENT\_EVT\_DISCOVER\_CHAR\_DESCRIPTORS\_COMPLETED event.

```
#include <gatt_client.h>
```

#### **Data Fields**

· bt\_uint count

The number of characteristic descriptors found.

bt\_gatt\_client\_char\_descriptor\_t \* descriptors

An array of characteristic descriptors.

## 4.161.1 Detailed Description

Parameter to GATT CLIENT EVT DISCOVER CHAR DESCRIPTORS COMPLETED event.

A pointer to this structure is passed to the GATT client application callback when "discover all characteristic descriptors" operation has completed.

### 4.161.2 Field Documentation

4.161.2.1 bt\_gatt\_client\_char\_descriptor\_t\* descriptors

An array of characteristic descriptors.

This is the same array that is passed to bt gatt client discover char descriptors().

## 4.162 bt\_gatt\_client\_evt\_discover\_service\_by\_uuid\_completed\_t Struct Reference

Parameter to GATT\_CLIENT\_EVT\_DISCOVER\_SERVICE\_BY\_UUID\_COMPLETED event.

```
#include <gatt_client.h>
```

## **Data Fields**

· bt uint count

The number of found services.

bt\_gatt\_client\_service\_definition\_t \* services

An array of service definitions.

## 4.162.1 Detailed Description

Parameter to GATT CLIENT EVT DISCOVER SERVICE BY UUID COMPLETED event.

A pointer to this structure is passed to the GATT client application callback when "discover services by UUID" operation has completed.

#### 4.162.2 Field Documentation

4.162.2.1 bt gatt client service definition t\* services

An array of service definitions.

This is the same array that is passed to bt\_gatt\_client\_discover\_by\_service\_uuid().

# 4.163 bt\_gatt\_client\_evt\_exchange\_mtu\_completed\_t Struct Reference

Parameter to GATT CLIENT EVT EXCHANGE MTU COMPLETED event.

```
#include <gatt_client.h>
```

#### **Data Fields**

· bt byte status

Operation status.

• bt uint mtu

Server's MTU.

#### 4.163.1 Detailed Description

Parameter to GATT\_CLIENT\_EVT\_EXCHANGE\_MTU\_COMPLETED event.

A pointer to this structure is passed to the GATT client application callback when the client received a response (either positive or negative) to a "exchange MTU" request.

# 4.164 bt gatt client evt find included completed t Struct Reference

Parameter to GATT\_CLIENT\_EVT\_FIND\_INCLUDED\_SERVICES\_COMPLETED event.

```
#include <gatt_client.h>
```

#### **Data Fields**

· bt\_uint count

The number of included service declarations found.

• bt gatt client inc service declaration t \* services

An array of included service declarations.

# 4.164.1 Detailed Description

 ${\tt Parameter\ to\ GATT\_CLIENT\_EVT\_FIND\_INCLUDED\_SERVICES\_COMPLETED\ event.}$ 

A pointer to this structure is passed to the GATT client application callback when "find included services" operation has completed.

#### 4.164.2 Field Documentation

4.164.2.1 bt\_gatt\_client\_inc\_service\_declaration\_t\* services

An array of included service declarations.

This is the same array that is passed to bt\_gatt\_client\_find\_included\_services\_ex().

# 4.165 bt\_gatt\_client\_evt\_profile\_found\_t Struct Reference

# 4.166 bt\_gatt\_client\_evt\_read\_by\_char\_uuid\_completed\_t Struct Reference

Parameter to GATT CLIENT EVT READ USING CHAR UUID COMPLETED event.

```
#include <gatt_client.h>
```

#### **Data Fields**

• bt\_att\_client\_uuid\_t \* uuid

Characteristic UUID.

· bt uint count

The number of characteristic values read.

• bt\_gatt\_client\_char\_value\_t \* values

An array of characteristic values.

### 4.166.1 Detailed Description

Parameter to GATT\_CLIENT\_EVT\_READ\_USING\_CHAR\_UUID\_COMPLETED event.

A pointer to this structure is passed to the GATT client application callback when "read using characteristic UUID" operation has completed.

#### 4.166.2 Field Documentation

```
4.166.2.1 bt_gatt_client_char_value_t* values
```

An array of characteristic values.

This is the same array that is passed to bt\_gatt\_client\_read\_by\_char\_uuid() or bt\_gatt\_client\_read\_by\_char\_uuid ← \_\_80().

# 4.167 bt\_gatt\_client\_evt\_read\_char\_descriptor\_completed\_t Struct Reference

Parameter to GATT\_CLIENT\_EVT\_READ\_CHAR\_DESCRIPTOR\_COMPLETED event.

```
#include <gatt_client.h>
```

#### **Data Fields**

• bt\_byte status

Operation status.

· const bt byte \* value

Characteristic descriptor. This is the same buffer that is passed to bt\_gatt\_client\_read\_char\_descriptor().

• bt uint len

The length of the characteristic descriptor.

### 4.167.1 Detailed Description

Parameter to GATT\_CLIENT\_EVT\_READ\_CHAR\_DESCRIPTOR\_COMPLETED event.

A pointer to this structure is passed to the GATT client application callback when "read characteristic descriptor" operation has completed.

# 4.168 bt\_gatt\_client\_evt\_read\_char\_long\_descriptor\_completed\_t Struct Reference

Parameter to GATT\_CLIENT\_EVT\_READ\_CHAR\_LONG\_DESCRIPTOR\_COMPLETED event.

```
#include <gatt_client.h>
```

#### **Data Fields**

· bt byte status

Operation status.

· bt uint offset

The offset of the first octet read.

• const bt\_byte \* value

Characteristic descriptor. This is the same buffer that is passed to bt\_gatt\_client\_read\_char\_long\_descriptor().

bt uint len

The length of the characteristic descriptor.

### 4.168.1 Detailed Description

Parameter to GATT CLIENT EVT READ CHAR LONG DESCRIPTOR COMPLETED event.

A pointer to this structure is passed to the GATT client application callback when "read long characteristic descriptor" operation has completed.

### 4.169 bt\_gatt\_client\_evt\_read\_char\_long\_value\_completed\_t Struct Reference

Parameter to GATT\_CLIENT\_EVT\_READ\_CHAR\_LONG\_VALUE\_COMPLETED event.

```
#include <gatt_client.h>
```

# **Data Fields**

bt\_byte status

Operation status.

· bt\_uint offset

The offset of the first octet read.

• const bt\_byte \* value

Characteristic value. This is the same buffer that is passed to bt\_gatt\_client\_read\_char\_long\_value().

bt\_uint len

The length of the characteristic value.

bt\_att\_client\_uuid\_t \* uuid

Characteristic UUID;.

### 4.169.1 Detailed Description

Parameter to GATT\_CLIENT\_EVT\_READ\_CHAR\_LONG\_VALUE\_COMPLETED event.

A pointer to this structure is passed to the GATT client application callback when "read long characteristic value" operation has completed.

# 4.170 bt\_gatt\_client\_evt\_read\_char\_value\_completed\_t Struct Reference

Parameter to GATT\_CLIENT\_EVT\_READ\_CHAR\_VALUE\_COMPLETED event.

```
#include <gatt_client.h>
```

#### **Data Fields**

· bt\_byte status

Operation status.

• const bt\_byte \* value

Characteristic value. This is the same buffer that is passed to bt\_gatt\_client\_read\_char\_value().

• bt\_uint len

The length of the characteristic value.

• bt\_att\_client\_uuid\_t \* uuid

Characteristic UUID;.

### 4.170.1 Detailed Description

Parameter to GATT CLIENT EVT READ CHAR VALUE COMPLETED event.

A pointer to this structure is passed to the GATT client application callback when "read characteristic value" operation has completed.

### 4.171 bt\_gatt\_client\_evt\_read\_multiple\_char\_values\_completed\_t Struct Reference

Parameter to GATT\_CLIENT\_EVT\_READ\_MULTIPLE\_CHAR\_VALUES\_COMPLETED event.

```
#include <gatt_client.h>
```

# **Data Fields**

· bt\_byte status

Operation status.

const bt\_byte \* value

Characteristic values. This is the same buffer that is passed to bt\_gatt\_client\_read\_multiple\_char\_values().

bt\_uint len

The length of the characteristic values.

### 4.171.1 Detailed Description

Parameter to GATT\_CLIENT\_EVT\_READ\_MULTIPLE\_CHAR\_VALUES\_COMPLETED event.

A pointer to this structure is passed to the GATT client application callback when "read multiple characteristic values" operation has completed.

# 4.172 bt\_gatt\_client\_evt\_t Union Reference

Parameter to GATT client application callback.

#include <gatt\_client.h>

#### **Data Fields**

- bt\_gatt\_client\_evt\_exchange\_mtu\_completed\_t exchange\_mtu
   Valid if event is GATT\_CLIENT\_EVT\_EXCHANGE\_MTU\_COMPLETED.
- bt\_gatt\_client\_evt\_discover\_all\_services\_completed\_t discover\_all\_services
   Valid if event is GATT\_CLIENT\_EVT\_DISCOVER\_ALL\_SERVICES\_COMPLETED.
- bt\_gatt\_client\_evt\_discover\_service\_by\_uuid\_completed\_t discover\_service\_by\_uuid Valid if event is GATT\_CLIENT\_EVT\_DISCOVER\_SERVICE\_BY\_UUID\_COMPLETED.
- bt\_gatt\_client\_evt\_discover\_all\_chars\_completed\_t discover\_all\_chars

  Valid if event is GATT\_CLIENT\_EVT\_DISCOVER\_ALL\_CHARS\_COMPLETED.
- bt\_gatt\_client\_evt\_discover\_char\_by\_uuid\_completed\_t discover\_char\_by\_uuid
   Valid if event is GATT\_CLIENT\_EVT\_DISCOVER\_CHAR\_BY\_UUID\_COMPLETED.
- bt\_gatt\_client\_evt\_read\_char\_value\_completed\_t read\_char\_value
   Valid if event is GATT\_CLIENT\_EVT\_READ\_CHAR\_VALUE\_COMPLETED.
- bt\_gatt\_client\_evt\_read\_char\_long\_value\_completed\_t read\_char\_long\_value
   Valid if event is GATT\_CLIENT\_EVT\_READ\_CHAR\_LONG\_VALUE\_COMPLETED.
- bt\_gatt\_client\_evt\_write\_char\_value\_completed\_t write\_char\_value
   Valid if event is GATT\_CLIENT\_EVT\_WRITE\_CHAR\_VALUE\_COMPLETED.
- bt\_gatt\_client\_evt\_write\_char\_long\_value\_completed\_t write\_char\_long\_value
   Valid if event is GATT\_CLIENT\_EVT\_WRITE\_CHAR\_LONG\_VALUE\_COMPLETED.
- bt\_gatt\_client\_evt\_value\_notification\_t value\_notification
   Valid if event is GATT\_CLIENT\_EVT\_VALUE\_NOTIFICATION.
- bt\_gatt\_client\_evt\_discover\_descriptors\_completed\_t discover\_descriptors
   Valid if event is GATT\_CLIENT\_EVT\_DISCOVER\_CHAR\_DESCRIPTORS\_COMPLETED.
- bt\_gatt\_client\_evt\_read\_char\_descriptor\_completed\_t read\_descriptor
   Valid if event is GATT\_CLIENT\_EVT\_READ\_CHAR\_DESCRIPTOR\_COMPLETED.
- bt\_gatt\_client\_evt\_read\_char\_long\_descriptor\_completed\_t read\_long\_descriptor
   Valid if event is GATT\_CLIENT\_EVT\_READ\_CHAR\_LONG\_DESCRIPTOR\_COMPLETED.
- bt\_gatt\_client\_evt\_write\_char\_descriptor\_completed\_t write\_descriptor Valid if event is GATT\_CLIENT\_EVT\_WRITE\_CHAR\_DESCRIPTOR\_COMPLETED.
- bt\_gatt\_client\_evt\_write\_char\_long\_descriptor\_completed\_t write\_long\_descriptor
   Valid if event is GATT\_CLIENT\_EVT\_WRITE\_CHAR\_LONG\_DESCRIPTOR\_COMPLETED.
- bt\_gatt\_client\_evt\_find\_included\_completed\_t find\_included
  - Valid if event is GATT\_CLIENT\_EVT\_FIND\_INCLUDED\_SERVICES\_COMPLETED.
- bt\_gatt\_client\_evt\_read\_by\_char\_uuid\_completed\_t read\_by\_char\_uuid
   Valid if event is GATT\_CLIENT\_EVT\_READ\_USING\_CHAR\_UUID\_COMPLETED.
- bt\_gatt\_client\_evt\_read\_multiple\_char\_values\_completed\_t read\_multiple\_char\_values Valid if event is GATT\_CLIENT\_EVT\_READ\_MULTIPLE\_CHAR\_VALUES\_COMPLETED.
- bt\_gatt\_client\_evt\_conn\_param\_update\_t conn\_param\_update
  - Valid if event is GATT\_CLIENT\_EVT\_CONN\_PARAM\_UPDATE\_REQUEST.
- bt\_gatt\_client\_evt\_conn\_param\_update\_completed\_t conn\_param\_update\_completed
   Valid if event is GATT\_CLIENT\_EVT\_CONN\_PARAM\_UPDATE\_COMPLETED.
- bt\_gatt\_client\_evt\_profile\_found\_t profile\_found

Valid if event is GATT\_CLIENT\_EVT\_PROFILE\_FOUND.

### 4.172.1 Detailed Description

Parameter to GATT client application callback.

This union is used to pass event specific data to the GATT client consumer. Which member of the union points to a valid structure depends on the event reported to the consumer. In general, each event has a corresponding member in the union.

# 4.173 bt\_gatt\_client\_evt\_value\_notification\_t Struct Reference

Parameter to GATT\_CLIENT\_EVT\_VALUE\_NOTIFICATION event.

```
#include <gatt_client.h>
```

#### **Data Fields**

· bt uint handle

Characteristic value attribute handle.

• const bt\_byte \* value

Characteristic value.

bt uint len

Characteristic value length.

· bt\_bool indication

If non-0 a value indication has been received.

# 4.173.1 Detailed Description

Parameter to GATT\_CLIENT\_EVT\_VALUE\_NOTIFICATION event.

A pointer to this structure is passed to the GATT client application callback when the client received a value notification or indication.

# 4.174 bt\_gatt\_client\_evt\_write\_char\_descriptor\_completed\_t Struct Reference

Parameter to GATT\_CLIENT\_EVT\_WRITE\_CHAR\_DESCRIPTOR\_COMPLETED event.

```
#include <gatt_client.h>
```

#### **Data Fields**

bt\_byte status

Operation status.

• bt\_uint handle

Characteristic descriptor attribute handle.

### 4.174.1 Detailed Description

Parameter to GATT\_CLIENT\_EVT\_WRITE\_CHAR\_DESCRIPTOR\_COMPLETED event.

