

# EZ Serial User Guide

## Vela IF820 Modules

*Version 2.1*

---

## Revision History

Version	Date	Notes	Contributors	Approver
1.0	13 Nov 2023	Initial Release	Rikki Horrigan Mark Duncombe	Jonathan Kaye
1.1	22 Dec 2023	Added migration guides for migrating legacy modules to Vela IF820 in <b>2.4.8 Bluetooth® classic SPP</b>	Rikki Horrigan	Dave Drogowski
2.0	3 Mar 2025	Converted to Ezurio template	Sue White	Dave Drogowski
2.1	12 Nov 2025	Noted spp_send_command not implemented in <b>7.2.10.1.</b>	Dave Drogowski	Jonathan Kaye

# Contents

1	Introduction.....	11
1.1	How to use this guide .....	11
1.2	Block diagram .....	12
1.3	Functional overview .....	12
1.3.1	Bluetooth® communication features.....	12
1.3.2	Hardware and communication features .....	12
1.3.3	Firmware overwrite .....	12
2	Getting started .....	13
2.1	Prerequisites .....	13
2.2	Factory default behavior .....	13
2.3	Connecting a Host device .....	14
2.3.1	Connecting the Evaluation board .....	14
2.3.2	Connecting the Serial interface .....	14
2.3.2.1	Connecting GPIO pins .....	15
2.4	Communicating with a Host device .....	15
2.4.1	Using the API protocol in text mode .....	16
2.4.2	Text mode protocol characteristics .....	16
2.4.3	Text mode API command categories.....	16
2.4.3.1	Text mode API example.....	17
2.4.4	Using the API protocol in binary mode .....	19
2.4.4.1	Binary mode protocol characteristics .....	19
2.4.4.2	Binary Mode API Example.....	19
2.4.5	Key similarities and differences between text and binary command mode.....	21
2.4.6	API protocol format auto-detection .....	22
2.4.7	Using CYSPP mode .....	22
2.4.7.1	Starting CYSPP operation.....	22
2.4.7.2	Sending and receiving data in CYSPP data mode.....	23
2.4.7.3	Exiting CYSPP mode .....	23
2.4.7.4	Customizing CYSPP behavior for specific needs .....	23
2.4.7.5	Understanding CYSPP connection keys.....	24
2.4.7.6	Using the CYSPP peripheral connection key.....	24
2.4.7.7	Using the CYSPP Central Connection key and mask.....	25
2.4.7.8	CYSPP configuration and pin states.....	25
2.4.7.9	CYSPP state machine.....	27
2.4.8	Bluetooth® classic SPP.....	27
2.5	Configuration settings, storage, and protection .....	28
2.5.1	Factory, boot, and runtime settings.....	28
2.5.2	Saving runtime settings in flash .....	29
2.5.3	Protected configuration settings .....	29
2.6	Finding related material.....	30
2.6.1	Latest EZ-Serial firmware platform for Ezurio Vela IF820 series module image .....	30
2.6.2	Latest host API protocol library .....	30
2.6.3	Comprehensive API reference .....	30
3	Operational examples.....	30
3.1	System setup examples.....	30
3.1.1	Identifying the running firmware and BLE stack version .....	30
3.1.1.1	Getting version details from boot event.....	30
3.1.1.2	Getting version details on demand.....	31
3.1.2	Changing the serial communication parameters.....	31

3.1.3	Changing device name and appearance .....	33
3.1.4	Changing output power .....	33
3.1.5	Managing sleep states .....	34
3.1.5.1	Configuring the system-wide sleep level .....	35
3.1.5.2	Configuring the CYSPP data mode sleep level .....	36
3.1.5.3	Preventing sleep with the LP_MODE pin.....	36
3.1.5.4	Managing host and module sleep simultaneously .....	36
3.1.6	Performing a factory reset .....	36
3.2	Cable replacement examples with CYSPP .....	37
3.2.1	Getting started in CYSPP mode with zero custom configuration .....	37
3.2.1.1	Starting CYSPP out of the box in peripheral mode .....	38
3.2.1.2	How to start CYSPP out of the box in central mode .....	38
3.3	GAP peripheral examples .....	40
3.3.1	Advertising as peripheral device .....	41
3.3.2	Stopping advertising as a peripheral device .....	41
3.3.3	Customizing advertisement and scanning response data .....	41
3.4	GAP central examples.....	44
3.4.1	How to scan peripherals .....	44
3.4.2	How to stop scanning for peripheral devices.....	45
3.4.3	How to connect to a peripheral device.....	46
3.4.4	How to cancel a pending connection to a peripheral device .....	46
3.4.5	How to disconnect from a peripheral device.....	47
3.5	GATT server examples.....	48
3.5.1	Defining custom local GATT services and characteristics .....	48
3.5.1.1	Understanding custom GATT limitations .....	49
3.5.1.2	Building custom services and characteristics .....	49
3.5.1.3	Choosing correct GATT permissions.....	50
3.5.2	Listing local GATT services, characteristics, and descriptors .....	51
3.5.2.1	Discovering local GATT services .....	51
3.5.2.2	Discovering local GATT characteristics .....	51
3.5.2.3	Discovering local GATT descriptors.....	53
3.5.3	Reading and writing local GATT attribute values.....	54
3.5.3.1	Reading local GATT data.....	54
3.5.3.2	Writing local GATT data.....	54
3.5.4	Notifying and indicating data to a remote client .....	55
3.5.4.1	Notifying data to a remote client.....	55
3.5.4.2	Indicating data to a remote client .....	56
3.5.5	Detecting and processing written data from a remote client .....	57
3.6	GATT client examples .....	57
3.6.1	How to discover a remote server's GATT structure .....	57
3.6.1.1	Discovering remote GATT services.....	57
3.6.1.2	Discovering remote GATT characteristics .....	58
3.6.1.3	Discovering remote GATT descriptors .....	59
3.6.2	How to read and write remote GATT attribute values .....	59
3.6.3	How to detect notified or indicated values from a remote GATT server .....	60
3.7	Security and encryption examples .....	60
3.7.1	Bonding with or without MITM protection .....	60
3.7.1.1	Pairing in "Just Works" mode without MITM protection (BLE) .....	60
3.7.1.2	Pairing with a fixed passkey (BLE) (Obsolete, not supported) .....	61
3.7.1.3	Pairing with a random passkey (BLE).....	62
3.7.1.4	Pairing with a random passkey (BT classic) .....	62

3.8	Performance testing examples .....	63
3.8.1	Maximizing throughput to a remote peer .....	63
3.8.1.1	Maximizing throughput to an iOS device .....	64
3.8.1.2	Maximizing throughput to an Android device .....	64
3.8.1.3	Minimizing power consumption .....	65
3.8.1.4	Minimizing power consumption while broadcasting .....	65
3.8.1.5	Minimizing power consumption while connected .....	65
3.9	Device firmware update examples .....	66
3.9.1	Updating firmware locally using UART .....	66
3.10	GPIO operation examples .....	66
3.10.1	Get current GPIO status .....	66
3.10.2	GPIO configuration when entering or exiting Low-Power state .....	67
3.10.3	GPIO pin configuration .....	67
3.11	Init command examples .....	69
3.11.1	Add Init command .....	69
3.11.2	Display current Init commands .....	69
3.11.3	Check Init command is executed at system start up .....	69
3.11.4	Delete Init command .....	70
3.11.5	Enable/disable Init command .....	70
4	Application design examples .....	72
4.1	Smart MCU host with 4-Wire UART and full GPIO connections .....	72
4.1.1	Hardware design .....	72
4.1.2	Module configuration .....	72
4.1.3	Host configuration .....	72
4.2	Dumb terminal host with CYSPP and simple GPIO state indication .....	72
4.2.1	Hardware design .....	72
4.2.2	Module configuration .....	72
4.2.3	Host configuration .....	73
4.3	Module-Only application with Beacon functionality .....	73
4.3.1	Hardware design .....	73
4.3.2	Module configuration .....	73
4.3.3	Host configuration .....	73
5	Host API library .....	73
5.1	Host API library overview .....	73
5.1.1	High level architecture .....	73
5.1.2	Host library design .....	74
5.2	Implementing a project using the Host API library .....	74
5.2.1	Basic application architecture .....	74
5.2.2	Exposed API functions .....	75
5.2.3	Command macros .....	76
5.2.4	Convenience macros .....	76
5.3	Porting the Host API library to different platforms .....	76
5.4	Using the API definition JSON file to create a custom library .....	77
6	Troubleshooting .....	77
6.1	UART communication issues .....	77
6.2	BLE connection issues .....	78
6.3	GPIO signal issues .....	79
7	API protocol reference .....	79
7.1	Protocol structure and communication flow .....	79
7.1.1	API protocol formats .....	79
7.1.1.1	Text format overview .....	79