A pointer to this structure is passed to the GATT client application callback when "write characteristic descriptor" operation has completed.

# 4.175 bt\_gatt\_client\_evt\_write\_char\_long\_descriptor\_completed\_t Struct Reference

Parameter to GATT\_CLIENT\_EVT\_WRITE\_CHAR\_LONG\_DESCRIPTOR\_COMPLETED event.

```
#include <gatt_client.h>
```

#### **Data Fields**

• bt\_byte status

Operation status.

### 4.175.1 Detailed Description

Parameter to GATT CLIENT EVT WRITE CHAR LONG DESCRIPTOR COMPLETED event.

A pointer to this structure is passed to the GATT client application callback when "write long characteristic descriptor" operation has completed.

# 4.176 bt\_gatt\_client\_evt\_write\_char\_long\_value\_completed\_t Struct Reference

Parameter to GATT\_CLIENT\_EVT\_WRITE\_CHAR\_LONG\_VALUE\_COMPLETED event.

```
#include <gatt_client.h>
```

#### **Data Fields**

· bt byte status

Operation status.

### 4.176.1 Detailed Description

Parameter to GATT\_CLIENT\_EVT\_WRITE\_CHAR\_LONG\_VALUE\_COMPLETED event.

A pointer to this structure is passed to the GATT client application callback when "write long characteristic value" operation has completed.

# 4.177 bt\_gatt\_client\_evt\_write\_char\_value\_completed\_t Struct Reference

Parameter to GATT\_CLIENT\_EVT\_WRITE\_CHAR\_VALUE\_COMPLETED event.

```
#include <gatt_client.h>
```

# **Data Fields**

· bt\_byte status

Operation status.

bt\_uint handle

Characteristic value attribute handle.

### 4.177.1 Detailed Description

Parameter to GATT\_CLIENT\_EVT\_WRITE\_CHAR\_VALUE\_COMPLETED event.

A pointer to this structure is passed to the GATT client application callback when "write characteristic value" or "write without response" operation has completed.

# 4.178 bt\_gatt\_client\_inc\_service\_declaration\_t Struct Reference

Included Service Declaration.

```
#include <gatt_client.h>
```

#### **Data Fields**

· bt uint uuid

Service UUID.

· bt uint decl handle

Included service declaration attribute handle.

bt\_uint start\_handle

First service attribute handle.

• bt uint end handle

Last service attribute handle.

### 4.178.1 Detailed Description

Included Service Declaration.

This structure is used to hold an include service declaration.

- 4.179 bt\_gatt\_client\_listener\_t Struct Reference
- 4.180 bt\_gatt\_client\_mgr\_t Struct Reference
- 4.181 bt\_gatt\_client\_profile\_finder\_state\_t Struct Reference
- 4.182 bt\_gatt\_client\_profile\_finder\_t Struct Reference
- 4.183 bt gatt client profile header t Struct Reference
- 4.184 bt\_gatt\_client\_service\_definition\_t Struct Reference

Service Definition.

#include <gatt\_client.h>

#### **Data Fields**

bt\_att\_client\_uuid\_t uuid

Service UUID.

bt\_uint start\_handle

First attribute handle.

· bt uint end handle

Last attribute handle.

# 4.184.1 Detailed Description

Service Definition.

This structure is used to hold a service definition.

# 4.185 bt\_gatt\_client\_session\_t Struct Reference

# 4.186 bt\_gatt\_evt\_client\_config\_changed\_t Struct Reference

Parameter to GATT\_SERVER\_EVT\_CLIENT\_CONFIG\_CHANGED event.

```
#include <gatt_server.h>
```

#### **Data Fields**

• bt\_gatt\_evt\_header\_t header

Common header.

• bt\_byte client\_config

New client configuration.

### 4.186.1 Detailed Description

Parameter to GATT\_SERVER\_EVT\_CLIENT\_CONFIG\_CHANGED event.

A pointer to this structure is passed to the GATT application callback when the characteristic's client configuration changed.

# 4.187 bt\_gatt\_evt\_ext\_properties\_changed\_t Struct Reference

Parameter to GATT SERVER EVT EXTENDED PROPERTIES CHANGED event.

```
#include <gatt_server.h>
```

#### **Data Fields**

bt\_gatt\_evt\_header\_t header

Common header.

bt\_byte ext\_properties

New extended properties.

### 4.187.1 Detailed Description

Parameter to GATT\_SERVER\_EVT\_EXTENDED\_PROPERTIES\_CHANGED event.

A pointer to this structure is passed to the GATT application callback when the characteristic's extended properties changed.

# 4.188 bt\_gatt\_evt\_header\_t Struct Reference

Common to all event parameters header.

```
#include <gatt_server.h>
```

#### **Data Fields**

• bt\_uuid\_t service\_type

128-bit service type UUID.

• bt\_uuid\_t service\_id

128-bit service id UUID.

· bt\_uuid\_t characteristic\_id

128-bit characteristic id UUID.

### 4.188.1 Detailed Description

Common to all event parameters header.

This structure holds characteristic's identifier - service type, service id, and characteristic id.

# 4.189 bt\_gatt\_evt\_server\_config\_changed\_t Struct Reference

Parameter to GATT\_SERVER\_EVT\_SERVER\_CONFIG\_CHANGED event.

```
#include <gatt_server.h>
```

### **Data Fields**

· bt\_gatt\_evt\_header\_t header

Common header.

• bt\_byte server\_config

New server configuration.

### 4.189.1 Detailed Description

Parameter to GATT\_SERVER\_EVT\_SERVER\_CONFIG\_CHANGED event.

A pointer to this structure is passed to the GATT application callback when the characteristic's server configuration changed.

# 4.190 bt\_gatt\_evt\_value\_changed\_t Struct Reference

Parameter to GATT\_SERVER\_EVT\_VALUE\_CHANGED event.

```
#include <gatt_server.h>
```

#### **Data Fields**

· bt\_gatt\_evt\_header\_t header

Common header.

· const bt byte \* value

New value.

bt\_uint len

New value length.

### 4.190.1 Detailed Description

Parameter to GATT\_SERVER\_EVT\_VALUE\_CHANGED event.

A pointer to this structure is passed to the GATT application callback when the characteristic's value changed.

# 4.191 bt\_gatt\_evt\_value\_read\_t Struct Reference

Parameter to GATT\_SERVER\_EVT\_VALUE\_READ event.

```
#include <gatt_server.h>
```

### **Data Fields**

· bt\_gatt\_evt\_header\_t header

Common header.

• const bt\_byte \* value

Value.

• bt uint len

Value length.

bt\_uint offset

Offset from which the value has been read.

### 4.191.1 Detailed Description

Parameter to GATT\_SERVER\_EVT\_VALUE\_READ event.

A pointer to this structure is passed to the GATT application callback when the characteristic's value has been read by a client.

# 4.192 bt\_gatt\_listener\_t Struct Reference

# 4.193 bt\_hci\_command\_p Struct Reference

- 4.194 bt\_hci\_data\_p Struct Reference
- 4.195 bt\_hci\_event\_p Struct Reference
- 4.196 bt\_hci\_inquiry\_response\_t Struct Reference
- 4.197 bt hf subscriber number t Struct Reference

#### **Data Fields**

• bt\_char number [HFP\_MAX\_NUMBER\_LENGTH+1]

Phone number.

• bt\_byte type

Number type.

· bt\_byte service

Service type.

- 4.198 bt\_hfp\_ag\_evt\_search\_completed\_t Struct Reference
- 4.199 bt\_hfp\_audio\_packet\_t Struct Reference

Parameter to HFP\_EVENT\_AUDIO\_DATA\_RECEIVED event.

#include <hfp.h>

### **Data Fields**

bt\_uint data\_len

The length of the audio data.

bt\_uint max\_data\_len

Maximum length of the media data that can be stored in the buffer pointed by the data member.

• bt byte \* data

Pointer to a buffer to store audio data.

### 4.199.1 Detailed Description

Parameter to HFP\_EVENT\_AUDIO\_DATA\_RECEIVED event.

HFP layer calls the session callback when it has received a chunk of audio data from the remote device. The callback is passed a pointer to bt\_hfp\_evt\_audio\_data\_t which contains a pointer to this structure.

#### 4.199.2 Field Documentation

4.199.2.1 bt\_byte\* data

Pointer to a buffer to store audio data.

This pointer must be allocated by the HFP consumer before adding this structure to receive (::bt\_hfp\_add\_audio ← \_rx\_buffer) or send queue (::bt\_hfp\_add\_audio\_tx\_buffer).

# 4.200 bt\_hfp\_call\_t Struct Reference

Stores information about a call.

#include <hfp.h>

#### **Data Fields**

• bt\_byte index

Call index inside AG.

bt\_byte direction

Call direction. HFP\_CALL\_DIR\_OUTGOING or HFP\_CALL\_DIR\_INCOMING.

· bt\_byte status

Call status.

· bt byte mode

Call mode.

• bt\_byte multiparty

Number or parties in the call.

• bt\_char number [HFP\_MAX\_NUMBER\_LENGTH+1]

Phone number.

• bt\_int type

Number type.

# 4.200.1 Detailed Description

Stores information about a call.

#### 4.200.2 Field Documentation

4.200.2.1 bt\_byte mode

Call mode.

- HFP\_CALL\_MODE\_VOICE
- HFP\_CALL\_MODE\_DATA
- HFP\_CALL\_MODE\_FAX

4.200.2.2 bt\_byte multiparty

Number or parties in the call.

- HFP\_CALL\_MPTY\_NOT\_MULTIPARTY
- HFP\_CALL\_MPTY\_MULTIPARTY

4.200.2.3 bt\_byte status

Call status.

- HFP\_CALL\_STATUS\_MASK\_IDLE
- HFP\_CALL\_STATUS\_MASK\_ACTIVE
- HFP\_CALL\_STATUS\_MASK\_HELD
- HFP\_CALL\_STATUS\_MASK\_DIALING
- HFP\_CALL\_STATUS\_MASK\_ALERTING
- HFP\_CALL\_STATUS\_MASK\_INCOMING
- · HFP CALL STATUS MASK WAITING

# 4.201 bt\_hfp\_event\_register\_t Struct Reference

Stores value of HF registrations.

#include <hfp.h>

#### **Data Fields**

• bt\_byte mode

The registration mode. Only HFP\_REG\_MODE\_FORWARD is supported by the current BT spec.

· bt\_byte ind

The indicator's value. The possible values are defined by HFP\_REG\_IND\_ constants.

### 4.201.1 Detailed Description

Stores value of HF registrations.

- 4.202 bt\_hfp\_evt\_activate\_voice\_recognition\_t Struct Reference
- 4.203 bt\_hfp\_evt\_answer\_call\_t Struct Reference
- 4.204 bt\_hfp\_evt\_audio\_connection\_state\_changed\_t Struct Reference

HFP\_EVENT\_AUDIO\_CONNECTION\_STATE\_CHANGED event parameter.

#include <hfp.h>

### **Data Fields**

bt\_bool connected

Defines the state of audio connection. TRUE - if audio is connected, FALSE - if audio is disconnected.

bt\_hci\_conn\_state\_t \* hci\_conn

A pointer to HCI SCO connection if connected is TRUE, otherwise - NULL.

#### 4.204.1 Detailed Description

HFP\_EVENT\_AUDIO\_CONNECTION\_STATE\_CHANGED event parameter.

HFP layer calls the session callback when the state of the audio connection has changed and passes a pointer to bt\_hfp\_evt\_audio\_connection\_state\_changed\_t which contains a flag identifying the current state of the audio connection (connected or disconnected).

# 4.205 bt\_hfp\_evt\_audio\_data\_t Struct Reference

HFP\_EVENT\_AUDIO\_DATA\_RECEIVED event parameter.

#include <hfp.h>

#### **Data Fields**

• bt\_hfp\_audio\_packet\_t \* packet

A pointer to an audio packet buffer.

### 4.205.1 Detailed Description

HFP EVENT AUDIO DATA RECEIVED event parameter.

HFP layer calls the session callback when it received a chunk of audio data from the remote device and passes a pointer to bt\_hfp\_evt\_audio\_data\_t which contains a pointer to the data and its length.

### 4.206 bt\_hfp\_evt\_audio\_packet\_sent\_t Struct Reference

# 4.207 bt\_hfp\_evt\_call\_waiting\_t Struct Reference

HFP\_EVENT\_CALL\_WAITING event parameter.

#include <hfp.h>

#### **Data Fields**

• bt\_char \* number

Pointer to 0-terminated string that contains the number of the incoming call.

bt\_ulong number\_type

The type of the number.

# 4.207.1 Detailed Description

HFP\_EVENT\_CALL\_WAITING event parameter.

HFP layer calls the session callback when it received the number of a waiting call (another incoming call while there is already ongoing call) from the AG and passes a pointer to bt\_hfp\_evt\_call\_waiting\_t which contains the number of the waiting call.

# 4.208 bt\_hfp\_evt\_chld\_t Struct Reference

# 4.209 bt\_hfp\_evt\_clip\_received\_t Struct Reference

HFP\_EVENT\_CLIP\_RECEIVED event parameter.

```
#include <hfp.h>
```

#### **Data Fields**

• bt char \* number

Pointer to 0-terminated string that contains the number of the incoming call.

• bt\_ulong number\_type

The type of the number.

#### 4.209.1 Detailed Description

HFP\_EVENT\_CLIP\_RECEIVED event parameter.

HFP layer calls the session callback when it received the number of an incoming call from the AG and passes a pointer to bt\_hfp\_evt\_clip\_received\_t which contains the number of the incoming call.

# 4.210 bt\_hfp\_evt\_command\_completed\_t Struct Reference

HFP\_EVENT\_CMD\_COMPLETED event parameter.

```
#include <hfp.h>
```

#### **Data Fields**

• bt\_hfp\_cmd\_id\_enum id

The command ID.

bt\_bool success

TRUE - if command succeeded, FALSE - if command failed.

### 4.210.1 Detailed Description

HFP\_EVENT\_CMD\_COMPLETED event parameter.

HFP layer calls the session callback when a command sent by HF to AG has been processed by the AG and a result (success or error) has been received. the callback is passed a pointer to bt\_hfp\_evt\_command\_completed\_t which contains the id of the command and result.

# 4.211 bt\_hfp\_evt\_dial\_request\_received\_t Struct Reference

HFP\_AG\_EVENT\_DIAL\_REQUEST\_RECEIVED event parameter.

```
#include <hfp.h>
```

#### **Data Fields**

• bt hfp dial request enum request type

The dial request type.

• bt\_char number [HFP\_MAX\_NUMBER\_LENGTH+1]

0-terminated string that contains the number to dial or memory location

bt\_int memory\_location

memory location to dial

#### 4.211.1 Detailed Description

HFP\_AG\_EVENT\_DIAL\_REQUEST\_RECEIVED event parameter.