7.1.1.2	Binary format overview .....	79
7.1.2	API protocol data types.....	79
7.1.3	Binary format details .....	81
7.1.3.1	Byte ordering and structure packing.....	81
7.1.3.2	Binary packet header.....	82
7.2	API commands and responses .....	83
7.2.1	Protocol group (ID=1) .....	83
7.2.1.1	protocol_set_parse_mode (SPPM, ID=1/1) .....	84
7.2.1.2	protocol_get_parse_mode (GPPM, ID=1/2) .....	85
7.2.1.3	protocol_set_echo_mode (SPEM, ID=1/3) .....	85
7.2.1.4	protocol_get_echo_mode (GPEM, ID=1/4) .....	86
7.2.2	System group (ID=2).....	87
7.2.2.1	system_ping (/PING, ID=2/1) .....	87
7.2.2.2	system_reboot (/RBT, ID=2/2) .....	88
7.2.2.3	system_dump (/DUMP, ID=2/3) .....	88
7.2.2.4	system_store_config (/SCFG, ID=2/4) .....	89
7.2.2.5	system_factory_reset (/RFAC, ID=2/5) .....	90
7.2.2.6	system_query_firmware_version (/QFV, ID=2/6) .....	90
7.2.2.7	system_query_random_number (/QRND, ID=2/8) .....	91
7.2.2.8	system_write_user_data (/WUD, ID=2/11) .....	92
7.2.2.9	system_read_user_data (/RUD, ID=2/12) .....	93
7.2.2.10	system_set_bluetooth_address (SBA, ID=2/13) .....	94
7.2.2.11	system_get_bluetooth_address (GBA, ID=2/14) .....	95
7.2.2.12	system_set_sleep_parameters (SSLP, ID=2/19) .....	96
7.2.2.13	system_get_sleep_parameters (GSLP, ID=2/20) .....	97
7.2.2.14	system_set_tx_power (STXP, ID=2/21) .....	98
7.2.2.15	system_get_tx_power (GTXP, ID=2/22) .....	99
7.2.2.16	system_set_transport (ST, ID=2/23) .....	100
7.2.2.17	system_get_transport (GT, ID=2/24) .....	102
7.2.2.18	system_set_uart_parameters (STU, ID=2/25) .....	104
7.2.2.19	system_get_uart_parameters (GTU, ID=2/26) .....	106
7.2.3	GAP Group (ID=4) .....	107
7.2.3.1	gap_connect (/C, ID=4/1) .....	107
7.2.3.2	gap_cancel_connection (/CX, ID=4/2) .....	109
7.2.3.3	gap_update_conn_parameters (/UCP, ID=4/3) .....	109
7.2.3.4	gap_disconnect (/DIS, ID=4/5) .....	111
7.2.3.5	gap_add_whitelist_entry (/WLA, ID=4/6) .....	111
7.2.3.6	gap_delete_whitelist_entry (/WLD, ID=4/7) .....	113
7.2.3.7	gap_start_adv (/A, ID=4/8) .....	113
7.2.3.8	gap_stop_adv (/AX, ID=4/9) .....	115
7.2.3.9	gap_start_scan (/S, ID=4/10) .....	116
7.2.3.10	gap_stop_scan (/SX, ID=4/11) .....	117
7.2.3.11	gap_query_peer_address (/QPA, ID=4/12) .....	117
7.2.3.12	gap_query_rssi (/QSS, ID=4/13) .....	118
7.2.3.13	gap_query_whitelist (/QWL, ID=4/14) .....	119
7.2.3.14	gap_set_device_name (SDN, ID=4/15) .....	119
7.2.3.15	gap_get_device_name (GDN, ID=4/16) .....	120
7.2.3.16	gap_set_device_appearance (SDA, ID=4/17) .....	120
7.2.3.17	gap_get_device_appearance (GDA, ID=4/18) .....	121
7.2.3.18	gap_set_adv_data (SAD, ID=4/19) .....	121
7.2.3.19	gap_get_adv_data (GAD, ID=4/20) .....	122

7.2.3.20	gap_set_sr_data (SSRD, ID=4/21)	123
7.2.3.21	gap_get_sr_data (GSRD, ID=4/22)	124
7.2.3.22	gap_set_adv_parameters (SAP, ID=4/23)	124
7.2.3.23	gap_get_adv_parameters (GAP, ID=4/24)	126
7.2.3.24	gap_set_scan_parameters (SSP, ID=4/25)	127
7.2.3.25	gap_get_scan_parameters (GSP, ID=4/26)	128
7.2.3.26	gap_set_conn_parameters (SCP, ID=4/27)	129
7.2.3.27	gap_get_conn_parameters (GCP, ID=4/28)	130
7.2.4	GATT Server Group (ID=5)	131
7.2.4.1	gatts_create_attr (/CAC, ID=5/1)	131
7.2.4.2	gatts_delete_attr (/CAD, ID=5/2)	133
7.2.4.3	gatts_validate_db (/VGDB, ID=5/3)	134
7.2.4.4	gatts_store_db (/SGDB, ID=5/4)	135
7.2.4.5	gatts_dump_db (/DGDB, ID=5/5)	135
7.2.4.6	gatts_discover_services (/DLS, ID=5/6)	136
7.2.4.7	gatts_discover_characteristics (/DLC, ID=5/7)	137
7.2.4.8	gatts_discover_descriptors (/DLD, ID=5/8)	138
7.2.4.9	gatts_read_handle (/RLH, ID=5/9)	139
7.2.4.10	gatts_write_handle (/WLH, ID=5/10)	139
7.2.4.11	gatts_notify_handle (/NH, ID=5/11)	140
7.2.4.12	gatts_indicate_handle (/IH, ID=5/12)	142
7.2.4.13	gatts_send_writereq_response (/WRR, ID=5/13)	142
7.2.4.14	gatts_set_parameters (SGSP, ID=5/14)	143
7.2.4.15	gatts_get_parameters (GGSP, ID=5/15)	144
7.2.5	GATT Client Group (ID=6)	145
7.2.5.1	gattc_discover_services (/DRS, ID=6/1)	145
7.2.5.2	gattc_discover_characteristics (/DRC, ID=6/2)	146
7.2.5.3	gattc_discover_descriptors (/DRD, ID=6/3)	147
7.2.5.4	gattc_read_handle (/RRH, ID=6/4)	147
7.2.5.5	gattc_write_handle (/WRH, ID=6/5)	148
7.2.5.6	gattc_confirm_indication (/CI, ID=6/6)	149
7.2.5.7	gattc_set_parameters (SGCP, ID=6/7)	150
7.2.5.8	gattc_get_parameters (GGCP, ID=6/8)	150
7.2.6	SMP Group (ID=7)	152
7.2.6.1	smp_query_bonds (/QB, ID=7/1)	152
7.2.6.2	smp_delete_bond (/BD, ID=7/2)	153
7.2.6.3	smp_pair (/P, ID=7/3)	153
7.2.6.4	smp_set_privacy_mode (SPRV, ID=7/9)	154
7.2.6.5	smp_get_privacy_mode (GPRV, ID=7/10)	155
7.2.6.6	smp_set_security_parameters (SSBP, ID=7/11)	155
7.2.6.7	smp_get_security_parameters (GSBP, ID=7/12)	157
7.2.6.8	smp_set_fixed_passkey (SFPK, ID=7/13)	158
7.2.6.9	smp_get_fixed_passkey (GFPK, ID=7/14)	159
7.2.6.10	smp_set_pin_code (SBTPIN, ID=7/15)	159
7.2.6.11	smp_get_pin_code (GBTPIN, ID=7/16)	160
7.2.6.12	smp_send_pinreq_response (/BTPIN, ID=7/17)	161
7.2.7	GPIO Group (ID=9)	162
7.2.7.1	gpio_query_adc (/QADC, ID=9/2)	162
7.2.7.2	gpio_set_drive (SIOD, ID=9/5)	163
7.2.7.3	gpio_get_drive (GIOD, ID=9/6)	164
7.2.7.4	gpio_set_logic (SIOL, ID=9/7)	165