HFP layer calls the session callback when it received a command from HF device with instruction to initiate a voice call. The callback is passed a pointer to \_bt\_hfp\_evt\_dial\_request\_received\_t which contains the type of the call, including place a call with a phone number supplied by HF, memory dial, or last number redial.

When application is able to place a call it should call HFP layer with corresponding call information.

# 4.212 bt\_hfp\_evt\_inband\_ring\_changed\_t Struct Reference

HFP\_EVENT\_INBAND\_RING\_CHANGED event parameter.

#include <hfp.h>

#### **Data Fields**

• bt bool enabled

TRUE - if in-band ringing is enabled, FALSE - otherwise.

### 4.212.1 Detailed Description

HFP\_EVENT\_INBAND\_RING\_CHANGED event parameter.

HFP layer calls the session callback when in-band ringing in the AG has been turned on or off the callback is passed a pointer to bt\_hfp\_evt\_inband\_ring\_changed\_t which contains the current status of in-band ringing in the AG.

if in-band ringing is enabled the AG will provide ringing tone. However the HF may ignore this tone and generate its own upon receiving the HFP\_EVENT\_RING event.

if in-band ringing is disabled the HF has to generate a ringing tone on its own. upon receiving the HFP\_EVENT\_← RING event.

# 4.213 bt\_hfp\_evt\_indicator\_received\_t Struct Reference

HFP\_EVENT\_INDICATOR\_RECEIVED event parameter.

#include <hfp.h>

#### **Data Fields**

• bt byte id

Indicator ID. The possible values are defined by HFP\_IND\_ constants.

• bt\_byte value

The value of the indicator.

### 4.213.1 Detailed Description

HFP EVENT INDICATOR RECEIVED event parameter.

HFP layer calls the session callback when it received indicator from the AG and passes a pointer to  $bt\_hfp\_\leftarrow evt\_indicator\_received\_t$  which contains the id of the indicator and its value.

# 4.214 bt\_hfp\_evt\_mic\_volume\_changed\_t Struct Reference

HFP\_EVENT\_MIC\_VOLUME\_CHANGED event parameter.

```
#include <hfp.h>
```

#### **Data Fields**

• bt byte volume

The value of the microphone gain.

### 4.214.1 Detailed Description

HFP\_EVENT\_MIC\_VOLUME\_CHANGED event parameter.

HFP layer calls the session callback when the microphone gain has been changed in the AG and passes a pointer to bt\_hfp\_evt\_mic\_volume\_changed\_t which contains the new value for the microphone gain.

### 4.215 bt\_hfp\_evt\_query\_operator\_completed\_t Struct Reference

HFP EVENT QUERY OPERATOR COMPLETED event parameter.

```
#include <hfp.h>
```

#### **Data Fields**

bt\_byte mode

Operator's mode.

• bt char \* name

Pointer to 0-terminated string that contains the name of the operator.

### 4.215.1 Detailed Description

HFP\_EVENT\_QUERY\_OPERATOR\_COMPLETED event parameter.

HFP layer calls the session callback when it received operator's name from the AG during SLC setup or as a result of calling  $bt_hfp_hf_query_operator$  and passes a pointer to  $bt_hfp_evt_query_operator_completed_t$  which contains the name of the operator.

# 4.216 bt\_hfp\_evt\_send\_dtmf\_code\_t Struct Reference

# 4.217 bt\_hfp\_evt\_slc\_connection\_state\_changed\_t Struct Reference

HFP\_EVENT\_SLC\_CONNECTION\_STATE\_CHANGED event parameter.

```
#include <hfp.h>
```

#### **Data Fields**

· bt bool connected

Defines the state of SLC. TRUE - if SLC is connected, FALSE - if SLC is disconnected.

#### 4.217.1 Detailed Description

HFP EVENT SLC CONNECTION STATE CHANGED event parameter.

HFP layer calls the session callback when the state of the service level connection (SLC) has changed and passes a pointer to bt\_hfp\_evt\_slc\_connection\_state\_changed\_t which contains a flag identifying the current state of SLC (connected or disconnected).

# 4.218 bt\_hfp\_evt\_spk\_volume\_changed\_t Struct Reference

HFP EVENT SPK VOLUME CHANGED event parameter.

```
#include <hfp.h>
```

#### **Data Fields**

• bt\_byte volume

The value of the speaker volume.

# 4.218.1 Detailed Description

HFP EVENT SPK VOLUME CHANGED event parameter.

HFP layer calls the session callback when the speaker volume has been changed in the AG and passes a pointer to bt\_hfp\_evt\_spk\_volume\_changed\_t which contains the new value for the speaker volume.

# 4.219 bt\_hfp\_evt\_terminate\_call\_t Struct Reference

# 4.220 bt\_hfp\_evt\_voice\_recognition\_changed\_t Struct Reference

HFP EVENT VOICE RECOGNITION CHANGED event parameter.

```
#include <hfp.h>
```

### **Data Fields**

· bt bool enabled

TRUE - if voice dialing is enabled, FALSE - otherwise.

### 4.220.1 Detailed Description

HFP\_EVENT\_VOICE\_RECOGNITION\_CHANGED event parameter.

HFP layer calls the session callback when voice dialing in the AG has been turned on or off as a result of calling bt\_hfp\_hf\_enable\_voice\_recognition. the callback is passed a pointer to bt\_hfp\_evt\_voice \_\_recognition\_changed\_t which contains the current status of voice dialing in the AG.

# 4.221 bt\_hfp\_hf\_t Struct Reference

# 4.222 bt\_hfp\_ind Struct Reference

Stores value of an indicator.

```
#include <hfp.h>
```

#### **Data Fields**

· bt byte val

The indicator's value.

bt\_byte id

HF indicator ID. The possible values are defined by HFP\_IND\_ constants.

• bt\_byte enabled

AG indicator enabled. If 0 indicator has been disabled by HF device.

### 4.222.1 Detailed Description

Stores value of an indicator.

- 4.223 bt\_hfp\_session Struct Reference
- 4.224 bt\_hid\_pdu\_t Struct Reference
- 4.225 bt\_hid\_report\_t Struct Reference
- 4.226 bt\_hid\_session\_t Struct Reference
- 4.227 bt\_media\_packet\_t Struct Reference

Media packet buffer.

```
#include <avdtp.h>
```

### **Data Fields**

- bt\_media\_packet\_t \* next\_packet
  - Pointer to next buffer.
- bt\_byte version

Version of the RTP implementation.

· bt\_byte csrc\_count

The CSRC count contains the number of CSRC identifiers that follow the fixed header.

· bt bool marker

The interpretation of the marker is defined by a profile.

bt\_byte payload\_type

This field identifies the format of the RTP payload and determines its interpretation by the application.

bt\_uint seq\_number

The sequence number increments by one for each media packet sent, and may be used by the receiver to detect packet loss and to restore packet sequence.

· bt ulong timestamp

The Time Stamp reflects the sampling instant of the first octet in the media packet.

• bt\_ulong ssrc

The SSRC field identifies the synchronization source.

bt\_ulong \* csrc\_list

The CSRC list identifies the contributing sources for the payload contained in this packet.

• bt byte \* data

Pointer to a buffer to store media data.

• bt int data len

Lentgth of the media data.

bt\_int max\_data\_len

Maximum length of the media data that can be stored in the buffer pointed by the data member.

bt cp header t cp header

Content protection header.

#### 4.227.1 Detailed Description

Media packet buffer.

This structure is used to receive and send media packet from/to the remote device. See more information about usage of this structure in descriptions of bt\_avdtp\_add\_media\_rx\_buffer and bt\_avdtp\_add\_media\_tx\_buffer.

#### 4.227.2 Field Documentation

4.227.2.1 bt\_byte\* data

Pointer to a buffer to store media data.

This pointer must be allocated by the AVDTP consumer before adding this structure to receive (bt\_avdtp\_add\_\top media\_rx\_buffer) or send queue (bt\_avdtp\_add\_media\_tx\_buffer).

4.227.2.2 bt\_bool marker

The interpretation of the marker is defined by a profile.

It is intended to allow significant events such as frame boundaries to be marked in the packet stream.

4.227.2.3 bt\_byte payload\_type

This field identifies the format of the RTP payload and determines its interpretation by the application.

A profile specifies default static mapping of payload type codes to payload formats.

### 4.227.2.4 bt\_ulong ssrc

The SSRC field identifies the synchronization source.

This identifier is chosen randomly, with the intent that no two synchronization sources, within the same media transport session, shall have the same SSRC identifier.

1.228	bt_sdp_client_evt_connected_t Struct Reference
1.229	bt_sdp_client_evt_disconnected_t Struct Reference
4.230	bt_sdp_data_element_t Struct Reference
4.231	bt_sdp_found_attr_list_t Struct Reference
4.232	bt_sdp_sequence_t Struct Reference
1.233	bt_sdp_serialization_state_p Struct Reference
1.234	bt_sdp_service_transaction_p Struct Reference
4.235	bt_sdp_transaction_t Struct Reference
1.236	bt_security_mgr_t Struct Reference
1.237	bt_sm_event_t Union Reference
1.238	bt_sm_evt_connected_t Struct Reference
1.239	bt_sm_evt_disconnected_t Struct Reference
1.240	bt_sm_evt_generate_lkt_t Struct Reference
1.241	bt_sm_evt_ltk_generated_t Struct Reference
1.242	bt_sm_evt_master_csrk_notif_t Struct Reference
1.243	bt_sm_evt_master_irk_notif_t Struct Reference
1.244	bt_sm_evt_master_ltk_notif_t Struct Reference

- 4.245 bt\_sm\_evt\_master\_ltk\_request\_t Struct Reference
- 4.246 bt\_sm\_evt\_oob\_data\_request\_t Struct Reference
- 4.247 bt\_sm\_evt\_pairing\_complete\_t Struct Reference
- 4.248 bt\_sm\_evt\_pairing\_request\_t Struct Reference
- 4.249 bt\_sm\_evt\_passkey\_notification\_t Struct Reference
- 4.250 bt\_sm\_evt\_passkey\_request\_t Struct Reference
- 4.251 bt\_sm\_evt\_peer\_keys\_notif\_t Struct Reference
- 4.252 bt\_sm\_evt\_slave\_ltk\_request\_t Struct Reference
- 4.253 bt\_sm\_listener\_t Struct Reference
- 4.254 bt\_sm\_ltk\_t Struct Reference
- 4.255 bt\_sm\_pairing\_features\_t Struct Reference
- 4.256 bt sm session t Struct Reference
- 4.257 bt\_spp\_port\_t Struct Reference

#### Serial port structure.

```
#include <spp.h>
```

#### **Data Structures**

• struct \_bt\_spp\_port\_flags\_t

### **Data Fields**

bt\_byte state
 Port state.

# 4.257.1 Detailed Description

### Serial port structure.

This structure represents a Bluetooth serial port. Application code may only use those fields that are documented. The rest of the fields are private to the SPP implementation.

# 4.257.2 Field Documentation

4.257.2.1 bt\_byte state

Port state.

The field is set to one of the values defined in the bt\_spp\_port\_state\_e enumeration. This field must never be modified by the application.

4.258	btx_csr_autobaud_buffer_t Struct Reference
4.259	btx_csr_bccmd_header_t Struct Reference
4.260	btx_csr_bccmd_listener_t Struct Reference
4.261	btx_csr_cached_temperature_t Struct Reference
4.262	btx_csr_create_operator_c_t Struct Reference
4.263	btx_csr_exec_hq_script_buffer_t Struct Reference
4.264	btx_csr_exec_script_buffer_t Struct Reference
4.265	btx_csr_pio_direction_mask_t Struct Reference
4.266	btx_csr_pio_protection_mask_t Struct Reference
4.267	btx_csr_pio_t Struct Reference
4.268	btx_csr_rssi_acl_t Struct Reference
4.269	btx_csr_script_t Struct Reference
4.270	btx_csr_set_ps_vars_buffer_t Struct Reference
4.271	btx_csr_strm_connect_t Struct Reference
4.272	btx_csr_strm_get_sink_t Struct Reference
4.273	btx_csr_strm_get_source_t Struct Reference
4.274	btx_csr_var_t Union Reference

# Index

_bt_a2dp_mgr_t	bt_att_client_allocate_session, 202
state, 256	bt_att_client_connect, 202
_bt_att_find_info_response_s	bt_att_client_disconnect, 203
format, 269	bt_att_client_exchange_mtu, 203
_bt_avctp_mgr_t	bt_att_client_execute_write, 203
flags, 291	bt_att_client_find_by_type_value, 204
state, 291	bt_att_client_find_info, 204
_bt_avdtp_sep_t	bt_att_client_free_session, 205
state, 317	bt att client get remote address, 205
type, 317	bt att client init, 205
_bt_avrcp_mgr_t	bt_att_client_prepare_write, 205
battery_status, 327	bt_att_client_read, 206
company_id, 327	bt_att_client_read_blob, 206
flags, 327	bt_att_client_read_by_group_type, 206
state, 327	bt_att_client_read_by_group_type_80, 207
supported_events, 327	bt_att_client_read_by_type, 207
_bt_hfp_audio_packet_s	bt_att_client_read_by_type_80, 208
data, 344	
bt hfp call s	bt_att_client_read_multiple, 208
mode, 345	bt_att_client_reject_conn_param_update, 208
multiparty, 345	bt_att_client_signed_write, 209
status, 345	bt_att_client_write, 209
_bt_media_packet_t	bt_att_client_write_cmd, 209
data, 353	bt_att_get_client_mgr, 210
marker, 353	ATT Server, 189
payload_type, 353	bt_att_get_mgr, 192
ssrc, 353	bt_att_get_remote_address, 192
_bt_spp_port_t	bt_att_init, 192
state, 356	bt_att_mgr_callback_fp, 191
	bt_att_server_authorize_access, 192
A2DP_EVT_STREAM_ABORTED	bt_att_server_disconnect, 193
Advanced Audio Distribution Profile (A2DP), 117	bt_att_server_indicate, 193
A2DP_EVT_STREAM_CLOSED	bt_att_server_indicate_to_all, 193
Advanced Audio Distribution Profile (A2DP), 117	bt_att_server_notify, 194
A2DP EVT STREAM CONFIGURED	bt_att_server_register_callback, 194
Advanced Audio Distribution Profile (A2DP), 117	bt_att_server_start, 194
A2DP_EVT_STREAM_OPENED	bt_att_server_stop, 195
Advanced Audio Distribution Profile (A2DP), 117	bt_att_server_write_attribute_value, 195
A2DP_EVT_STREAM_RECONFIGURED	bt_att_update_conn_parameters, 196
Advanced Audio Distribution Profile (A2DP), 117	ATT_CLIENT_MAX_SESSIONS
A2DP_EVT_STREAM_STARTED	Configuration, 211
Advanced Audio Distribution Profile (A2DP), 117	ATT_ERROR_UNLIKELY_ERROR
A2DP_EVT_STREAM_SUSPENDED	ATT, 188
Advanced Audio Distribution Profile (A2DP), 117	ATT_ERROR_UNSUPPORTED_GROUP_TYPE
ATT, 187	ATT, 188
ATT_ERROR_UNLIKELY_ERROR, 188	ATT_MAX_CLIENTS
ATT_ERROR_UNSUPPORTED_GROUP_TYPE,	Configuration, 197
188	ATT_MAX_FOUND_ATTRIBUTES
ATT Client, 199	Configuration, 197
bt_att_client_accept_conn_param_update, 202	ATT_MAX_QUEUED_WRITE_BUFFER_SIZE