7.2.7.5	gpio_get_logic (GIOL, ID=9/8) .....	165
7.2.7.6	gpio_set_pwm_mode (SPWM, ID=9/11) (Not implemented) .....	166
7.2.7.7	gpio_get_pwm_mode (GPWM, ID=9/12) (Not Implemented) .....	167
7.2.8	CYSPP Group (ID=10) .....	169
7.2.8.1	p_cyspp_start (.CYSPPSTART, ID=10/2) .....	169
7.2.8.2	p_cyspp_set_parameters (.CYSPPSP, ID=10/3) .....	169
7.2.8.3	p_cyspp_get_parameters (.CYSPPGP, ID=10/4) .....	171
7.2.8.4	p_cyspp_set_packetization (.CYSPPSK, ID=10/7) .....	172
7.2.8.5	p_cyspp_get_packetization (.CYSPPGK, ID=10/8) .....	174
7.2.9	BT group (ID=14) .....	176
7.2.9.1	bt_start_inquiry (/BTI, ID=14/1) .....	176
7.2.9.2	bt_cancel_inquiry (/BTIX, ID=14/2) .....	177
7.2.9.3	bt_query_name (/BTQN, ID=14/3) .....	177
7.2.9.4	bt_connect (/BTC, ID=14/4) .....	178
7.2.9.5	bt_cancel_connection (/BTCX, ID=14/5) (Not implemented) .....	179
7.2.9.6	bt_disconnect (/BTDIS, ID=14/6) .....	180
7.2.9.7	bt_query_connections (/BTQC, ID=14/7) .....	181
7.2.9.8	bt_query_peer_address (/BTQPA, ID=14/8) .....	181
7.2.9.9	bt_query_rssi (/BTQSS, ID=14/9) .....	182
7.2.9.10	bt_set_parameters (SBTP, ID=14/10) .....	183
7.2.9.11	bt_get_parameters (GBTP, ID=14/11) .....	183
7.2.9.12	bt_set_device_class (SBTDC, ID=14/12) .....	184
7.2.9.13	bt_get_device_class (GBTDC, ID=14/13) .....	185
7.2.10	Spp group (ID=19) .....	185
7.2.10.1	spp_send_command (.SPPS, ID=19/1) .....	186
7.2.10.2	spp_set_config (.SPPSC, ID=19/2) (Not implemented) .....	187
7.2.10.3	spp_get_config (.SPPGC, ID=19/3) .....	187
7.3	API events .....	188
7.3.1	System Group (ID=2) .....	188
7.3.1.1	system_boot (BOOT, ID=2/1) .....	188
7.3.1.2	system_error (ERR, ID=2/2) .....	189
7.3.1.3	system_factory_reset_complete (RFAC, ID=2/3) .....	189
7.3.1.4	system_dump_blob (DBLOB, ID=2/5) .....	189
7.3.2	GAP Group (ID=4) .....	191
7.3.2.1	gap_whitelist_entry (WL, ID=4/1) .....	191
7.3.2.2	gap_adv_state_changed (ASC, ID=4/2) .....	191
7.3.2.3	gap_scan_state_changed (SSC, ID=4/3) .....	192
7.3.2.4	gap_scan_result (S, ID=4/4) .....	193
7.3.2.5	gap_connected (C, ID=4/5) .....	194
7.3.2.6	gap_disconnected (DIS, ID=4/6) .....	194
7.3.2.7	gap_connection_updated (CU, ID=4/8) .....	195
7.3.3	GATT Server Group (ID=5) .....	196
7.3.3.1	gatts_discover_result (DL, ID=5/1) .....	196
7.3.3.2	gatts_data_written (W, ID=5/2) .....	197
7.3.3.3	gatts_indication_confirmed (IC, ID=5/3) .....	198
7.3.3.4	gatts_db_entry_blob (DGATT, ID=5/4) .....	198
7.3.4	GATT Client Group (ID=6) .....	200
7.3.4.1	gattc_discover_result (DR, ID=6/1) .....	200
7.3.4.2	gattc_remote_procedure_complete (RPC, ID=6/2) .....	202
7.3.4.3	gattc_data_received (D, ID=6/3) .....	202
7.3.4.4	gattc_write_response (WRR, ID=6/4) .....	203



7.3.5	SMP Group (ID=7) .....	204
7.3.5.1	smp_bond_entry (B, ID=7/1) .....	204
7.3.5.2	smp_pairing_requested (P, ID=7/2) .....	204
7.3.5.3	smp_pairing_result (PR, ID=7/3) .....	205
7.3.5.4	smp_encryption_status (ENC, ID=7/4) .....	206
7.3.5.5	smp_passkey_display_requested (PKD, ID=7/5) .....	206
7.3.5.6	smp_pin_entry_requested (BTPIN, ID=7/7) .....	207
7.3.6	GPIO Group (ID=9) .....	208
7.3.6.1	gpio_interrupt (INT, ID=9/1) .....	208
7.3.7	CYSPP Group (ID=10) .....	209
7.3.7.1	p_cyspp_status (.CYSPP, ID=10/1) .....	209
7.3.8	Bluetooth® Classic Group (ID=14) .....	210
7.3.8.1	bt_inquiry_result (BTIR, ID=14/1) .....	210
7.3.8.2	bt_name_result (BTINR, ID=14/2) .....	211
7.3.8.3	bt_inquiry_complete (BTIC, ID=14/3) .....	211
7.3.8.4	bt_connected (BTCON, ID=14/4) .....	212
7.3.8.5	bt_connection_status (BTCS, ID=14/5) .....	212
7.3.8.6	bt_connection_failed (BTCF, ID=14/6) .....	213
7.3.8.7	bt_disconnected (BTDIS, ID=14/7) .....	213
7.3.9	Spp group (ID=19) .....	215
7.3.9.1	SPP_data_received (SPPD, ID=19/1) .....	215
7.4	Error codes .....	216
7.4.1	EZ-Serial firmware platform for Vela IF820 system error codes .....	216
7.4.2	EZ-Serial firmware platform for Vela IF820 GATT database validation error codes .....	220
7.5	Macro definitions .....	221
8	GPIO Reference .....	222
8.1	GPIO pin map for Vela IF820 DVK .....	222
9	GATT profile .....	222
9.1	CYSPP Profile .....	222
10	Configuration example reference .....	222
10.1	Factory default settings .....	223
10.2	Adopted Bluetooth SIG GATT profile structure snippets .....	224
10.2.1	Generic access service (0x1800) .....	224
10.2.2	Generic Attribute Service (0x1801) .....	225
10.2.3	Immediate alert service (0x1802) .....	225
10.2.4	Link loss service (0x1803) .....	225
10.2.5	TX power service (0x1804) .....	225
11	EZ-Serial firmware platform for Ezurio Vela IF820 series module MAC address .....	226
12	Glossary .....	227
13	Additional Information .....	229

## About this document

This document provides a complete guide to EZ-Serial firmware platform for Ezurio's Vela IF820 series modules.

## Scope and purpose

This document introduces EZ-Serial firmware platform for Ezurio Vela IF820 series modules. EZ-Serial is a firmware platform built on top of Ezurio Vela IF820 series modules, provides an easy-to-use method for accessing the most common hardware and communication features for dual-mode Bluetooth® applications.

This document covers the following concepts related to EZ-Serial and provides all information required to interface to the EZ-Serial firmware platform on target of Ezurio Vela IF820 series module:

- System description and functional overview ([Introduction and Getting started](#))
- Firmware configuration examples ([Operational examples](#))
- Complete design examples ([Application design examples](#))
- API protocol implementation examples for external MCU ([Host API library](#))
- Troubleshooting guides ([Troubleshooting](#))
- Reference material ([API protocol reference through Configuration example reference](#))
- MAC address generation ([EZ-Serial firmware platform for Ezurio Vela IF820 series module MAC address](#))

## Intended audience

This document is intended for application developers creating and testing designs based on EZ-Serial firmware platform for the Ezurio Vela IF820 series modules.

# 1 Introduction

This guide explains EZ-Serial firmware platform for Ezurio Vela IF820 series modules and covers the following:

- Cypress Serial Port Profile (CYSPP) UART-to-BLE bridge functionality
- GPIO status and control connections
- GAP Central and Peripheral operation
- GATT Server and Client data transfers
- Customizable GATT structures
- Security features such as encryption, pairing, and bonding
- API protocol allowing full control over all of these behaviors from an external host
- MAC address generation
- Serial Port Profile (SPP) using Bluetooth® for RS232 serial cable emulation functionality

## 1.1 How to use this guide

The high-level concepts covered in this document are organized into the following categories:

- System description and functional overview ([Introduction and Getting started](#))
- Firmware configuration examples ([Operational examples](#))
- Complete design examples ([Application design examples](#))
- API protocol implementation examples for external MCU ([Host API library](#))
- Troubleshooting guides ([Troubleshooting](#))
- Reference material ([API protocol reference through Configuration example reference](#))
- MAC address generation ([EZ-Serial firmware platform for Ezurio Vela IF820 series module MAC address](#))
- SPP service ([Cable replacement examples with CYSPP](#))

The following approach provides an effective way to gain familiarity with EZ-Serial firmware platform for Ezurio Vela IF820 series module quickly:

1. Read through [Introduction](#) and [Getting started](#) for a functional overview.
2. Find at least one example from [Operational examples](#) that is interesting or relevant to your intended design. Follow with the described configuration on a development kit for a true hands-on experience. These examples provide excellent out-of-the-box feature demonstration:
  - [Getting started in CYSPP mode with zero custom configuration](#)
  - [Defining custom local GATT services and characteristics](#)
  - [Detecting and processing written data from a remote client](#)
  - [Bonding with or without MITM protection](#)
3. Find at least one design example from [Application design examples](#) that is similar to the type of system you intend to use a Ezurio Vela IF820 series module with, especially noting the functional capabilities provided by the configuration and GPIO connections.
4. If you are combining EZ-Serial firmware platform for Ezurio Vela IF820 series module with an external host microcontroller, read through [Host API library](#) to understand how the external MCU will need to communicate with the module.
5. Spend a few minutes reading through the guides in [Troubleshooting](#) to avoid unnecessary frustration later on, in the event that something does not behave in the way you expect.

Note the reference material available in this document allows fast access to additional information and resources available from Infineon. When in doubt, always consult the API reference for helpful information and related content concerning any API command, response, or event.

Throughout the guide, you will find API methods referenced in the following format:

[gap\\_set\\_adv\\_parameters \(SAP, ID=4/23\)](#)

These links contain three important sections:

- Proper descriptive name (for example, [gap\\_set\\_parameters](#)), unique among all other methods.
- Text-mode name (for example, [SAP](#)), applicable when using the API protocol in text mode (see Section [Using the API protocol in text mode](#)).
- Group/method ID values (for example, "4/23"), present in the 4-byte header when using the binary API protocol (see section [Using the API protocol in binary mode](#)).

Click any linked API method for detailed reference material in API protocol reference.

## 1.2 Block diagram

Depending on the specific application, this platform may utilize an external host device, such as a microcontroller (MCU), connected to the module via UART, GPIO pins, or both. Ezurio Vela IF820 series module may communicate with a remote device using Bluetooth® Low Energy (BLE) and Basic Rate/Enhanced Data Rate (BR/EDR) protocol, or both. Throughout this document, BT is used instead of BR/EDR to indicate the support of BR/EDR protocol.

**Note:** All GPIO pins are pre-defined with EZ-Serial firmware for our Vela IF820 series module. See [GPIO pin map](#) for more details.

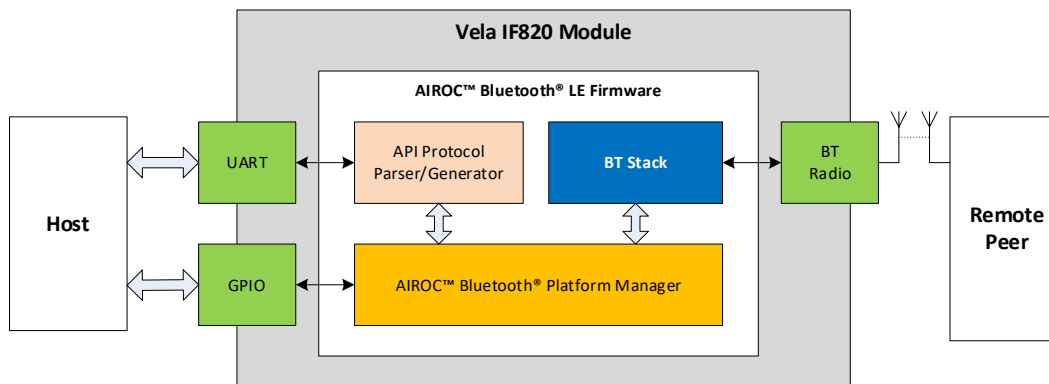


Figure 1: EZ-Serial firmware platform for Ezurio Vela IF820 series module system block diagram

## 1.3 Functional overview

EZ-Serial firmware platform for Ezurio Vela IF820 series module provides an easy way to access the most needed hardware and communication features in Bluetooth® / Bluetooth® LE-based applications. To accomplish this, the firmware implements an intuitive API protocol over the UART interface and exposes a number of status and control signals through the module's GPIO pins.

### 1.3.1 Bluetooth® communication features

The EZ-Serial firmware platform for Ezurio Vela IF820 series modules have the following Bluetooth® / Bluetooth® LE-related features:

- |   |   |
|---|---|
| Bluetooth® 5.x support on compatible modules                              | CYSPP (Cypress Serial Port Profile) mode for bidirectional serial data transmission |
| Master and slave connection roles   | UART and over-the-air (OTA) bootloader for firmware updates                         |
| Central, Peripheral, Broadcaster, and Observer GAP roles                  | Efficient low-power operation   |
| Client and Server GATT roles  | Serial Port Profile (SPP)   |
| Customizable GATT database definition                                     |   |
| Encryption, bonding, and protection from man-in-the-middle (MITM) threats |   |

### 1.3.2 Hardware and communication features

The EZ-Serial firmware platform for Ezurio Vela IF820 series module also implements a number of features that rely on internal CYW208xx chipset features and local interfaces:

- Flexible text-mode and binary-mode API protocols
- GPIO reading, writing, and interrupt detection
- On-demand ADC conversion
- Configurable PWM output
- UART wake-on-RX support
- Initialization commands

### 1.3.3 Firmware overwrite

EZ-Serial firmware platform for Ezurio Vela IF820 series module is a ready-to-use platform intended to satisfy a wide variety of application design requirements with minimal effort. If you have use cases that cannot be handled easily with EZ-Serial on the Ezurio Vela IF820 series module, use the [ModusToolbox™](#) software to build your application firmware image. You can flash a custom firmware image onto any module via the HCI UART interface and completely replace the existing EZ-Serial image at any time. To return to EZ-Serial later, simply download the latest image from the [Ezurio Vela IF820 Firmware repository on Github](#) and flash it using the same mechanism.

## 2 Getting started

### 2.1 Prerequisites

For a streamlined experience, it is recommended that you have the following parts available:

- 453-00172-K1 Vela IF820 - Development Kit with MHF4 Connector  
or 453-00171-K1 Vela IF820 - Development Kit with integrated chip antenna
- Computer with serial terminal software such as [Laird UWTerminalX](#) for text mode or your preferred serial terminal software
- Optional: 450-00185 Vela IF820 - Dual Mode Bluetooth USB Adapter with integrated antenna
- Optional: BLE/ Bluetooth®-capable mobile device such as an iPad, iPhone, or Android phone/tablet
- Optional: Serial terminal software such as RealTerm for use with binary mode

The Ezurio Vela IF820 series module development kits include

- Development Board x1: The development board has the required Vela IF820 module already soldered onto it and exposes all the various hardware interfaces available
- Power supply USB Cable – Type A to MicroUSB. This cable is also used for the following:  
Bluetooth connectivity via the RP2040 on the DVK when the Vela IF820 is configured for HCI UART interface
- RF Antenna (supplied with development kit part # 453-00172-K1 only):  
External antenna, 2 dBi, FlexPIFA (Laird part #001-0022) with integral RF coaxial cable with 100 mm length and IPEX-4 compatible RF connector.
- Jumper cap x 5 Five jumpers for 2.54 mm pitch headers used on Vela IF820 development board.
- Fly leads x 6 Six fly leads for IO pin connection

You can control EZ-Serial firmware platform for Ezurio Vela IF820 series module over a UART interface without additional GPIOs; see Application design examples for details. However, it is recommended that you use the K1 Vela IF820 - Development Kit board for the best experience learning and prototyping due to its more comprehensive design and peripheral support.

### 2.2 Factory default behavior

The following is the default configuration of EZ-Serial firmware platform for Ezurio Vela IF820 series module:

- UART interface configured for 115200 baud, 8 data bits, no parity, 1 stop bit
- UART flow control disabled (signals from the module are not generated, signals from the host are ignored)
- CYSPP serial data transfer profile **enabled in auto-start mode**

When the module is powered on or reset, it will generate the `system_boot` (BOOT, ID=2/1) API event. This is only an example of one API method used by the platform; see API protocol reference for details on the structure and behavior of the API protocol.

The boot event will appear as shown below. The EZ-Serial firmware platform for Ezurio Vela IF820 series module version shown in the below example is 1.4.12.12. This information may differ from the final firmware version for your product.

In text mode, the boot event would look like this:

```
@E, 0076, BOOT, E=01040C0C, S=03010000, P=0104, H=F1, C=00, A=E48AC81565B1, F=EZ-Serial-VELA_IF820_INT V1.4.12.12 Sep 11 2023 10:17:21
```

This text-mode string of data indicates:

- @E - An event has occurred.
- 0076 - There are 118 bytes (0x76) of content to follow.
- BOOT - The event which occurred is the BOOT event.
- E=01040C0C - The EZ-Serial firmware platform for Ezurio Vela IF820 series module application version is 1.4.12.12.
- S=03010000 - The BLE stack component version is 3.1.0 build 0.
- P=0104 - The protocol version is 1.4.
- H=F1 - The hardware platform is CYW20820.
- C=00 - Cause (this is always zero as this feature is not currently supported).
- A=E48AC81565B1 - The Static Random Bluetooth® MAC address of this module is E4:8A:C8:15:65:B1.
- F=EZ-Serial-VELA\_IF820\_INT V1.4.12.12 Sep 11 2023 10:17:21 - Firmware string which includes additional information about the firmware, including release date.