Configuration, 198	A2DP_EVT_STREAM_ABORTED, 117
AVCTP_ALLOCATE_BUFFERS_VARS	A2DP_EVT_STREAM_CLOSED, 117
Configuration, 147	A2DP_EVT_STREAM_CONFIGURED, 117
AVCTP_MAX_CHANNELS	A2DP_EVT_STREAM_OPENED, 117
Configuration, 148	A2DP_EVT_STREAM_RECONFIGURED, 117
AVCTP_MAX_TRANSPORT_CHANNELS	A2DP_EVT_STREAM_STARTED, 117
Configuration, 148	A2DP_EVT_STREAM_SUSPENDED, 117
AVDTP_CODEC_CONFIG_BUFFER_LEN	bt_a2dp_abort_stream, 117
Configuration, 110	bt_a2dp_add_media_rx_buffer, 118
AVDTP_EVT_STREAM_ABORTED	bt_a2dp_add_media_tx_buffer, 118
Audio/Video Distribution Protocol (AVDP), 86	bt_a2dp_cancel_listen, 119
AVDTP_EVT_STREAM_CLOSED	bt_a2dp_close_stream, 119
Audio/Video Distribution Protocol (AVDP), 86	bt_a2dp_connect, 119
AVDTP_EVT_STREAM_CONFIGURED	bt_a2dp_connect_ex, 121
Audio/Video Distribution Protocol (AVDP), 86	bt_a2dp_create_stream, 121
AVDTP_EVT_STREAM_OPENED	bt_a2dp_destroy_stream, 121
Audio/Video Distribution Protocol (AVDP), 86	bt_a2dp_disconnect, 122
AVDTP_EVT_STREAM_RECONFIGURED	bt_a2dp_discover, 122
Audio/Video Distribution Protocol (AVDP), 86	bt_a2dp_find_codec, 123
AVDTP_EVT_STREAM_STARTED	bt_a2dp_find_server_callback_fp, 131
Audio/Video Distribution Protocol (AVDP), 87	bt a2dp find sink, 133
AVDTP_EVT_STREAM_SUSPENDED	bt_a2dp_find_source, 133
Audio/Video Distribution Protocol (AVDP), 87	bt_a2dp_get_all_capabilities, 123
AVDTP MAX CMD BUFFERS	bt_a2dp_get_capabilities, 124
Configuration, 110	bt_a2dp_get_hci_connection, 124
AVDTP_MAX_CMD_PARAM_LEN	bt_a2dp_get_mgr, 134
Configuration, 110	bt_a2dp_get_stream_codec_config, 125
AVDTP_MAX_REMOTE_DEVICES	bt_a2dp_get_stream_codec_type, 125
Configuration, 110	bt_a2dp_get_stream_local_sep_id, 126
AVDTP_MAX_SEP	bt_a2dp_get_stream_remote_address, 126
Configuration, 110	bt_a2dp_get_stream_remote_sep_id, 126
AVDTP_MAX_STREAMS	bt_a2dp_get_stream_state, 127
Configuration, 110	bt_a2dp_init, 134
AVDTP_MAX_TRANSPORT_CHANNELS	bt_a2dp_listen, 127
Configuration, 110	bt_a2dp_mgr_callback_fp, 131
AVDTP_MAX_TX_BUFFER_LEN	bt_a2dp_open_and_start_stream, 134
Configuration, 111	bt_a2dp_reconfigure_stream, 128
AVDTP_STREAM_STATE_ABORTING	bt_a2dp_register_aac_codec, 135
Audio/Video Distribution Protocol (AVDP), 87	bt_a2dp_register_callback, 135
AVDTP STREAM STATE CLOSING	bt_a2dp_register_mpeg_codec, 137
Audio/Video Distribution Protocol (AVDP), 87	bt_a2dp_register_sink, 128
AVDTP_STREAM_STATE_IDLE	bt_a2dp_register_source, 128
Audio/Video Distribution Protocol (AVDP), 87	bt_a2dp_remove_media_rx_buffer, 129
AVDTP STREAM STATE STREAMING	bt a2dp remove media tx buffer, 129
Audio/Video Distribution Protocol (AVDP), 87	bt_a2dp_start, 137
AVRCP_CMD_TIMEOUT	bt_a2dp_start_stream, 130
Configuration, 186	bt_a2dp_suspend_stream, 130
AVRCP MAX CHANNELS	bt_avdtp_get_stream_config, 130
Configuration, 186	attributes
AVRCP MAX CMD BUFFERS	bt_av_get_element_attributes_t, 274
Configuration, 186	Audio/Video Control Transport Protocol (AVCTP), 138
AVRCP_MAX_CMD_PARAM_LEN	bt_avctp_cancel_command, 141
Configuration, 186	bt_avctp_cancel_listen, 141
AVRCP_MAX_DEVICE_NAME_LEN	bt_avctp_cancel_response, 141
Configuration, 186	bt_avctp_connect, 141
AVRCP_MAX_SEARCH_RESULTS	bt_avctp_create_channel, 142
Configuration, 186	bt_avctp_create_outgoing_channel, 142
Advanced Audio Distribution Profile (A2DP), 112	bt_avctp_destroy_channel, 143
V // //	

bt_avctp_disconnect, 143	bt_avdtp_remove_media_tx_buffer, 104
bt_avctp_get_channel_remote_address, 143	bt_avdtp_report_delay, 104
bt_avctp_get_channel_state, 143	bt_avdtp_security_control, 105
bt_avctp_get_hci_connection, 144	bt_avdtp_set_configuration, 105
bt_avctp_get_mgr, 144	bt_avdtp_start, 106
bt_avctp_init, 144	bt_avdtp_start_stream, 106
bt_avctp_listen, 144	bt_avdtp_suspend_stream, 106
bt_avctp_mgr_callback_fp, 140	bt_avdtp_unregister_codec, 107
bt_avctp_send_command, 145	Audio/Video Remote Control Profile (AVRCP), 149
bt_avctp_send_response, 145	bt_avrcp_add_to_now_playing, 162
bt_avctp_set_callback, 145	bt_avrcp_cancel_find, 162
Audio/Video Distribution Protocol (AVDP), 77	bt_avrcp_cancel_listen, 162
AVDTP_EVT_STREAM_ABORTED, 86	bt_avrcp_connect, 162
AVDTP EVT STREAM CLOSED, 86	bt_avrcp_create_channel, 163
AVDTP EVT STREAM CONFIGURED, 86	bt_avrcp_create_outgoing_channel, 163
AVDTP_EVT_STREAM_OPENED, 86	bt_avrcp_destroy_channel, 164
AVDTP_EVT_STREAM_RECONFIGURED, 86	bt_avrcp_disconnect, 164
AVDTP EVT STREAM STARTED, 87	bt_avrcp_find_callback_fp, 160
AVDTP EVT STREAM SUSPENDED, 87	bt_avrcp_find_controller, 164
AVDTP_STREAM_STATE_ABORTING, 87	bt_avrcp_find_target, 165
AVDTP_STREAM_STATE_CLOSING, 87	bt_avrcp_find_targets, 166
AVDTP_STREAM_STATE_IDLE, 87	bt_avrcp_get_browsing_channel_state, 166
AVDTP_STREAM_STATE_STREAMING, 87	bt_avrcp_get_channel_remote_address, 166
bt_avdtp_abort_stream, 91	bt_avrcp_get_company_id_list, 167
bt_avdtp_abort_stream, 91 bt_avdtp_add_media_rx_buffer, 91	bt_avrcp_get_control_channel_state, 167
bt_avdtp_add_media_tx_buffer, 91	bt_avrcp_get_current_player_application_setting
bt_avdtp_cancel_listen, 92	_value, 167
bt_avdtp_close_stream, 92	bt_avrcp_get_element_attributes, 168
bt_avdtp_close_stream, 92 bt_avdtp_codec_handler_fp, 88	
bt_avdtp_conect, 87	bt_avrcp_get_hci_connection, 169 bt_avrcp_get_mgr, 169
bt_avdtp_connect_ex, 88	_ ,
bt_avdtp_connect_ex, 88 bt_avdtp_create_stream, 93	bt_avrcp_get_play_status, 169
bt_avdtp_create_stream, 93 bt_avdtp_destroy_stream, 93	bt_avrcp_get_player_application_setting_attr_text, 169
bt_avdtp_disconnect, 93 bt avdtp discover, 94	bt_avrcp_get_player_application_setting_value_←
— ·— ·	text, 171
bt_avdtp_find_codec, 94	bt_avrcp_get_subuint_info, 171
bt_avdtp_get_all_capabilities, 95	bt_avrcp_get_supported_event_id_list, 171
bt_avdtp_get_capabilities, 95	bt_avrcp_get_unit_info, 172
bt_avdtp_get_configuration, 96	bt_avrcp_inform_battery_status, 172
bt_avdtp_get_hci_connection, 96	bt_avrcp_inform_displayable_character_set, 173
bt_avdtp_get_mgr, 97	bt_avrcp_init_controller, 173
bt_avdtp_get_sep, 97	bt_avrcp_init_target, 173
bt_avdtp_get_stream_codec_config, 97	bt_avrcp_list_player_application_setting_attributes,
bt_avdtp_get_stream_codec_type, 98	174
bt_avdtp_get_stream_config, 98	bt_avrcp_list_player_application_setting_values,
bt_avdtp_get_stream_local_sep_id, 98	174
bt_avdtp_get_stream_remote_address, 99	bt_avrcp_listen, 176
bt_avdtp_get_stream_remote_sep_id, 99	bt_avrcp_mgr_callback_fp, 160
bt_avdtp_get_stream_state, 99	bt_avrcp_play_item, 176
bt_avdtp_init, 100	bt_avrcp_register_notification, 176
bt_avdtp_listen, 100	bt_avrcp_register_notifications, 178
bt_avdtp_mgr_callback_fp, 89	bt_avrcp_send_button_click, 179
bt_avdtp_open_stream, 100	bt_avrcp_send_cmd, 179
bt_avdtp_reconfigure_stream, 101	bt_avrcp_send_panel_control, 179
bt_avdtp_register_callback, 101	bt_avrcp_send_press_panel_control, 180
bt_avdtp_register_codec, 103	bt_avrcp_send_release_panel_control, 180
bt_avdtp_register_sep, 103	bt_avrcp_send_simple_panel_cmd, 180
bt_avdtp_remove_media_rx_buffer, 104	bt_avrcp_set_absolute_volume, 181

bt_avrcp_set_addressed_player, 181 bt_avrcp_set_player_application_setting_value,	bt_a2dp_get_stream_codec_config Advanced Audio Distribution Profile (A2DP), 125
181	bt_a2dp_get_stream_codec_type
bt_avrcp_start, 182	Advanced Audio Distribution Profile (A2DP), 125
bt_avrcp_tg_send_element_attributes, 182	bt_a2dp_get_stream_local_sep_id
bt_avrcp_tg_set_absolute_volume, 182	Advanced Audio Distribution Profile (A2DP), 126
bt_avrcp_tg_set_battery_status, 183	bt_a2dp_get_stream_remote_address
bt_avrcp_tg_set_current_track, 183	Advanced Audio Distribution Profile (A2DP), 126
bt_avrcp_tg_set_play_status, 183	bt_a2dp_get_stream_remote_sep_id
bt_avrcp_tg_set_system_status, 184	Advanced Audio Distribution Profile (A2DP), 126
	bt_a2dp_get_stream_state
BEGIN_DE_SEQUENCE	Advanced Audio Distribution Profile (A2DP), 127
SDP, 45	bt_a2dp_init
battery_status	Advanced Audio Distribution Profile (A2DP), 134
_bt_avrcp_mgr_t, 327	bt_a2dp_listen
bt_a2dp_aac_config_t, 253	Advanced Audio Distribution Profile (A2DP), 127
bt_a2dp_abort_stream	bt_a2dp_mgr_callback_fp
Advanced Audio Distribution Profile (A2DP), 117	Advanced Audio Distribution Profile (A2DP), 131
bt_a2dp_add_media_rx_buffer	bt_a2dp_mgr_t, 255
Advanced Audio Distribution Profile (A2DP), 118	bt a2dp mpeg config t, 256
bt_a2dp_add_media_tx_buffer	bt_a2dp_open_and_start_stream
Advanced Audio Distribution Profile (A2DP), 118	Advanced Audio Distribution Profile (A2DP), 134
bt_a2dp_cancel_listen	bt_a2dp_reconfigure_stream
Advanced Audio Distribution Profile (A2DP), 119	Advanced Audio Distribution Profile (A2DP), 128
bt_a2dp_close_stream  Advanced Audia Distribution Profile (A2DD) 110	bt_a2dp_register_aac_codec
Advanced Audio Distribution Profile (A2DP), 119	Advanced Audio Distribution Profile (A2DP), 135
bt_a2dp_connect  Advanced Audia Distribution Profile (A2DP) 110	bt_a2dp_register_callback
Advanced Audio Distribution Profile (A2DP), 119	Advanced Audio Distribution Profile (A2DP), 135
bt_a2dp_connect_ex  Advanced Audia Distribution Profile (A2DP) 121	bt_a2dp_register_mpeg_codec
Advanced Audio Distribution Profile (A2DP), 121 bt_a2dp_create_stream	Advanced Audio Distribution Profile (A2DP), 137
Advanced Audio Distribution Profile (A2DP), 121	bt_a2dp_register_sink
bt_a2dp_destroy_stream	Advanced Audio Distribution Profile (A2DP), 128
Advanced Audio Distribution Profile (A2DP), 121	bt a2dp register source
bt_a2dp_disconnect	Advanced Audio Distribution Profile (A2DP), 128
Advanced Audio Distribution Profile (A2DP), 122	bt_a2dp_remove_media_rx_buffer
bt_a2dp_discover	Advanced Audio Distribution Profile (A2DP), 129
Advanced Audio Distribution Profile (A2DP), 122	bt_a2dp_remove_media_tx_buffer
bt_a2dp_event_t, 253	Advanced Audio Distribution Profile (A2DP), 129
bt_a2dp_evt_open_and_start_stream_completed_	bt_a2dp_sbc_config_t, 256
t, 254	bt_a2dp_sbc_packet_info_t, 256
err_code, 255	bt_a2dp_start
failed cmd, 255	Advanced Audio Distribution Profile (A2DP), 137
bt_a2dp_find_codec	bt_a2dp_start_stream
Advanced Audio Distribution Profile (A2DP), 123	Advanced Audio Distribution Profile (A2DP), 130
bt_a2dp_find_server_callback_fp	bt_a2dp_suspend_stream
Advanced Audio Distribution Profile (A2DP), 131	Advanced Audio Distribution Profile (A2DP), 130
bt_a2dp_find_sink	bt_att_attribute_t, 256
Advanced Audio Distribution Profile (A2DP), 133	bt_att_client_accept_conn_param_update
bt_a2dp_find_source	ATT Client, 202
Advanced Audio Distribution Profile (A2DP), 133	bt_att_client_allocate_session
bt_a2dp_get_all_capabilities	ATT Client, 202
Advanced Audio Distribution Profile (A2DP), 123	bt_att_client_connect
bt_a2dp_get_capabilities	ATT Client, 202
Advanced Audio Distribution Profile (A2DP), 124	bt_att_client_disconnect
bt_a2dp_get_hci_connection	ATT Client, 203
Advanced Audio Distribution Profile (A2DP), 124	bt_att_client_evt_conn_param_update_completed_t,
bt_a2dp_get_mgr	256
Advanced Audio Distribution Profile (A2DP), 134	bt_att_client_evt_conn_param_update_t, 257