---

**Note:** The version data and MAC address shown here are examples only. Actual values may differ.

---

Once the system boots, EZ-Serial firmware platform for Ezurio Vela IF820 series module will start Bluetooth® classic SPP service and at the same time will start the CYSPP connection process by advertising as peripheral device. When this occurs, the `gap_adv_state_changed` (ASC, ID=4/2) API event will follow the boot event:

```
80 02 04 02 01 03 25
```

In text mode, the same advertisement state change event would look like:

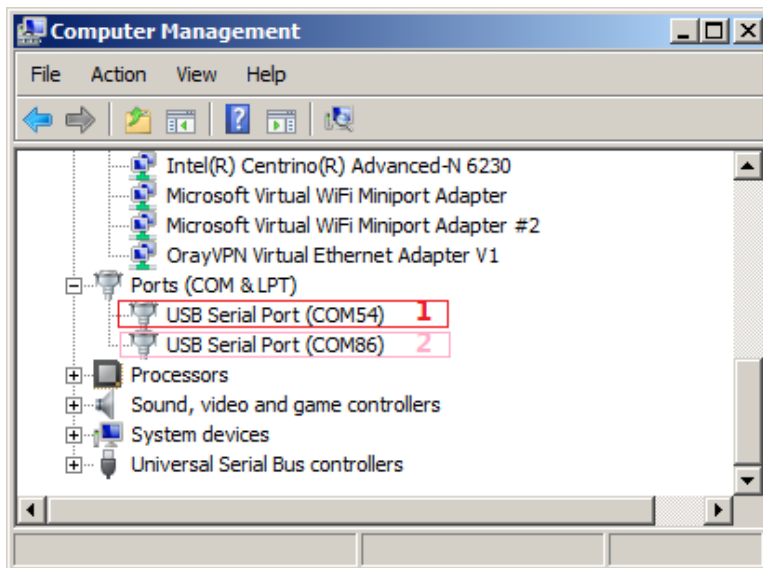
```
@E,000E,ASC,S=01,R=03
```

## 2.3 Connecting a Host device

EZ-Serial firmware platform for Ezurio Vela IF820 series module communicates with an external host device, such as a microcontroller, using serial data (UART), simple GPIO signals, or both for status and control. Depending on your application, you may need to use one, both, or neither of these in your final design. [Application design examples](#) describes each of these use cases.

### 2.3.1 Connecting the Evaluation board

When using the recommended evaluation kit for prototyping, simply connect the micro-USB cable between your PC and the evaluation board. This provides power to the module and a communication interface (UART) via the onboard USB-to-UART bridge. Once you have connected the cable and allowed any necessary drivers to install, two new virtual COM port will become available, as shown in **0 usually the lower one (#1 COM54) is for HCI UART** and higher one (**#2 COM86**) is for PUART.



: Binary command mode session with RealTerm

---

**Note:** COM54 and COM86 are shown in **0** but your port numbers may differ.

---

You can then use the serial port of PUART with any compatible serial terminal software on your PC such as Laird UWTerminalX, Tera Term, Realterm, or PuTTY.

### 2.3.2 Connecting the Serial interface

You can also connect your own host or USB adapter for UART communication. The module's UART interface uses standard true-type logic (TTL) signals, with logic LOW at the GND (0 V) level and logic HIGH at the VDD level (typically 3.3 V).

---

**Attention:** Do not connect the module directly to RS-232 signals which have VDD level range between  $\pm 3 \sim \pm 15$ . To prevent damage to the device, you must add voltage convertors before connecting to RS-232 signals.

---

EZ-Serial firmware platform for Ezurio Vela IF820 series modules UART interface is implemented on the Ezurio Vela IF820 series module PUART interface, which has two required signals for data and two optional signals for flow control, if enabled:

Required: RXD – Receive data (input), connect to host TXD (output)  
 Required: TXD – Transmit data (output), connect to host RXD (input)  
 Optional: RTS – Module-side flow control (output), connect to host CTS (input)  
 Optional: CTS – Host-side flow control (input), connect to host RTS (output)

See section [GPIO pin map](#) for pin-to-function correlations.

**Note:** When connecting to and external UART using an FTDI cable, the cable must supply 3v3 power, or be powered via an alternate 3v3 source. The standard 5V FTDI cable can damage the board. See the [Vela IF820 Development Kit User Guide](#) for information on connecting to an external UART.

The default port settings are 115200 baud, 8 data bits, no parity, and one stop bit. **Flow control is supported but must be specifically enabled if desired.**

You can change these settings using the [system\\_set\\_uart\\_parameters \(STU, ID=2/25\)](#) API command. UART transport settings are protected, which means these settings cannot be written to flash until they have first been applied to RAM. This prevents unintentional communication lockouts. See section [Protected configuration settings](#) for details concerning protected settings.

If you experience any problems communicating over the serial interface, see [Troubleshooting](#) for solutions to common issues.

### 2.3.2.1 Connecting GPIO pins

See the [Ezurio Vela IF820 datasheet and DVK user manual](#) for more information on GPIO pins.

[Table 1](#) summarizes the functions provided by these pins. For additional information, including module-specific pin assignments, operational side effects, and default logic states, see [GPIO Reference](#). Note that some pins are active-HIGH, while some are active-LOW.

#### : Binary packet structure

Pin name	Direction	Functional description
LP_MODE	Input	<p>Low-power mode control.</p> <p>Assert (LOW) to allow sleep, de-assert (HIGH) to disable sleep or exit sleep mode.</p> <hr/> <p><b>Note:</b> The LP_MODE pin is internally pulled up.</p>
CYSPP	Input/output	<p>CYSPP mode control. Assert (LOW) for CYSPP data mode, de-assert (HIGH) for command mode.</p> <hr/> <p><b>Note:</b> Asserting this pin will begin CYSPP operation in the configured role even if the CYSPP profile is disabled in the platform configuration. See section <a href="#">Using CYSPP mode</a> for details. CYSPP is also used in SPP connection: When SPP is established, CYSPP pin is set to LOW by EZ - Serial firmware platform for Ezurio Vela IF820 series module. When host sets CYSPP pin to HIGH, SPP connection will be closed.</p>
CP_ROLE	Input	CYSPP role control. Assert (LOW) for central mode, de-assert (HIGH) for peripheral mode.

For more details on GPIO functionality, see [GPIO Reference](#).

## 2.4 Communicating with a Host device

Once you have connected a host to the module via the serial interface, you can send and receive data. EZ-Serial firmware platform for Ezurio Vela IF820 series module supports two different modes of communication: command mode (API protocol communication and control) and SPP/CYSPP mode (transparent wireless cable replacement to remote device with BT classic or BLE). The following sections describe these modes.

The active communication mode depends on the state of the CYSPP pin, which can be one of the following options:

CYSPP pin externally de-asserted (HIGH): Command mode  
 CYSPP pin externally asserted (LOW): CYSPP mode

CYSPP pin left floating: Command mode until activating CYSPP data pipe, then CYSPP mode

Ensure that the CYSPP pin is in the intended state during boot to achieve the desired behavior. If you assert this pin, the API parser and generator become inactive, because all serial data is piped through the BLE connection (once established). You will experience a lack of communication if you attempt to send API commands to the module while in the CYSPP mode.

SPP service is also initially active. You can establish SPP connection using Bluetooth® classic. The CYSPP pin is also used for the SPP active communication mode. When a SPP connection is established, CYSPP will be asserted to LOW. **When CYSPP pin externally de-asserted to HIGH, the SPP connection is terminated, and the system will return to command mode.**

#### 2.4.1 Using the API protocol in text mode

EZ-Serial firmware platform for Ezurio Vela IF820 series module implements a text-mode API protocol which allows full control of the platform using human-readable commands, responses, and events. This mode is the default setting from the factory to provide the fastest possible path to rapid prototyping. Commands are typed using short codes, and responses and events come back with predictable timing and formats.

#### 2.4.2 Text mode protocol characteristics

The text mode protocol has the following general behavior:

- Commands sent from the host must be terminated with a carriage return (0x0D), line feed (0x0A) byte, or both.
- Commands begin with '/' (forward slash), 'S', 'G', or '.' to indicate ACTION, SET, GET, or PROFILE commands, respectively.
- Commands are always immediately followed by a corresponding response, if they are parsed correctly.
- Commands with multiple arguments allow the arguments to be supplied in any order.
- Commands with multiple arguments do not require all arguments to be present in most cases; SET commands with some arguments omitted will leave non-set values unchanged, and ACTION commands with some arguments omitted will fall back to the default platform settings relevant for those arguments.
- Commands with syntax errors are followed by the **system\_error (ERR, ID=2/2)** API event with an error code indicating the nature of the problem, rather than a response packet (see section 0).
- All numeric data must be entered in hexadecimal notation, without prefixes ("0x") or signs ("+" or "-"); negative numbers should be entered in two's complement form (for example, -1 = FF, -16 = F0, -128 = 80).
- All multi-byte numeric data is entered and expressed in big-endian byte order (for example, 0x12345678 is "12345678").
- Text command codes and hexadecimal data are not case sensitive.
- New command entry in text mode must start with a printable ASCII character (0x20 – 0x7E), or the byte will be ignored.
- Responses always begin with "@R," followed by a 16-bit "length" value describing the number of bytes that come after the four length characters (including the comma), followed by the response text code.
- Responses always include a "result" value as the first parameter after the text code, indicating success or failure.
- Events always begin with "@E," followed by a 16-bit "length" value similar to responses described above.
- Responses and events are terminated with carriage return (0x0D) and line feed (0x0A) bytes.
- Lines beginning with a "#" symbol are treated as comments and discarded by the parser.

#### 2.4.3 Text mode API command categories

There are four main categories of commands in text mode: ACTION, SET, GET, and PROFILE. All these categories use the same basic syntax, but execute different types of behavior.