bt_att_client_evt_execute_write_response_t, 257	bt_att_evt_attr_notification_sent_t, 265
bt_att_client_evt_find_by_type_value_response_t, 258	bt_att_evt_attr_value_changed_t, 265
bt_att_client_evt_info_response_t, 258	bt_att_evt_attr_value_read_t, 266
bt_att_client_evt_mtu_response_t, 258	bt_att_evt_authorization_requested_t, 266
bt_att_client_evt_prepare_write_response_t, 259	bt_att_evt_data_received_t, 267
bt_att_client_evt_read_blob_response_t, 259	bt_att_evt_session_connected_t, 267
bt_att_client_evt_read_by_group_type_response_t, 260	bt_att_evt_session_disconnected_t, 267
bt_att_client_evt_read_by_type_response_t, 260	bt_att_evt_tran_timeout_t, 267
bt_att_client_evt_read_multiple_response_t, 261	bt_att_find_by_type_value_response_t, 268
bt_att_client_evt_read_response_t, 261	bt_att_find_info_response_t, 268
bt att client evt t, 262	bt att found attribyte t, 269
bt_att_client_evt_value_indication_t, 262	bt_att_get_client_mgr
bt_att_client_evt_value_notification_t, 263	ATT Client, 210
bt_att_client_evt_write_response_t, 263	bt_att_get_mgr
bt_att_client_exchange_mtu	ATT Server, 192
ATT Client, 203	bt_att_get_remote_address
bt att client execute write	ATT Server, 192
ATT Client, 203	bt_att_init
bt_att_client_find_by_type_value	ATT Server, 192
ATT Client, 204	bt_att_listener_t, 269
bt_att_client_find_info	bt_att_mgr_callback_fp
ATT Client, 204	ATT Server, 191
bt_att_client_free_session	
	bt_att_mgr_t, 269
ATT Client, 205	bt_att_read_by_group_type_response_t, 269
bt_att_client_get_remote_address	bt_att_read_by_type_response_t, 270
ATT Client, 205	bt_att_server_authorize_access
bt_att_client_init	ATT Server, 192
ATT Client, 205	bt_att_server_disconnect
bt_att_client_mgr_t, 264	ATT Server, 193
bt_att_client_prepare_write	bt_att_server_indicate
ATT Client, 205	ATT Server, 193
bt_att_client_read	bt_att_server_indicate_to_all
ATT Client, 206	ATT Server, 193
bt_att_client_read_blob	bt_att_server_notify
ATT Client, 206	ATT Server, 194
bt_att_client_read_by_group_type	bt_att_server_register_callback
ATT Client, 206	ATT Server, 194
bt_att_client_read_by_group_type_80	bt_att_server_start
ATT Client, 207	ATT Server, 194
bt_att_client_read_by_type	bt_att_server_stop
ATT Client, 207	ATT Server, 195
bt_att_client_read_by_type_80	bt_att_server_write_attribute_value
ATT Client, 208	ATT Server, 195
bt_att_client_read_multiple	bt_att_session_evt_authorization_requested_t, 270
ATT Client, 208	bt_att_session_evt_packet_sent_t, 270
bt_att_client_reject_conn_param_update	bt_att_session_t, 270
ATT Client, 208	bt_att_update_conn_parameters
bt_att_client_session_t, 264	ATT Server, 196
bt_att_client_signed_write	bt_av_add_to_now_playing_t, 270
ATT Client, 209	bt_av_battery_status_of_ct_t, 271
bt_att_client_uuid_t, 264	status, 271
len, 264	bt_av_capability_company_id_t, 271
bt_att_client_write	bt_av_capability_event_id_t, 272
ATT Client, 209	bt_av_command_t, 272
bt_att_client_write_cmd	bt_av_displayable_character_set_t, 272
ATT Client, 209	bt_av_element_attribute_t, 273
bt_att_enum_context_t, 264	bt_av_element_attributes_t, 273
bt_att_evt_attr_indication_sent_t, 264	bt_av_element_id_t, 274

bt_av_get_element_attributes_t, 274 attributes, 274	bt_avctp_evt_command_received_t, 288 bt_avctp_evt_command_sent_t, 288
bt_av_notification_addressed_player_changed_t, 275	bt_avctp_evt_connection_failed_t, 289
bt_av_notification_app_setting_changed_t, 275	bt_avctp_evt_response_cancelled_t, 289
bt_av_notification_battery_status_t, 275	bt_avctp_evt_response_received_t, 290
status, 276	bt_avctp_evt_response_sent_t, 290
bt_av_notification_playback_pos_changed_t, 276	bt_avctp_get_channel_remote_address
bt_av_notification_playback_status_changed_t, 276	Audio/Video Control Transport Protocol (AVCTP),
status, 277	143
bt_av_notification_system_status_changed_t, 277	bt_avctp_get_channel_state
bt_av_notification_t, 277	Audio/Video Control Transport Protocol (AVCTP),
bt_av_notification_track_changed_t, 279	143
bt_av_notification_uids_changed_t, 279	bt_avctp_get_hci_connection
bt_av_notification_volume_changed_t, 280	Audio/Video Control Transport Protocol (AVCTP),
bt_av_play_item_t, 280	144
bt_av_play_status_t, 280	bt_avctp_get_mgr
play status, 281	Audio/Video Control Transport Protocol (AVCTP),
bt_av_player_setting_current_values_t, 281	144
bt_av_player_setting_values_t, 282	bt_avctp_init
bt_av_player_setting_values_text_t, 282	Audio/Video Control Transport Protocol (AVCTP),
bt_av_player_settings_t, 282	144
bt_av_player_settings_text_t, 283	bt_avctp_listen
bt_av_player_text_t, 283	Audio/Video Control Transport Protocol (AVCTP),
bt_av_register_notification_t, 283	144
event_id, 284	bt_avctp_message_t, 290
bt_av_response_t, 284	bt_avctp_mgr_callback_fp
ctype, 285	Audio/Video Control Transport Protocol (AVCTP),
bt_av_set_absolute_volume_t, 285	140
bt_av_set_addressed_player_t, 286	bt_avctp_mgr_t, 291
bt_avctp_cancel_command	bt_avctp_send_command
Audio/Video Control Transport Protocol (AVCTP),	Audio/Video Control Transport Protocol (AVCTP), 145
bt avctp cancel listen	bt_avctp_send_response
Audio/Video Control Transport Protocol (AVCTP),	Audio/Video Control Transport Protocol (AVCTP),
141	145
bt_avctp_cancel_response	bt_avctp_set_callback
Audio/Video Control Transport Protocol (AVCTP),	Audio/Video Control Transport Protocol (AVCTP),
141	145
bt_avctp_channel_t, 286	bt_avctp_transport_t, 291
bt_avctp_connect	bt_avdtp_abort_stream
Audio/Video Control Transport Protocol (AVCTP),	Audio/Video Distribution Protocol (AVDP), 91
141	bt_avdtp_add_media_rx_buffer
bt_avctp_create_channel	Audio/Video Distribution Protocol (AVDP), 91
Audio/Video Control Transport Protocol (AVCTP),	bt_avdtp_add_media_tx_buffer
142	Audio/Video Distribution Protocol (AVDP), 91
bt_avctp_create_outgoing_channel	bt_avdtp_cancel_listen
Audio/Video Control Transport Protocol (AVCTP),	Audio/Video Distribution Protocol (AVDP), 92
142	bt_avdtp_close_stream
bt_avctp_destroy_channel	Audio/Video Distribution Protocol (AVDP), 92
Audio/Video Control Transport Protocol (AVCTP),	bt_avdtp_codec_handler_fp
143	Audio/Video Distribution Protocol (AVDP), 88
bt_avctp_disconnect	bt_avdtp_codec_op_decode_t, 292
Audio/Video Control Transport Protocol (AVCTP),	bt_avdtp_codec_op_encode_t, 292
143	bt_avdtp_codec_op_param_t, 292
bt_avctp_event_t, 286	bt_avdtp_codec_op_parse_config_t, 292
bt_avctp_evt_channel_connected_t, 287	codec_config, 293
bt_avctp_evt_channel_disconnected_t, 287	offset, 293
bt_avctp_evt_command_cancelled_t, 288	bt_avdtp_codec_op_parse_packet_t, 293

bt_avdtp_codec_op_serialize_config_t, 293	bt_avdtp_evt_stream_configured_t, 308
codec_config, 294	bt_avdtp_evt_stream_opened_t, 309
offset, 294	bt_avdtp_evt_stream_reconfigure_completed_t, 309
bt_avdtp_codec_t, 294	err_code, 309
codec_type, 294	bt_avdtp_evt_stream_reconfigured_t, 310
bt_avdtp_connect	bt_avdtp_evt_stream_security_control_completed_
Audio/Video Distribution Protocol (AVDP), 87	t, 310
bt_avdtp_connect_ex	err_code, 310
Audio/Video Distribution Protocol (AVDP), 88	bt_avdtp_evt_stream_started_t, 310
bt_avdtp_create_stream	bt_avdtp_evt_stream_suspended_t, 311
Audio/Video Distribution Protocol (AVDP), 93	bt_avdtp_evt_suspend_stream_completed_t, 311
bt_avdtp_destroy_stream	err_code, 312
Audio/Video Distribution Protocol (AVDP), 93	bt_avdtp_evt_suspend_stream_requested_t, 312
bt_avdtp_disconnect Audio/Video Distribution Protocol (AVDP), 93	err_code, 312 bt_avdtp_find_codec
	Audio/Video Distribution Protocol (AVDP), 94
bt_avdtp_discover Audio/Video Distribution Protocol (AVDP), 94	bt_avdtp_get_all_capabilities
bt_avdtp_event_t, 295	Audio/Video Distribution Protocol (AVDP), 95
bt_avdtp_evt_abort_stream_requested_t, 296	bt_avdtp_get_capabilities
err_code, 297	Audio/Video Distribution Protocol (AVDP), 95
bt_avdtp_evt_close_stream_completed_t, 297	bt_avdtp_get_configuration
err_code, 297	Audio/Video Distribution Protocol (AVDP), 96
bt_avdtp_evt_close_stream_requested_t, 297	bt_avdtp_get_hci_connection
err code, 298	Audio/Video Distribution Protocol (AVDP), 96
bt_avdtp_evt_ctrl_channel_connected_t, 298	bt_avdtp_get_mgr
bt_avdtp_evt_ctrl_channel_disconnected_t, 298	Audio/Video Distribution Protocol (AVDP), 97
bt_avdtp_evt_delay_report_completed_t, 299	bt_avdtp_get_sep
err_code, 299	Audio/Video Distribution Protocol (AVDP), 97
bt_avdtp_evt_discover_completed_t, 299	bt_avdtp_get_stream_codec_config
err_code, 299	Audio/Video Distribution Protocol (AVDP), 97
bt_avdtp_evt_get_sep_capabilities_completed_t, 300	bt_avdtp_get_stream_codec_type
err_code, 300	Audio/Video Distribution Protocol (AVDP), 98
bt_avdtp_evt_get_stream_configuration_completed_t,	bt_avdtp_get_stream_config
300	Advanced Audio Distribution Profile (A2DP), 130
bt_avdtp_evt_media_packet_received_t, 301	Audio/Video Distribution Protocol (AVDP), 98
bt_avdtp_evt_media_packet_send_failed_t, 301	bt_avdtp_get_stream_local_sep_id
bt_avdtp_evt_media_packet_sent_t, 301	Audio/Video Distribution Protocol (AVDP), 98
bt_avdtp_evt_open_stream_completed_t, 302	bt_avdtp_get_stream_remote_address
err_code, 302	Audio/Video Distribution Protocol (AVDP), 99
bt_avdtp_evt_open_stream_requested_t, 302	bt_avdtp_get_stream_remote_sep_id
err_code, 303	Audio/Video Distribution Protocol (AVDP), 99
bt_avdtp_evt_reconfigure_stream_requested_t, 303	bt_avdtp_get_stream_state
err_code, 304	Audio/Video Distribution Protocol (AVDP), 99
bt_avdtp_evt_sep_capabilities_received_t, 304	bt_avdtp_init
bt_avdtp_evt_sep_info_received_t, 304	Audio/Video Distribution Protocol (AVDP), 100
media_type, 305	bt_avdtp_listen
bt_avdtp_evt_set_stream_configuration_completed_t,	Audio/Video Distribution Protocol (AVDP), 100
305	bt_avdtp_mgr_callback_fp
err_code, 305	Audio/Video Distribution Protocol (AVDP), 89
bt_avdtp_evt_set_stream_configuration_requested_t,	bt_avdtp_mgr_t, 312
306	flags, 313
err_code, 306	state, 313
bt_avdtp_evt_start_stream_completed_t, 306	bt_avdtp_open_stream
err_code, 307	Audio/Video Distribution Protocol (AVDP), 100
bt_avdtp_evt_start_stream_requested_t, 307	bt_avdtp_reconfigure_stream
err_code, 307	Audio/Video Distribution Protocol (AVDP), 101
bt_avdtp_evt_stream_aborted_t, 307 bt_avdtp_evt_stream_closed_t, 308	bt_avdtp_register_callback Audio/Video Distribution Protocol (AVDP), 101
Di_avaip_evi_stream_closed_t, 500	Addio/ video Distribution Fibitoto (AvDi <sup>*</sup> ), 101