##### : Binary packet structure

Category	Features
ACTION	<p>ACTION commands trigger operations that cannot persist across resets or power-cycles, with very few exceptions. These commands establish connection, enter into advertisement mode, discover local GATT, and transfer data.</p> <p>The following are the exceptions to the "current session only" rule:</p> <ul style="list-style-type: none"> <li><b>system_store_config (/SCFG, ID=2/4)</b>: Writes all modified settings to flash immediately</li> <li>• <b>system_factory_reset (/RFAC, ID=2/5)</b>: Clears all modified settings and reset the module</li> <li><b>system_write_user_data (/WUD, ID=2/11)</b>: Writes arbitrary user data to a dedicated section of flash</li> <li><b>gatts_create_attr (/CAC, ID=5/1)</b>: Adds custom GATT database attributes</li> <li><b>gatts_delete_attr (/CAD, ID=5/2)</b>: Removes custom GATT database attributes</li> <li><b>smp_pair (/P, ID=7/3)</b>: Initiates pairing, resulting in new bonding data stored in flash</li> </ul>
SET	<p>SET commands affect configuration settings that control many types of behavior, but do not typically trigger immediate changes to the operational state like ACTION commands do.</p> <p>Every argument in a SET command may be stored in non-volatile (flash) memory so that it persists across power-cycles. Modified settings are stored in RAM only by default, and you must use the /SCFG command to write the modified</p>

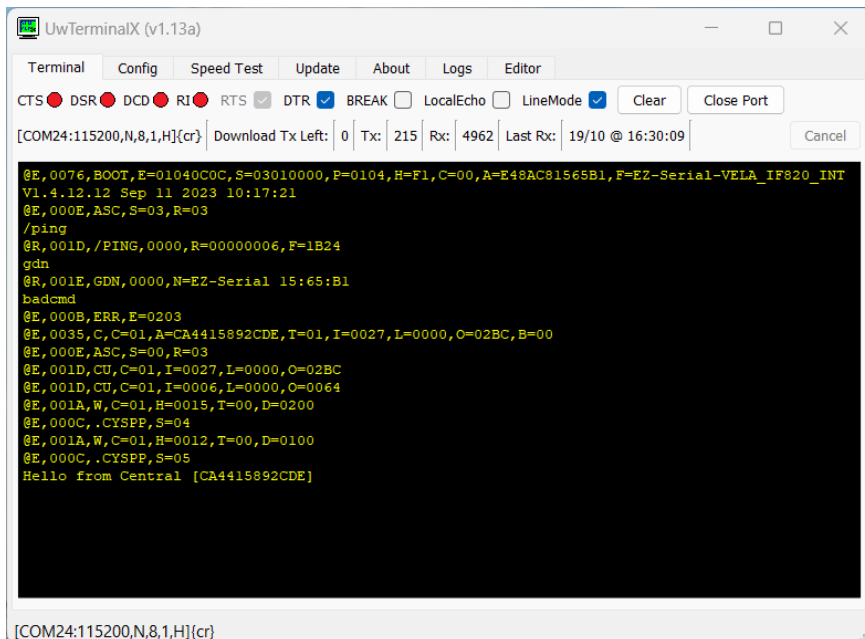


Category	Features
	<p>settings to flash. In text mode, you can also invoke a SET command with a '\$' after the text code (for example, “SDN\$,N=...” ) to cause the change to be written to both RAM and flash immediately.</p> <p>A small number of SET commands also manage protected settings, which are the settings that can affect core chipset operation and communication. For these settings, you cannot write changed values directly to flash without first performing a separate write to RAM only. This prevents accidental changes that are difficult to undo. See section <a href="#">Protected configuration settings</a> has more details on this behavior.</p>
GET	<p>GET commands provide the ability to read all settings that can be changed with SET commands. There is a corresponding GET command for every SET command found in the protocol with matching parameters returned in the response.</p> <p>Like SET commands, GET commands return data from the RAM-stored configuration structure by default. However, using the '\$' after the text code will cause the flash-stored data to be returned instead.</p> <p>Keep in mind that GET/SET commands concern user-defined settings, while ACTION commands concern immediate behavior changes. Always see the API reference material when in doubt about the intended use and behavior of any API method.</p>
PROFILE	<p>PROFILE commands configure the behavior of special built-in behaviors, such as CYSPP data mode. Depending on the profile, these commands may perform actions or get or set configuration values as described for the previous three command types.</p>

For more information on these command categories and behaviors, see the configuration hierarchy in section [Factory, boot, and runtime settings](#) and the material in API protocol reference.

#### 2.4.3.1 Text mode API example

The easiest way to use text command mode is with a serial terminal application. We recommend using [UwTerminalX](#), or you can use any serial terminal application, if it works with standard serial ports and can be configured to open the port with the proper baud rate, flow control, and other settings. [0](#) shows an example session using factory default Vela IF820 EZ-Serial firmware and UwTerminalX application, starting with the [system\\_boot \(BOOT, ID=2/1\)](#) API event and demonstrating a few commands, responses, and other events.



```

[COM24:115200,N,8,1,H]{cr}
@E,0076,BOOT,E=01040C0C,S=03010000,P=0104,H=F1,C=00,A=E48AC81565B1,F=EZ-Serial-VELA_IF820_INT
V1.4.12.12 Sep 11 2023 10:17:21
@E,000E,ASC,S=03,R=03
/ping
@R,001D,/PING,0000,R=00000006,F=1B24
gdn
@R,001E,GDN,0000,N=EZ-Serial 15:65:B1
badcmd
@E,000B,ERR,E=0203
@E,0035,C,C=01,A=CA4415892CDE,T=01,I=0027,L=0000,O=02BC,B=00
@E,000E,ASC,S=00,R=03
@E,001D,CU,C=01,I=0027,L=0000,O=02BC
@E,001D,CU,C=01,I=0006,L=0000,O=0064
@E,001A,W,C=01,H=0015,T=00,D=0200
@E,000C,.CYSPP,S=04
@E,001A,W,C=01,H=0012,T=00,D=0100
@E,000C,.CYSPP,S=05
Hello from Central [CA4415892CDE]
[COM24:115200,N,8,1,H]{cr}

```

#### : Binary command mode session with RealTerm

[0](#) describes the various protocol methods shown in [0](#).

## : Binary packet structure

Direction	Content	Detail
←RX	@E,0076,BOOT,E=01040C0C,S=03010000,P=0104,H=F1,C=00,A=E48AC81565B1,F=EZ-Serial-VELA_IF820_INT V1.4.12.12 Sep 11 2023 10:17:21	<b>system_boot (BOOT, ID=2/1)</b> API event received: E: FW version = 1.4.12.12 S: stack = 3.1.0 build 00 P: protocol = 1.04 H: hardware = CYW20820 C: boot cause = 00 (Not supported) A: MAC address = E4:8A:C8:15:65:B1  F: FW String = EZ-Serial-VELA_IF820_INT V1.4.12.12 Sep 11 2023 10:17:21
←RX	@E,000E,ASC,S=01,R=03	<b>gap_adv_state_changed (ASC, ID=4/2)</b> API event received:  <b>state</b> = 1 (active) <b>reason</b> = 3 (CYSPP operation)
TX→	/ping	<b>system_ping (/PING, ID=2/1)</b> API command sent to ping the local module to verify proper communication
←RX	@R,001D,/PING,0000,R=00000006,F=1B24	<b>system_ping (/PING, ID=2/1)</b> API response received:  <b>result</b> = 0 (success) <b>runtime</b> = 5 seconds <b>fraction</b> = 6948/32768 seconds
TX→	gdn	<b>gap_get_device_name (GDN, ID=4/16)</b> API command sent to get the configured device name
←RX	@R,001E,GDN,0000,N=EZ-Serial 15:65:B1	<b>gap_get_device_name (GDN, ID=4/16)</b> API response received:  <b>result</b> = 0 (success) <b>name</b> = "EZ-Serial BA:A6:19"
←TX	badcmd	Invalid API command sent to demonstrate text mode error event
←RX	@E,000B,ERR,E=0203	<b>system_error (ERR, ID=2/2)</b> API event received:  <b>reason</b> = 0x0203 (Unrecognized Command)
RX→	@E,0035,C,C=01,A=CA4415892CDE,T=01,I=0027,L=0000,O=02BC,B=00	<b>gap_connected (C, ID=4/5)</b> API event received:  <b>conn_handle</b> = 1 <b>peer</b> = CA:44:15:89:2C:DE <b>addr_type</b> = 1 (random/private) <b>interval</b> = 6 (7.5ms) <b>slave_latency</b> = 0 <b>supervision_timeout</b> = 0x64 (100 = 1 second) <b>bond</b> = 0 (not bonded)
←RX	Hello from Central [CA4415892CDE]  @E,001A,W,C=01,H=0015,T=00,D=4000	<b>gatts_data_written (W, ID=5/2)</b> API event received:  <b>conn_handle</b> = 4 <b>attr_handle</b> = 0x15 (21) <b>type</b> = 0 (simple write) <b>data</b> = 2 bytes [40 00]

See the reference material in API protocol reference for details on each of these API methods and text-mode syntax rules.

#### 2.4.4 Using the API protocol in binary mode

EZ-Serial firmware platform for Ezurio Vela IF820 series module implements a binary-format API protocol that allows complete control of the platform using compact binary commands, responses, and events.

The binary protocol uses a fixed packet structure for every transaction in either direction. This fixed structure comprises a 4-byte header followed by an optional payload, terminating with a checksum byte. The payload carries information related to the command, response, or event. If present, this payload always comes immediately after the header and before the checksum byte.