bt_avdtp_register_codec Audio/Video Distribution Protocol (AVDP), 103	Audio/Video Remote Control Profile (AVRCP), 164 bt_avrcp_device_t, 321
bt avdtp register sep	bt_avrcp_disconnect
Audio/Video Distribution Protocol (AVDP), 103	Audio/Video Remote Control Profile (AVRCP), 164
bt_avdtp_remove_media_rx_buffer	bt_avrcp_event_t, 321
Audio/Video Distribution Protocol (AVDP), 104	notification, 322
bt_avdtp_remove_media_tx_buffer	bt_avrcp_evt_channel_connected_t, 323
Audio/Video Distribution Protocol (AVDP), 104	bt_avrcp_evt_channel_disconnected_t, 323
bt_avdtp_report_delay	bt_avrcp_evt_connection_failed_t, 323
Audio/Video Distribution Protocol (AVDP), 104	bt_avrcp_evt_panel_command_received_t, 324
•	
bt_avdtp_security_control	bt_avrcp_evt_panel_response_received_t, 324
Audio/Video Distribution Protocol (AVDP), 105	bt_avrcp_evt_register_events_completed_t, 325
bt_avdtp_sep_capabilities_t, 314	bt_avrcp_evt_search_completed_t, 325
categories, 315	bt_avrcp_find_callback_fp
codec_type, 315	Audio/Video Remote Control Profile (AVRCP), 160
cp_specific, 315	bt_avrcp_find_controller
cp_specific_len, 315	Audio/Video Remote Control Profile (AVRCP), 164
cp_type, 315	bt_avrcp_find_target
header_compression_options, 315	Audio/Video Remote Control Profile (AVRCP), 165
max_parity_code_packets, 316	bt_avrcp_find_targets
max_recovery_window, 316	Audio/Video Remote Control Profile (AVRCP), 166
media_type, 316	bt_avrcp_get_browsing_channel_state
multiplexing_options, 316	Audio/Video Remote Control Profile (AVRCP), 166
recovery_type, 316	bt_avrcp_get_channel_remote_address
bt_avdtp_sep_t, 317	Audio/Video Remote Control Profile (AVRCP), 166
bt_avdtp_set_configuration	bt_avrcp_get_company_id_list
Audio/Video Distribution Protocol (AVDP), 105	Audio/Video Remote Control Profile (AVRCP), 167
bt_avdtp_start	bt_avrcp_get_control_channel_state
Audio/Video Distribution Protocol (AVDP), 106	Audio/Video Remote Control Profile (AVRCP), 167
bt_avdtp_start_stream	bt_avrcp_get_current_player_application_setting_value
Audio/Video Distribution Protocol (AVDP), 106	Audio/Video Remote Control Profile (AVRCP), 167
bt_avdtp_stream_t, 317	bt_avrcp_get_element_attributes
flags, 318	Audio/Video Remote Control Profile (AVRCP), 168
sessions, 318	bt_avrcp_get_hci_connection
state, 319	Audio/Video Remote Control Profile (AVRCP), 169
bt_avdtp_suspend_stream	bt_avrcp_get_mgr
Audio/Video Distribution Protocol (AVDP), 106	Audio/Video Remote Control Profile (AVRCP), 169
bt_avdtp_transport_channel_t, 319	bt_avrcp_get_play_status
ref_count, 320	Audio/Video Remote Control Profile (AVRCP), 169
type, 320	bt_avrcp_get_player_application_setting_attr_text
bt_avdtp_transport_session_t, 320	Audio/Video Remote Control Profile (AVRCP), 169
type, 320	bt_avrcp_get_player_application_setting_value_text
bt_avdtp_unregister_codec	Audio/Video Remote Control Profile (AVRCP), 171
Audio/Video Distribution Protocol (AVDP), 107	bt_avrcp_get_subuint_info
bt_avrcp_add_to_now_playing	Audio/Video Remote Control Profile (AVRCP), 171
Audio/Video Remote Control Profile (AVRCP), 162	bt_avrcp_get_supported_event_id_list
bt_avrcp_cancel_find	Audio/Video Remote Control Profile (AVRCP), 171
Audio/Video Remote Control Profile (AVRCP), 162	bt_avrcp_get_unit_info
bt_avrcp_cancel_listen	Audio/Video Remote Control Profile (AVRCP), 172
Audio/Video Remote Control Profile (AVRCP), 162	bt_avrcp_inform_battery_status
bt_avrcp_channel_t, 321	Audio/Video Remote Control Profile (AVRCP), 172
bt_avrcp_connect	bt_avrcp_inform_displayable_character_set
Audio/Video Remote Control Profile (AVRCP), 162	Audio/Video Remote Control Profile (AVRCP), 173
bt_avrcp_create_channel	bt_avrcp_init_controller
Audio/Video Remote Control Profile (AVRCP), 163	Audio/Video Remote Control Profile (AVRCP), 173
bt_avrcp_create_outgoing_channel	bt_avrcp_init_target
Audio/Video Remote Control Profile (AVRCP), 163	Audio/Video Remote Control Profile (AVRCP), 173
bt_avrcp_destroy_channel	bt_avrcp_list_player_application_setting_attributes

Audio/Video Remote Control Profile (AVRCP), 174 bt_avrcp_list_player_application_setting_values	bt_gatt_client_discover_all_chars_ex GATT Client, 228
Audio/Video Remote Control Profile (AVRCP), 174 bt_avrcp_listen	bt_gatt_client_discover_all_services_ex GATT Client, 228
Audio/Video Remote Control Profile (AVRCP), 176	bt_gatt_client_discover_by_service_uuid
	GATT Client, 228
bt_avrcp_mgr_callback_fp	•
Audio/Video Remote Control Profile (AVRCP), 160	bt_gatt_client_discover_by_service_uuid_80
bt_avrcp_mgr_t, 326	GATT Client, 229
bt_avrcp_play_item	bt_gatt_client_discover_char_by_uuid
Audio/Video Remote Control Profile (AVRCP), 176	GATT Client, 229
bt_avrcp_register_notification	bt_gatt_client_discover_char_by_uuid_80
Audio/Video Remote Control Profile (AVRCP), 176	GATT Client, 229
bt_avrcp_register_notifications	bt_gatt_client_discover_char_descriptors
Audio/Video Remote Control Profile (AVRCP), 178	GATT Client, 230
bt_avrcp_send_button_click	bt_gatt_client_evt_conn_param_update_completed_t,
Audio/Video Remote Control Profile (AVRCP), 179	329
bt_avrcp_send_cmd	bt_gatt_client_evt_conn_param_update_t, 329
Audio/Video Remote Control Profile (AVRCP), 179	bt gatt client evt discover all chars completed ←
bt_avrcp_send_panel_control	t, 330
Audio/Video Remote Control Profile (AVRCP), 179	chars, 330
bt_avrcp_send_press_panel_control	bt_gatt_client_evt_discover_all_services_completed_t,
Audio/Video Remote Control Profile (AVRCP), 180	330
bt_avrcp_send_release_panel_control	services, 331
Audio/Video Remote Control Profile (AVRCP), 180	bt_gatt_client_evt_discover_char_by_uuid_completed←
bt_avrcp_send_simple_panel_cmd	_t, 331
Audio/Video Remote Control Profile (AVRCP), 180	_t, 331 chars, 331
·	
bt_avrcp_set_absolute_volume	bt_gatt_client_evt_discover_descriptors_completed_t,
Audio/Video Remote Control Profile (AVRCP), 181	332
bt_avrcp_set_addressed_player	descriptors, 332
Audio/Video Remote Control Profile (AVRCP), 181	bt_gatt_client_evt_discover_service_by_uuid_completed
bt_avrcp_set_player_application_setting_value	_t, 332
Audio/Video Remote Control Profile (AVRCP), 181	services, 332
bt_avrcp_start	bt_gatt_client_evt_exchange_mtu_completed_t, 333
Audio/Video Remote Control Profile (AVRCP), 182	bt_gatt_client_evt_find_included_completed_t, 333
bt_avrcp_tg_send_element_attributes	services, 333
Audio/Video Remote Control Profile (AVRCP), 182	bt_gatt_client_evt_profile_found_t, 334
bt_avrcp_tg_set_absolute_volume	bt_gatt_client_evt_read_by_char_uuid_completed_t,
Audio/Video Remote Control Profile (AVRCP), 182	334
bt_avrcp_tg_set_battery_status	values, 334
Audio/Video Remote Control Profile (AVRCP), 183	bt_gatt_client_evt_read_char_descriptor_completed_t,
bt_avrcp_tg_set_current_track	334
Audio/Video Remote Control Profile (AVRCP), 183	bt_gatt_client_evt_read_char_long_descriptor_←
bt_avrcp_tg_set_play_status	completed_t, 335
Audio/Video Remote Control Profile (AVRCP), 183	bt_gatt_client_evt_read_char_long_value_completed ←
bt_avrcp_tg_set_system_status	_t, 335
Audio/Video Remote Control Profile (AVRCP), 184	bt_gatt_client_evt_read_char_value_completed_t, 336
bt_cp_header_t, 328	bt_gatt_client_evt_read_multiple_char_values_ ←
bt_gatt_client_accept_conn_param_update	completed_t, 336
GATT Client, 226	bt_gatt_client_evt_t, 337
bt_gatt_client_allocate_session	bt_gatt_client_evt_value_notification_t, 338
GATT Client, 227	bt_gatt_client_evt_write_char_descriptor_completed_t,
bt_gatt_client_char_declaration_t, 328	338
bt_gatt_client_char_descriptor_t, 328	bt_gatt_client_evt_write_char_long_descriptor_←
bt_gatt_client_char_value_t, 329	completed_t, 339
bt_gatt_client_connect	bt_gatt_client_evt_write_char_long_value_completed↔
GATT Client, 227	_t, 339
bt_gatt_client_disconnect	bt_gatt_client_evt_write_char_value_completed_t, 339
GATT Client, 227	bt_gatt_client_find_included_services_ex
5/ ( ) Ollong == /	gan_onon_ma_moladod_oorviood_ox

GATT Client, 230	bt_gatt_server_notify_char_value
bt_gatt_client_free_sessions	GATT Server, 217
GATT Client, 230	bt_gatt_server_notify_char_value_80
bt_gatt_client_get_hci_connection	GATT Server, 217
GATT Client, 231	bt_gatt_server_register_callback
bt_gatt_client_inc_service_declaration_t, 340	GATT Server, 218
bt_gatt_client_init	bt_gatt_server_register_listener
GATT Client, 231	GATT Server, 219
bt_gatt_client_listener_t, 340	bt_gatt_server_start
bt_gatt_client_mgr_t, 340	GATT Server, 215
bt_gatt_client_profile_finder_state_t, 340	bt_gatt_server_start_ex
bt_gatt_client_profile_finder_t, 340	GATT Server, 219
bt_gatt_client_profile_header_t, 340	bt_gatt_server_unregister_listener
bt_gatt_client_read_by_char_uuid	GATT Server, 220
GATT Client, 231	bt_gatt_server_write_char_value
bt_gatt_client_read_by_char_uuid_80	GATT Server, 220
GATT Client, 232	bt_gatt_server_write_char_value_80
bt_gatt_client_read_char_descriptor	GATT Server, 221
GATT Client, 232	bt_hci_command_p, 343
bt_gatt_client_read_char_long_descriptor	bt_hci_connect
GATT Client, 232	HCI, 24
bt_gatt_client_read_char_long_value	bt_hci_connect_callback_fp
GATT Client, 233	HCI, 23
bt_gatt_client_read_char_value	bt_hci_data_p, 344
GATT Client, 233	bt_hci_disconnect_callback_fp
bt_gatt_client_read_multiple_char_values	HCI, 24
GATT Client, 233	bt_hci_event_p, 344
bt_gatt_client_reject_conn_param_update	bt_hci_init
GATT Client, 226	HCI, 25
bt_gatt_client_service_definition_t, 340	bt_hci_init_ex
bt_gatt_client_session_t, 341	HCI, 25
bt_gatt_client_set_char_value	bt_hci_inquiry_response_t, 344
GATT Client, 234	bt_hci_listen
bt_gatt_client_write_char_descriptor	HCI, 25
GATT Client, 234	bt_hci_reset
bt_gatt_client_write_char_long_descriptor	HCI, 25
GATT Client, 234	bt_hci_start
bt_gatt_client_write_char_long_value	HCI, 26
GATT Client, 235	bt_hci_start_callback_fp
bt_gatt_client_write_char_value	HCI, 24
GATT Client, 235	bt_hci_start_no_init
bt_gatt_evt_client_config_changed_t, 341	HCI, 26
bt_gatt_evt_ext_properties_changed_t, 341	bt_hci_stop
bt_gatt_evt_header_t, 342	HCI, 26
bt_gatt_evt_server_config_changed_t, 342	bt_hci_stop_callback_fp
bt_gatt_evt_value_changed_t, 343	HCI, 24
bt_gatt_evt_value_read_t, 343	bt_hci_write_local_name
bt_gatt_exchange_mtu	HCI, 27
GATT Client, 235	bt_hcitr_uart_init
bt_gatt_get_client_mgr	HCI UART (H4) transport protocol, 28
GATT Client, 236	bt_hcitr_uart_reset
bt_gatt_listener_t, 343	HCI UART (H4) transport protocol, 28
bt_gatt_server_authorize_access	bt_hcitr_uart_start
GATT Server, 214	HCI UART (H4) transport protocol, 28
bt_gatt_server_indicate_char_value	bt_hf_subscriber_number_t, 344
GATT Server, 215	bt_hfp_activate_voice_recognition
bt_gatt_server_indicate_char_value_80	Hands Free Profile (HFP), 64
GATT Server, 216	bt_hfp_ag_call_connected