##### : Binary packet structure

Header				Payload (optional)	Checksum
[ 0 ] Type	[ 1 ] Length	[ 2 ] Group	[ 3 ] ID	[ 4 . . . N-1 ] Parameter(s)	[ N ] Summation

The checksum byte is calculated by starting from 0x99 and adding the value of each header and payload byte, rolling over back to 0 (instead of 256) to stay within the 8-bit boundary. The checksum byte itself is not included in the summation process. For the example 4-byte binary packet for the `system_ping` (/PING, ID=2/1) API command:

```
C0 00 02 01
```

Calculate the checksum as follows:

$$0x99 + 0xC0 + 0x00 + 0x02 + 0x01 = 0x15C$$

Retain only the final lower 8 bits (0x5C) for the 1-byte checksum value. The final 5-byte packet (including checksum) is:

```
C0 00 02 01 5C
```

The structure above allows a packet parser implementation to know exactly how much data to expect in advance any time a new packet begins to arrive, and to calculate the checksum as new bytes arrive.

The "Type" byte in the header contains information not only about the packet type (highest two bits), but also the memory scope (where applicable), and the highest three bits of the 11-bit "Length" value. For details on the binary packet format and flow, see the API structural definition in section [Protocol structure and communication flow](#).

##### 2.4.4.1 Binary mode protocol characteristics

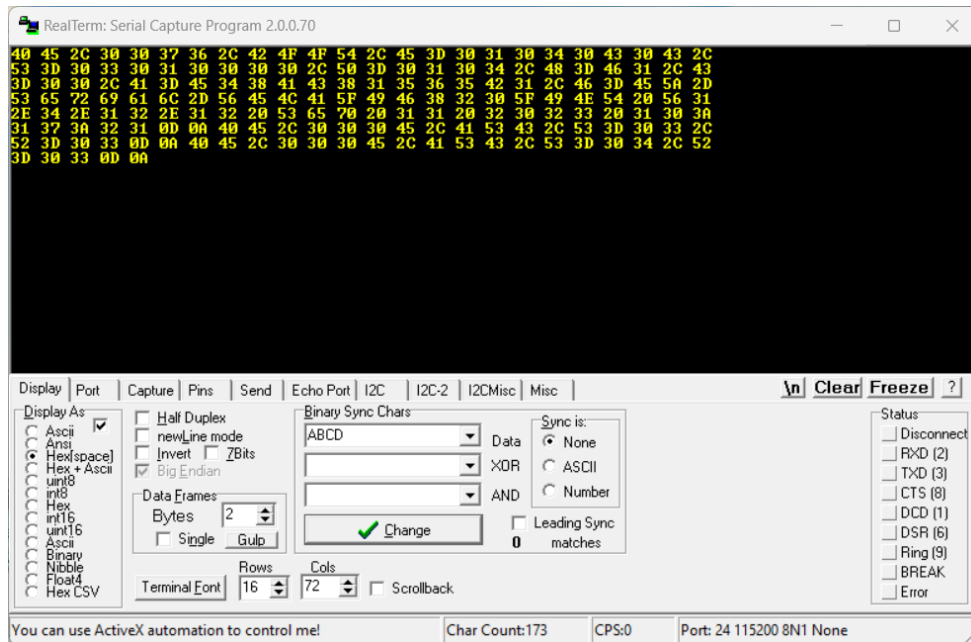
The binary mode protocol has the following general behavior:

- Commands sent from the host must begin with a properly formatted 4-byte header.
- Commands must contain the number of payload bytes specified in the **Length** field from the header.
- Commands must end with a valid checksum byte, but no additional termination such as NULL or carriage return.
- Commands are always *immediately* followed by a response, if they are parsed correctly.
- Commands require all arguments to be supplied in the binary payload according to the protocol structural definition, in the right order (no arguments are optional).
- Commands with syntax errors are followed by a `system_error` (ERR, ID=2/2) API event with an error code indicating the nature of the problem, rather than a response packet.
- Commands must be fully transmitted within one second of the first byte, or the parser will time out and return to an idle state after triggering the `system_error` (ERR, ID=2/2) API event with a timeout error code.
- All multi-byte integer data is entered and expressed in little-endian byte order (for example, 0x12345678 is [78 56 34 12]). Note that this applies only to API method arguments and parameters with a fixed width – 1, 2, or 4 – byte integers, and 6-byte MAC addresses.
- All multi-byte data passed inside a variable-length byte array (uint8a or longuint8a) remains in the original order provided by the source. This includes UUID data found during GATT discovery. If unsure, consult the API reference manual to verify the argument data type.
- Response payloads always begin with a 16-bit "result" value as the first parameter, indicating success or failure of the command triggering the response.
- The binary command header includes a single bit in the first byte, which performs the same duty as the '\$' character in text mode, to cause changed settings to be written to flash immediately instead of just RAM.

##### 2.4.4.2 Binary Mode API Example

The easiest way to use binary command mode is with a host MCU or other application that has a complete parser and generator implementation available, such as the host API library example provided by Infineon and discussed in [Host API Library](#).

However, it is also possible to test individual commands manually with a serial terminal application capable of entering and displaying binary data. 0 shows an example of testing individual commands manually using Realterm, including hexadecimal representation of data. There is no local echo when binary mode is used, so 0 does not show the command packets sent to the module. To assist in identifying the packet types and boundaries, responses are colored cyan, events are yellow, and the final checksum byte of each packet is red.



#### : Binary command mode session with RealTerm

**Note:** Currently UWTerminalX does not support IF820 binary mode but this does not stop you using binary mode with your MCU

Each binary packet (including the checksum byte) is described in Table 5. For better comparison between text mode and binary mode, the API transactions demonstrated here are the same as those used in the text mode example. Note that multi-byte integer data such as the 6-byte MAC address and the 16-bit advertisement interval are transmitted in little-endian byte order.

#### : Binary mode communication example

Direction	Content	Detail
←RX	40 45 2C 30 30 37 36 2C 42 4F 4F 54 2C 45 3D 30 31 30 34 30 43 30 43 2C 53 3D 30 33 30 31 30 30 30 30 2C 50 3D 30 31 30 34 2C 48 3D 46 31 2C 43 3D 30 30 2C 41 3D 45 34 38 41 43 38 31 35 36 35 42 31 2C 46 3D 45 5A 2D 53 65 72 69 61 6C 2D 56 45 4C 41 5F 49 46 38 32 30 5F 49 4E 54 20 56 31 2E 34 2E 31 32 2E 31 32 20 53 65 70 20 31 31 20 32 30 32 33 20 31 30 3A 31 37 3A 32 31 0D 0A	system_boot (BOOT, ID=2/1) API event received: app = 1.4.12.12 stack = 3.1.0 Build 00 protocol = 1.4 hardware = F1 boot cause = N/A MAC address = E4:8A:C8:15:65:B1 Firmware String = F=EZ-Serial-VELA_IF820_INT V1.4.12.12 Sep 11 2023 10:17:21
←RX	80 02 04 02 01 03 25	gap_adv_state_changed (ASC, ID=4/2) API event received: state = 1 (active) reason = 3 (CYSPP operation)

Direction	Content	Detail
TX→	C0 00 02 01 5C <i>(not visible)</i>	system_ping (/PING, ID=2/1) API command sent to ping the local module to verify proper communication
←RX	C0 08 02 01 00 00 0A 00 00 00 A0 6E 7C	system_ping (/PING, ID=2/1) API response received:  result = 0 (success) runtime = 10 seconds fraction = 28320/32768
TX→	C0 00 04 10 6D <i>(not visible)</i>	gap_get_device_name (GDN, ID=4/16) API command sent to get the configured Device Name
←RX	C0 15 04 10 00 00 12 45 5A 2D 53 65 72 69 61 6C 20 31 41 3A 32 31 3A 44 33 A0	gap_get_device_name (GDN, ID=4/16) API response received:  result = 0 (success) name = "EZ-Serial 1A:21:D3"
←RX	80 0F 04 05 40 80 95 19 29 49 80 00 06 00 00 00 64 00 00 FB	gap_connected (C, ID=4/5) API event received:  handle = 40 peer = 80:49:29:19:95:80 addr_type = 0 interval = 6 (7.5 ms) slave_latency = 0 supervision_timeout = 0x64 (100 = 1 second) bond = 0 (not bonded)
←RX	80 0A 05 02 40 0E 00 00 04 00 11 22 33 44 26	gatts_data_written (W, ID=5/2) API event received:  conn_handle = 4 attr_handle = 0x1F (31) type = 0 (simple write) data = 4 bytes [11 22 33 44]
TX→	C0 00 EE EE 35 <i>(not visible)</i>	Invalid API command (group and ID bytes set to 0xEE) sent to demonstrate binary mode error event
←RX	80 02 02 02 03 02 24	system_error (ERR, ID=2/2) API event received:  reason = 0x0203 (Unrecognized Command)

See the reference material in API protocol reference for details concerning each of these API methods and the binary packet format, including information on all header fields and supported data types.