Hands Free Profile (HFP), 64	bt_hfp_evt_query_operator_completed_t, 350
bt_hfp_ag_call_disconnected	bt_hfp_evt_send_dtmf_code_t, 351
Hands Free Profile (HFP), 64	bt_hfp_evt_slc_connection_state_changed_t, 351
bt_hfp_ag_evt_search_completed_t, 344	bt_hfp_evt_spk_volume_changed_t, 351
bt_hfp_ag_incoming_call	bt_hfp_evt_terminate_call_t, 351
Hands Free Profile (HFP), 64	bt_hfp_evt_voice_recognition_changed_t, 351
bt_hfp_ag_outgoing_call	bt_hfp_find_remote_callback_pf
Hands Free Profile (HFP), 65	Hands Free Profile (HFP), 62
bt_hfp_ag_outgoing_call_alerting	bt_hfp_free_session
Hands Free Profile (HFP), 65	Hands Free Profile (HFP), 70
bt_hfp_ag_reject_outgoing_call	bt_hfp_get_indicator_value
Hands Free Profile (HFP), 65	Hands Free Profile (HFP), 70
bt_hfp_ag_set_battery_level	bt_hfp_get_remote_address
Hands Free Profile (HFP), 66	Hands Free Profile (HFP), 60
bt_hfp_ag_set_inband_ring	bt_hfp_has_active_call
Hands Free Profile (HFP), 66	Hands Free Profile (HFP), 60
bt_hfp_ag_set_mic_gain	bt_hfp_has_alerting_call
Hands Free Profile (HFP), 59	Hands Free Profile (HFP), 60
bt_hfp_ag_set_operator_name	bt_hfp_has_call_with_status
Hands Free Profile (HFP), 66	Hands Free Profile (HFP), 71
bt_hfp_ag_set_roam_state	bt_hfp_has_dialing_call
Hands Free Profile (HFP), 66	Hands Free Profile (HFP), 61
bt_hfp_ag_set_service_state	bt_hfp_has_held_call
Hands Free Profile (HFP), 68	Hands Free Profile (HFP), 61
bt_hfp_ag_set_signal_strength	bt_hfp_has_incoming_call
Hands Free Profile (HFP), 68	Hands Free Profile (HFP), 61
bt_hfp_ag_set_speaker_volume	bt_hfp_has_outgoing_call
Hands Free Profile (HFP), 59	Hands Free Profile (HFP), 61
bt_hfp_ag_set_subscriber_number	bt_hfp_has_waiting_call
Hands Free Profile (HFP), 68	Hands Free Profile (HFP), 62
bt_hfp_allocate_session	bt_hfp_hf_answer
Hands Free Profile (HFP), 69	Hands Free Profile (HFP), 71
bt_hfp_allocate_session_ag	bt_hfp_hf_dial_number
Hands Free Profile (HFP), 59	Hands Free Profile (HFP), 71
bt_hfp_allocate_session_hf	bt_hfp_hf_disable_nrec
Hands Free Profile (HFP), 60	Hands Free Profile (HFP), 71
bt_hfp_audio_packet_t, 344	bt_hfp_hf_disconnect
bt_hfp_call_t, 345	Hands Free Profile (HFP), 72
bt_hfp_connect	bt_hfp_hf_enable_call_waiting_notification
Hands Free Profile (HFP), 69	Hands Free Profile (HFP), 72
bt_hfp_connect_audio	bt_hfp_hf_enable_calling_line_identification
Hands Free Profile (HFP), 69	Hands Free Profile (HFP), 72
bt_hfp_disconnect_audio	bt_hfp_hf_enable_voice_recognition
Hands Free Profile (HFP), 70	Hands Free Profile (HFP), 73
bt_hfp_event_register_t, 346	bt_hfp_hf_find_ag
bt_hfp_evt_activate_voice_recognition_t, 346	Hands Free Profile (HFP), 73
bt hfp evt answer call t, 346	bt_hfp_hf_get_subscriber_number
bt_hfp_evt_audio_connection_state_changed_t, 346	Hands Free Profile (HFP), 73
bt_hfp_evt_audio_data_t, 347	bt_hfp_hf_hold
bt_hfp_evt_audio_packet_sent_t, 347	Hands Free Profile (HFP), 74
bt_hfp_evt_call_waiting_t, 347	bt_hfp_hf_query_operator
bt_hfp_evt_chld_t, 348	Hands Free Profile (HFP), 74
bt_hfp_evt_clip_received_t, 348	bt_hfp_hf_redial
bt_hfp_evt_command_completed_t, 348	Hands Free Profile (HFP), 74
bt_hfp_evt_dial_request_received_t, 348	bt_hfp_hf_refresh_call_list
bt_hfp_evt_inband_ring_changed_t, 349	Hands Free Profile (HFP), 75
bt_hfp_evt_indicator_received_t, 349	bt_hfp_hf_reject
bt_hfp_evt_mic_volume_changed_t, 350	Hands Free Profile (HFP), 75

bt_hfp_hf_t, 352	OEM - Non-volatile Storage Interface, 245
bt_hfp_hf_terminate	bt_oem_timer_clear
Hands Free Profile (HFP), 62	OEM - Timer Interface, 242
bt_hfp_ind, 352	bt_oem_timer_set
bt_hfp_init	OEM - Timer Interface, 242
Hands Free Profile (HFP), 75	bt_sdp_client_evt_connected_t, 354
bt_hfp_listen	bt_sdp_client_evt_disconnected_t, 354
Hands Free Profile (HFP), 75	bt_sdp_data_element_t, 354
bt_hfp_send_audio_callback_pf	bt_sdp_found_attr_list_t, 354
Hands Free Profile (HFP), 63	bt_sdp_request_service_attribute
bt hfp session, 352	SDP, 48
bt_hfp_session_callback_pf	bt_sdp_request_service_search
Hands Free Profile (HFP), 63	SDP, 49
bt_hfp_set_mic_gain	bt_sdp_sequence_t, 354
Hands Free Profile (HFP), 76	bt_sdp_serialization_state_p, 354
bt_hfp_set_speaker_volume	bt_sdp_service_transaction_p, 354
Hands Free Profile (HFP), 76	bt_sdp_start
bt_hid_add_report	SDP, 49
HID Profile (HIDP), 52	bt_sdp_transaction_t, 354
bt_hid_allocate_session	bt_security_mgr_t, 354
HID Profile (HIDP), 52	bt_sm_event_t, 354
bt_hid_connect	bt_sm_evt_connected_t, 354
HID Profile (HIDP), 53	bt_sm_evt_disconnected_t, 354
bt_hid_disconnect	bt_sm_evt_generate_lkt_t, 354
HID Profile (HIDP), 53	bt_sm_evt_ltk_generated_t, 354
bt_hid_free_session	bt_sm_evt_master_csrk_notif_t, 354
HID Profile (HIDP), 53	bt_sm_evt_master_irk_notif_t, 354
bt_hid_init	bt_sm_evt_master_ltk_notif_t, 354
HID Profile (HIDP), 54	bt_sm_evt_master_ltk_request_t, 355
bt_hid_listen	bt_sm_evt_oob_data_request_t, 355
HID Profile (HIDP), 54	bt_sm_evt_pairing_complete_t, 355
bt_hid_pdu_t, 352	bt_sm_evt_pairing_request_t, 355
bt_hid_report_t, 352	bt_sm_evt_passkey_notification_t, 355
bt_hid_send_report	bt_sm_evt_passkey_request_t, 355
HID Profile (HIDP), 54	bt_sm_evt_peer_keys_notif_t, 355
bt_hid_session_t, 352	bt_sm_evt_slave_ltk_request_t, 355
bt hid unplug	bt sm listener t, 355
HID Profile (HIDP), 54	bt_sm_ltk_t, 355
bt_media_packet_t, 352	bt sm pairing features t, 355
bt_oem_log_write	bt_sm_session_t, 355
OEM - Logging Interface, 246	bt_snr_session_t, 500
bt_oem_recv	Serial Port Profile (SPP), 33
OEM - HCI Communication Interface, 241	bt_spp_cancel_receive
bt_oem_recv_callback_fp	Serial Port Profile (SPP), 33
OEM - HCI Communication Interface, 240	bt_spp_cancel_send
bt_oem_send	Serial Port Profile (SPP), 33
OEM - HCI Communication Interface, 241	
	bt_spp_connect
bt_oem_send_callback_fp	Serial Port Profile (SPP), 33
OEM - HCI Communication Interface, 240	bt_spp_deallocate
bt_oem_storage_get_capacity	Serial Port Profile (SPP), 35
OEM - Non-volatile Storage Interface, 244	bt_spp_disconnect
bt_oem_storage_read	Serial Port Profile (SPP), 35
OEM - Non-volatile Storage Interface, 244	bt_spp_get_frame_length
bt_oem_storage_start	Serial Port Profile (SPP), 35
OEM - Non-volatile Storage Interface, 244	bt_spp_get_local_modem_status
bt_oem_storage_stop	Serial Port Profile (SPP), 35
OEM - Non-volatile Storage Interface, 244	bt_spp_get_remote_address
bt oem storage write	Serial Port Profile (SPP), 36

bt_spp_get_remote_modem_status Serial Port Profile (SPP), 36	btx_csr_exec_script_buffer_t, 356 btx_csr_get_cached_temperature
bt_spp_init Serial Port Profile (SPP), 36	CSR, 249 btx_csr_get_rssi_acl
	CSR, 250
bt_spp_listen Serial Port Profile (SPP), 36	btx_csr_init
	CSR, 250
bt_spp_port_event_e	btx_csr_patch_controller
Serial Port Profile (SPP), 31	CSR, 250
bt_spp_port_state_e	btx_csr_pio_direction_mask_t, 356
Serial Port Profile (SPP), 32	btx_csr_pio_unection_mask_t, 356
bt_spp_port_t, 355	btx_csr_pio_t, 356
bt_spp_port_t::_bt_spp_port_flags_t, 253	btx_csr_rssi_acl_t, 356
bt_spp_receive	btx_csr_script_t, 356
Serial Port Profile (SPP), 37	
bt_spp_receive_callback_fp	btx_csr_set_ps_vars CSR, 250
Serial Port Profile (SPP), 31	
bt_spp_send	btx_csr_set_ps_vars_buffer_t, 356
Serial Port Profile (SPP), 37	btx_csr_set_ps_vars_ex
bt_spp_send_callback_fp	CSR, 251
Serial Port Profile (SPP), 31	btx_csr_strm_connect_t, 356
bt_spp_send_status_e	btx_csr_strm_get_sink_t, 356
Serial Port Profile (SPP), 32	btx_csr_strm_get_source_t, 356
bt_spp_set_dtr	btx_csr_var_t, 356
Serial Port Profile (SPP), 38	btx_csr_warm_reset
bt_spp_set_local_modem_status	CSR, 251
Serial Port Profile (SPP), 38	btx_csr_warm_reset_ex
bt_spp_set_rts	CSR, 251
Serial Port Profile (SPP), 38	CSR, 248
bt_spp_state_callback_fp	btx_csr_autobaud, 249
Serial Port Profile (SPP), 31	btx_csr_bc7_sel_host_interface_h4, 249
bt_sys_callback_fp	btx_csr_enable_tx, 249
System Functions, 13	btx_csr_exec_script, 249
bt_sys_get_l2cap_manager	btx_csr_get_cached_temperature, 249
System Functions, 14	<del>*</del>
bt_sys_get_version	btx_csr_get_rssi_acl, 250
System Functions, 14	btx_csr_init, 250
bt_sys_init	btx_csr_patch_controller, 250
System Functions, 14	btx_csr_set_ps_vars, 250
bt_sys_init_ex	btx_csr_set_ps_vars_ex, 251
System Functions, 14	btx_csr_warm_reset, 251
bt_sys_start	btx_csr_warm_reset_ex, 251
System Functions, 15	categories
bt_timer_callback_fp	bt_avdtp_sep_capabilities_t, 315
OEM - Timer Interface, 242	chars
btx_csr_autobaud	bt_gatt_client_evt_discover_all_chars_completed ←
CSR, 249	_t, 330
btx_csr_autobaud_buffer_t, 356	bt_gatt_client_evt_discover_char_by_uuid_←
	completed_t, 331
btx_csr_bc7_sel_host_interface_h4	codec_config
CSR, 249	bt_avdtp_codec_op_parse_config_t, 293
btx_csr_bccmd_header_t, 356	bt_avdtp_codec_op_serialize_config_t, 294
btx_csr_bccmd_listener_t, 356	codec_type
btx_csr_cached_temperature_t, 356	bt_avdtp_codec_t, 294
btx_csr_create_operator_c_t, 356	bt_avdtp_sep_capabilities_t, 315
btx_csr_enable_tx	company_id
CSR, 249	_bt_avrcp_mgr_t, 327
btx_csr_exec_hq_script_buffer_t, 356	Configuration, 50, 109, 147, 185, 197, 211, 237
btx_csr_exec_script	ATT_CLIENT_MAX_SESSIONS, 211
CSR, 249	ATT_MAX_CLIENTS, 197

ATT_MAX_FOUND_ATTRIBUTES, 197 ATT_MAX_QUEUED_WRITE_BUFFER_SIZ↔	END_DE_SEQUENCE SDP, 48
E, 198	err_code
AVCTP_ALLOCATE_BUFFERS_VARS, 147 AVCTP_MAX_CHANNELS, 148	bt_a2dp_evt_open_and_start_stream_completed ← _t, 255
AVCTP MAX TRANSPORT CHANNELS, 148	bt_avdtp_evt_abort_stream_requested_t, 297
AVDTP_CODEC_CONFIG_BUFFER_LEN, 110	bt_avdtp_evt_close_stream_completed_t, 297
AVDTP_MAX_CMD_BUFFERS, 110	bt_avdtp_evt_close_stream_requested_t, 298
AVDTP_MAX_CMD_PARAM_LEN, 110	bt_avdtp_evt_delay_report_completed_t, 299
AVDTP MAX REMOTE DEVICES, 110	bt_avdtp_evt_discover_completed_t, 299
AVDTP_MAX_SEP, 110	bt_avdtp_evt_get_sep_capabilities_completed_t,
AVDTP MAX STREAMS, 110	300
AVDTP MAX TRANSPORT CHANNELS, 110	bt_avdtp_evt_open_stream_completed_t, 302
AVDTP_MAX_TX_BUFFER_LEN, 111	bt_avdtp_evt_open_stream_requested_t, 303
AVBOR_IMAX_TX_BOTTER_EEN, TTT AVRCP_CMD_TIMEOUT, 186	bt_avdtp_evt_reconfigure_stream_requested_t,
	304
AVRCP_MAX_CHANNELS, 186	bt_avdtp_evt_set_stream_configuration_completed-
AVRCP_MAX_CMD_BUFFERS, 186	_t, 305
AVRCP_MAX_CMD_PARAM_LEN, 186	_t, 000 bt_avdtp_evt_set_stream_configuration_requested←
AVRCP_MAX_DEVICE_NAME_LEN, 186	_t, 306
AVRCP_MAX_SEARCH_RESULTS, 186	bt_avdtp_evt_start_stream_completed_t, 307
SDP_MAX_ATTRIBUTE_RESULT_LEN, 50	bt_avdtp_evt_start_stream_requested_t, 307
SDP_MAX_PDU_BUFFERS, 50	bt_avdtp_evt_stream_reconfigure_completed_t,
SDP_MAX_SEARCH_RESULT_LEN, 50	309
cp_specific	bt_avdtp_evt_stream_security_control_completed↔
bt_avdtp_sep_capabilities_t, 315	_t, 310
cp_specific_len	bt_avdtp_evt_suspend_stream_completed_t, 312
bt_avdtp_sep_capabilities_t, 315	bt_avdtp_evt_suspend_stream_requested_t, 312
cp_type	event_id
bt_avdtp_sep_capabilities_t, 315	bt_av_register_notification_t, 284
ctype bt_av_response_t, 285	
bt_av_1esponse_t, 200	failed_cmd
DE BOOL	bt_a2dp_evt_open_and_start_stream_completed ←
SDP, 46	_t, 255
DE INT	flags
SDP, 46	_bt_avctp_mgr_t, 291
DE STRING	_bt_avrcp_mgr_t, 327
SDP, 46	bt_avdtp_mgr_t, 313
DE STRING2	bt_avdtp_stream_t, 318
SDP, 46	format
DE UINT	_bt_att_find_info_response_s, 269
SDP, 46	CATT 040
DE_UINT16	GATT, 212
SDP, 47	GATT Client, 222
DE URL	bt_gatt_client_accept_conn_param_update, 226
SDP, 47	bt_gatt_client_allocate_session, 227
DE_UUID128	bt_gatt_client_connect, 227 bt_gatt_client_disconnect, 227
SDP, 47	bt_gatt_client_discover_all_chars_ex, 228
DE UUID16	bt_gatt_client_discover_all_services_ex, 228
SDP, 47	bt_gatt_client_discover_by_service_uuid, 228
DE_UUID32	bt_gatt_client_discover_by_service_uuid_80, 229
SDP, 47	bt_gatt_client_discover_char_by_uuid, 229
data	bt_gatt_client_discover_char_by_uuid_80, 229
_bt_hfp_audio_packet_s, 344	bt_gatt_client_discover_char_descriptors, 230
_bt_media_packet_t, 353	bt_gatt_client_find_included_services_ex, 230
descriptors	bt_gatt_client_free_sessions, 230
bt_gatt_client_evt_discover_descriptors_completed <	— <del>-</del> — — —
_t, 332	bt_gatt_client_init, 231
<del></del>	<del></del> /

bt_gatt_client_read_by_char_uuid, 231	bt_hfp_ag_call_connected, 64
bt_gatt_client_read_by_char_uuid_80, 232	bt_hfp_ag_call_disconnected, 64
bt_gatt_client_read_char_descriptor, 232	bt_hfp_ag_incoming_call, 64
bt_gatt_client_read_char_long_descriptor, 232	bt_hfp_ag_outgoing_call, 65
bt_gatt_client_read_char_long_value, 233	bt_hfp_ag_outgoing_call_alerting, 65
bt_gatt_client_read_char_value, 233	bt_hfp_ag_reject_outgoing_call, 65
bt_gatt_client_read_multiple_char_values, 233	bt_hfp_ag_set_battery_level, 66
bt_gatt_client_reject_conn_param_update, 226	bt_hfp_ag_set_inband_ring, 66
bt_gatt_client_set_char_value, 234	bt_hfp_ag_set_mic_gain, 59
bt gatt client write char descriptor, 234	bt_hfp_ag_set_operator_name, 66
bt_gatt_client_write_char_long_descriptor, 234	bt_hfp_ag_set_roam_state, 66
bt_gatt_client_write_char_long_value, 235	bt_hfp_ag_set_service_state, 68
bt_gatt_client_write_char_value, 235	bt_hfp_ag_set_signal_strength, 68
bt_gatt_exchange_mtu, 235	bt_hfp_ag_set_speaker_volume, 59
bt_gatt_get_client_mgr, 236	bt_hfp_ag_set_subscriber_number, 68
	_ · _ ·
GATT Server, 213	bt_hfp_allocate_session, 69
bt_gatt_server_authorize_access, 214	bt_hfp_allocate_session_ag, 59
bt_gatt_server_indicate_char_value, 215	bt_hfp_allocate_session_hf, 60
bt_gatt_server_indicate_char_value_80, 216	bt_hfp_connect, 69
bt_gatt_server_notify_char_value, 217	bt_hfp_connect_audio, 69
bt_gatt_server_notify_char_value_80, 217	bt_hfp_disconnect_audio, 70
bt_gatt_server_register_callback, 218	bt_hfp_find_remote_callback_pf, 62
bt_gatt_server_register_listener, 219	bt_hfp_free_session, 70
bt_gatt_server_start, 215	bt_hfp_get_indicator_value, 70
bt_gatt_server_start_ex, 219	bt_hfp_get_remote_address, 60
bt_gatt_server_unregister_listener, 220	bt_hfp_has_active_call, 60
bt_gatt_server_write_char_value, 220	bt_hfp_has_alerting_call, 60
bt_gatt_server_write_char_value_80, 221	bt_hfp_has_call_with_status, 71
HCI, 16	bt_hfp_has_dialing_call, 61
	bt_hfp_has_held_call, 61
bt_hci_connect, 24 bt_hci_connect_callback_fp, 23	bt_hfp_has_incoming_call, 61
bt_hci_disconnect_callback_fp, 24	bt_hfp_has_outgoing_call, 61
bt_hci_init, 25	bt_hfp_has_waiting_call, 62
bt_nci_init, 25 bt hci init ex, 25	bt_hfp_hf_answer, 71
bt_hci_listen, 25	bt_hfp_hf_dial_number, 71
bt_hci_reset, 25	bt_hfp_hf_disable_nrec, 71
	bt_hfp_hf_disconnect, 72
bt_hci_start, 26	bt_hfp_hf_enable_call_waiting_notification, 72
bt_hci_start_callback_fp, 24	bt_hfp_hf_enable_calling_line_identification, 72
bt_hci_start_no_init, 26	bt_hfp_hf_enable_voice_recognition, 73
bt_hci_stop, 26	bt_hfp_hf_find_ag, 73
bt_hci_stop_callback_fp, 24 bt hci write local name, 27	bt_hfp_hf_get_subscriber_number, 73
HCI UART (H4) transport protocol, 28	bt_hfp_hf_hold, 74
bt_hcitr_uart_init, 28	bt_hfp_hf_query_operator, 74
bt_hcitr_uart_reset, 28	bt_hfp_hf_redial, 74
bt_hcitr_uart_start, 28	bt_hfp_hf_refresh_call_list, 75
HID Profile (HIDP), 52	bt_hfp_hf_reject, 75
bt_hid_add_report, 52	bt_hfp_hf_terminate, 62
bt_hid_allocate_session, 52	bt_hfp_init, 75
bt_hid_connect, 53	bt_hfp_listen, 75
bt_hid_disconnect, 53	bt_hfp_send_audio_callback_pf, 63
bt_hid_free_session, 53	bt_hfp_session_callback_pf, 63
	bt_hfp_set_mic_gain, 76
bt_hid_init, 54 bt_hid_listen, 54	bt_hfp_set_speaker_volume, 76
bt_nid_listeri, 54 bt_hid_send_report, 54	header_compression_options
bt_hid_unplug, 54	bt_avdtp_sep_capabilities_t, 315
Hands Free Profile (HFP), 56	INIT_DE_SEQUENCE
bt_hfp_activate_voice_recognition, 64	SDP, 48
bi_iiip_activate_voice_recognition, 04	JDF, <del>40</del>

len	RFCOMM_INFO_LEN
bt_att_client_uuid_t, 264	RFCOMM Configuration, 42
bt_att_dild_t, 204	RFCOMM_LOCAL_CREDIT
marker	RFCOMM Configuration, 42
_bt_media_packet_t, 353	RFCOMM_MAX_CMD_BUFFERS
max_parity_code_packets	RFCOMM Configuration, 42
bt_avdtp_sep_capabilities_t, 316	RFCOMM_MAX_DLCS
max_recovery_window	RFCOMM Configuration, 42
bt avdtp sep capabilities t, 316	RFCOMM MAX SERVER CHANNELS
media type	RFCOMM Configuration, 42
bt_avdtp_evt_sep_info_received_t, 305	RFCOMM MAX SESSIONS
bt avdtp sep capabilities t, 316	RFCOMM Configuration, 42
mode	recovery_type
_bt_hfp_call_s, 345	bt_avdtp_sep_capabilities_t, 316
multiparty	ref_count
_bt_hfp_call_s, 345	bt_avdtp_transport_channel_t, 320
multiplexing_options	st_avatp_nanoport_onannoi_t, o20
bt avdtp sep capabilities t, 316	SDP, 44
51_474tp_55p_54p45iii(155_1, 575	BEGIN_DE_SEQUENCE, 45
notification	bt_sdp_request_service_attribute, 48
bt_avrcp_event_t, 322	bt_sdp_request_service_search, 49
	bt_sdp_start, 49
OEM - HCI Communication Interface, 240	DE_BOOL, 46
bt oem recv, 241	DE_INT, 46
bt_oem_recv_callback_fp, 240	DE STRING, 46
bt_oem_send, 241	DE_STRING2, 46
bt_oem_send_callback_fp, 240	DE_UINT, 46
OEM - Logging Interface, 246	DE_UINT16, 47
bt_oem_log_write, 246	DE URL, 47
OEM - Non-volatile Storage Interface, 244	DE UUID128, 47
bt_oem_storage_get_capacity, 244	DE_UUID16, 47
bt_oem_storage_read, 244	DE UUID32, 47
bt_oem_storage_start, 244	END DE SEQUENCE, 48
bt_oem_storage_stop, 244	INIT DE SEQUENCE, 48
bt_oem_storage_write, 245	SDP MAX ATTRIBUTE RESULT LEN
OEM - Timer Interface, 242	Configuration, 50
bt_oem_timer_clear, 242	SDP_MAX_PDU_BUFFERS
bt_oem_timer_set, 242	Configuration, 50
bt_timer_callback_fp, 242	SDP MAX SEARCH RESULT LEN
offset	Configuration, 50
bt_avdtp_codec_op_parse_config_t, 293	SPP Configuration, 40
bt_avdtp_codec_op_serialize_config_t, 294	SPP MAX PORTS, 40
51_474tp_00400_0p_001141120_0011119_t;	SPP MAX PORTS
payload_type	SPP Configuration, 40
_bt_media_packet_t, 353	SPP_PORT_EVENT_CONNECTED
play_status	Serial Port Profile (SPP), 32
bt_av_play_status_t, 281	SPP_PORT_EVENT_CONNECTION_FAILED
	Serial Port Profile (SPP), 32
RFCOMM Configuration, 41	SPP_PORT_EVENT_DISCONNECTED
RFCOMM_ENABLE_MULTIDEVICE_CHANNE↔	Serial Port Profile (SPP), 32
LS, 42	SPP_PORT_EVENT_LOCAL_MODEM_STATUS_C↔
RFCOMM_INFO_LEN, 42	HANGE_FAILED
RFCOMM_LOCAL_CREDIT, 42	Serial Port Profile (SPP), 32
RFCOMM_MAX_CMD_BUFFERS, 42	SPP_PORT_EVENT_LOCAL_MODEM_STATUS_C↔
RFCOMM_MAX_DLCS, 42	HANGED
RFCOMM_MAX_SERVER_CHANNELS, 42	Serial Port Profile (SPP), 32
RFCOMM_MAX_SESSIONS, 42	SPP_PORT_EVENT_REMOTE_MODEM_STATUS_
RFCOMM_ENABLE_MULTIDEVICE_CHANNELS	CHANGED
RFCOMM Configuration, 42	Serial Port Profile (SPP), 32
• • • • • • • • • • • • • • • • • • •	

SPP_PORT_EVENT_SEND_PROGRESS	SPP_PORT_STATE_CONNECTING, 32
Serial Port Profile (SPP), 32	SPP_PORT_STATE_DISCONNECTED, 32
SPP_PORT_STATE_CONNECTED	SPP_PORT_STATE_DISCONNECTING, 32
Serial Port Profile (SPP), 32	SPP_PORT_STATE_FREE, 32
SPP_PORT_STATE_CONNECTING	SPP_SEND_STATUS_CANCELED, 32
Serial Port Profile (SPP), 32	SPP_SEND_STATUS_INTERRUPTED, 32
SPP PORT STATE DISCONNECTED	SPP_SEND_STATUS_NOT_ENOUGH_RESO↔
Serial Port Profile (SPP), 32	URCES, 32
SPP_PORT_STATE_DISCONNECTING	SPP SEND STATUS SUCCESS, 32
Serial Port Profile (SPP), 32	SPP_SEND_STATUS_TIMEOUT, 32
SPP_PORT_STATE_FREE	services
Serial Port Profile (SPP), 32	bt_gatt_client_evt_discover_all_services_←
SPP SEND STATUS CANCELED	completed_t, 331
Serial Port Profile (SPP), 32	bt_gatt_client_evt_discover_service_by_uuid_←
SPP_SEND_STATUS_INTERRUPTED	completed_t, 332
	bt_gatt_client_evt_find_included_completed_t, 333
Serial Port Profile (SPP), 32	sessions
SPP_SEND_STATUS_NOT_ENOUGH_RESOURCES	bt_avdtp_stream_t, 318
Serial Port Profile (SPP), 32	ssrc
SPP_SEND_STATUS_SUCCESS	_bt_media_packet_t, 353
Serial Port Profile (SPP), 32	_bt_media_packet_t, 555
SPP_SEND_STATUS_TIMEOUT	_bt_a2dp_mgr_t, 256
Serial Port Profile (SPP), 32	_bt_azop_mgr_t, 290 _bt_avctp_mgr_t, 291
Security Manager, 239	
Serial Port Profile (SPP), 29	_bt_avdtp_sep_t, 317
bt_spp_allocate, 33	_bt_avrcp_mgr_t, 327
bt_spp_cancel_receive, 33	_bt_spp_port_t, 356
bt_spp_cancel_send, 33	bt_avdtp_mgr_t, 313
bt_spp_connect, 33	bt_avdtp_stream_t, 319
bt_spp_deallocate, 35	status
bt_spp_disconnect, 35	_bt_hfp_call_s, 345
bt_spp_get_frame_length, 35	bt_av_battery_status_of_ct_t, 271
bt_spp_get_local_modem_status, 35	bt_av_notification_battery_status_t, 276
bt_spp_get_remote_address, 36	<pre>bt_av_notification_playback_status_changed_t,</pre>
bt_spp_get_remote_modem_status, 36	
bt_spp_init, 36	supported_events
bt_spp_listen, 36	_bt_avrcp_mgr_t, 327
bt_spp_port_event_e, 31	System Functions, 13
bt_spp_port_state_e, 32	bt_sys_callback_fp, 13
bt_spp_receive, 37	bt_sys_get_l2cap_manager, 14
bt_spp_receive_callback_fp, 31	bt_sys_get_version, 14
bt_spp_send, 37	bt_sys_init, 14
bt_spp_send_callback_fp, 31	bt_sys_init_ex, 14
bt_spp_send_status_e, 32	bt_sys_start, 15
bt_spp_set_dtr, 38	thum a
bt_spp_set_local_modem_status, 38	type
bt_spp_set_rts, 38	_bt_avdtp_sep_t, 317
bt_spp_state_callback_fp, 31	bt_avdtp_transport_channel_t, 320
SPP PORT EVENT CONNECTED, 32	bt_avdtp_transport_session_t, 320
SPP_PORT_EVENT_CONNECTION_FAILED, 32	values
SPP PORT EVENT DISCONNECTED, 32	bt_gatt_client_evt_read_by_char_uuid_completed←
SPP_PORT_EVENT_LOCAL_MODEM_STATU	t, 334
	t, 334 Vendor specific extensions to HCI, 247
S_CHANGE_FAILED, 32	vendul spedilid extensions to HOI, 24/
SPP_PORT_EVENT_LOCAL_MODEM_STATU↔	
S_CHANGED, 32	
SPP_PORT_EVENT_REMOTE_MODEM_STA ↔	
TUS_CHANGED, 32	
SPP_PORT_EVENT_SEND_PROGRESS, 32	
SPP_PORT_STATE_CONNECTED, 32	