## 2.4.5 Key similarities and differences between text and binary command mode

The text-mode and binary-mode protocol formats provided by EZ-Serial firmware platform for Ezurio's Vela IF820 series module both have their own advantages. As a general guideline, text mode is better for initial development or one-time configuration, while binary mode is a better choice for production-stage control from an external host device due to the significantly less complex parser/generator implementation on an external host. The following lists contain key factors to consider when choosing which mode to use:

### Similarities:

Both modes access the same internal API functionality. They are not different protocols, only different formats.

Both follow the same command, response, and event flow.

EZ-Serial firmware platform for Ezurio Vela IF820 series module supports both modes simultaneously. There is no need to switch between firmware images.

Your choice of protocol format only affects local communication with an external host over the wired serial interface. It does not have any impact on data sent over a wireless BLE connection, or on the type of host communication used on a remote device (for example, another Ezurio Vela IF820 module running EZ-Serial).

## Differences:

- Binary multi-byte integer data is transmitted in little-endian byte order for more efficient direct memory structure mapping on most common platforms, while text mode uses big-endian for easier left-to-right readability.
- Binary commands have a one-second timeout, while text mode commands have no timeout.
- Binary commands are semantically organized by functional group (system, protocol, GAP, GATT Server, and so on) rather than the four categories used in text mode (ACTION, SET, GET, and PROFILE).
- Binary commands require all arguments in every case, while text mode commands often have optional arguments which fall back to default/preset values if omitted.
- Binary packets include basic checksum validation, while text mode packets do not.
- Binary is more efficient for MCU-based communication, while text mode is easier for manual entry in a terminal.
- Binary commands are never echoed back to the host, while text mode commands are (by default).

### 2.4.6 API protocol format auto-detection

EZ-Serial firmware platform for Ezurio Vela IF820 series module uses text mode for API protocol communication by default, but you can change this setting with the `protocol_set_parse_mode (SPPM, ID=1/1)` API command. If "binary" mode is specified and written to flash, the module will use binary mode automatically on subsequent resets or power-cycles.

The parser also automatically detects whether the external host is using binary or text mode, and temporarily switches to the detected mode for the active session. The detection logic behaves in the following way:

- If the parser is in text mode, a byte received at any time with the two most significant bits (MSBs) set (0xC0-0xFF) will switch the parser to binary mode immediately. The "trigger" byte will not be discarded, but will be processed as the first byte in the command packet. This mechanism is considered safe because no valid text-mode command begins with a byte that has the highest two bits set.
- If the parser is in binary mode, a byte received when the parser is idle (not mid-command) that is one of the initial category characters for any of the four types of commands ('I', 'S', 'G', and '.') will switch the parser to text mode immediately. The "trigger" byte will not be discarded but will be processed as the first byte in the text command string. This mechanism is considered safe because no binary command begins with one of these characters. Note that this requires the parser to be idle, not in the middle of a packet, because a binary command packet could easily have one of these characters in its header or payload.

The automatically detected parse mode is not retained across power-cycles, nor is it stored in the same configuration setting area as a value explicitly set by the `protocol_set_parse_mode (SPPM, ID=1/1)` API command. For more detail on this type of temporary configuration, see section [Factory, boot, and runtime settings](#).

### 2.4.7 Using CYSPP mode

EZ-Serial firmware platform for Ezurio Vela IF820 series module implements a special CYSPP profile that provides a simple method to send and receive serial data over a BLE connection. This operational mode is separate from the normal command mode where the API protocol may be used. When CYSPP data mode is active, any data received from an external host will be transmitted to the remote peer, and any data received from the remote peer will be sent out through the hardware serial interface to the external host.

#### 2.4.7.1 Starting CYSPP operation

You can start CYSPP mode using any of these three methods:

1. Assert (LOW) the CYSPP pin externally. You may connect this pin to ground in hardware designs that require CYSPP operation only and never need API communication. You can also use this pin to enter CYSPP mode even if the CYSPP profile is disabled in the platform configuration.
2. Use the `p_cyspp_start (.CYSPPSTART, ID=10/2)` API command. You can use this command to enter CYSPP mode even if the CYSPP profile is disabled in the platform configuration.
3. Have a remote GATT Client connect and subscribe to the CYSPP acknowledged data characteristic (enabling indications) or unacknowledged data characteristic (enabling notifications). This method will enter CYSPP mode only if the CYSPP profile is enabled in the platform configuration.

When starting CYSPP mode locally using either the CYSPP pin or the `p_cyspp_start (.CYSPPSTART, ID=10/2)` API command, the data pipe will not be immediately available because the remote device must still connect and set up proper GATT data subscriptions. If 100% data delivery is required in this context, the Host should monitor the CONNECTION pin to determine when it is safe to begin sending data from the Host for BLE transmission. Once the CONNECTION pin is asserted while the CYSPP pin is also asserted, the Host may send and receive data over CYSPP.

---

**Note:** Externally asserting (LOW) the CYSPP pin will always begin CYSPP operation, even if the profile has been disabled in the platform configuration via the `p_cyspp_set_parameters (.CYSPPSP, ID=10/3)` API command. If you do not require CYSPP operation, you should ensure that this pin remains electrically floating or externally de-asserted (HIGH).

---

#### 2.4.7.2 Sending and receiving data in CYSPP data mode

Once you have started CYSPP mode, the EZ-Serial firmware platform for Ezurio Vela IF820 series module will take care of the rest of the connection process and data pipe construction on the module side. If you are using modules running EZ-Serial firmware platform for Ezurio Vela IF820 series module on both ends of the connection, then simply start CYSPP mode with complementary roles (Peripheral on one end, Central on the other), and the modules will automatically connect and prepare the data pipe using the processes described below.

A non-Vela IF820 device such as a BLE-enabled smartphone will frequently be used for one end of the connection; you must configure the device to follow the same procedure.

For configuration examples in each mode, see section [Cable replacement examples with CYSPP](#).

Follow these steps for other (Non Ezurio Vela IF820 EZ-Serial) devices, such as smartphones, to communicate with EZ-Serial firmware platform for Ezurio Vela IF820 series module in CYSPP mode:

1. EZ-Serial firmware platform for Ezurio Vela IF820 series module begins advertising with configured advertisement settings.
2. Upon connection, a remote peer must subscribe to one of the two "Data" characteristics:
  - a. Acknowledged Data, enable indications (guaranteed reliability)
  - b. Unacknowledged Data, enable notifications (faster potential throughput)
3. Remote peer may optionally subscribe to the "RX Flow Control" characteristic to allow the Server to communicate whether it is safe to write new data.
4. EZ-Serial firmware platform for Ezurio Vela IF820 series module will assert the CONNECTION pin, indicating that CYSPP is ready to send and receive data.
5. The data pipe remains open until the central device disconnects or unsubscribes from the data characteristic, or the CYSPP pin is de-asserted locally.

#### 2.4.7.3 Exiting CYSPP mode

Once in CYSPP mode, the API parser is logically disconnected from incoming serial data, so you will not be able to send any commands to the module. However, you can still exit CYSPP mode in two ways:

1. De-assert (HIGH) the CYSPP pin externally.
2. Have the remote GATT Client unsubscribe from the relevant CYSPP data characteristic (applies only when the CYSPP pin is not externally asserted).

When the CYSPP operation ends, EZ-Serial firmware platform for Ezurio Vela IF820 series module returns to command mode.

***It is not possible to use an API command to exit the CYSPP data mode, because the API parser is not available while in this mode. If your design needs to switch between modes on demand, include external access to the CYSPP pin so you can control the operational mode.***

#### 2.4.7.4 Customizing CYSPP behavior for specific needs

While the default behavior is suitable in many cases, there are configuration settings that allow a great deal of control over this behavior. The following list describes the options that can be changed and how to change the options:

CYSPP mode uses the system's configured UART host transport settings for sending and receiving serial data. To change these settings, use the [system\\_set\\_uart\\_parameters](#) (STU, ID=2/25) API command.

CYSPP mode uses the system's configured radio transmit power setting for all BLE communication. To change this setting, use the [system\\_set\\_tx\\_power](#) (STXP, ID=2/21) API command.

CYSPP mode supports special incoming data packetization modes. This helps make radio transmissions and data delivery more efficient in a variety of use cases. To change these settings, use the [p\\_cyspp\\_set\\_packetization](#) (.CYSPPSK, ID=10/7) API command.

When operating in Peripheral mode, CYSPP uses the system's configured advertisement parameters, including the advertisement and scan response packet content (which may be based on the device name). To change these settings, use one or more of the following API commands:

[gap\\_set\\_adv\\_parameters](#) (SAP, ID=4/23)  
[gap\\_set\\_adv\\_data](#) (SAD, ID=4/19)  
[gap\\_set\\_sr\\_data](#) (SSRD, ID=4/21)  
[gap\\_set\\_device\\_name](#) (SDN, ID=4/15)



#### 2.4.7.5 Understanding CYSPP connection keys

EZ-Serial firmware platform for Ezurio Vela IF820 series module also supports CYSPP connection keys, which improve usability in environments where multiple CYSPP-capable devices are operating in an automated configuration. This feature allows an advertising peripheral device to broadcast an arbitrary 4-byte value that a scanning device can filter against, searching either for a masked range of devices or a single specific device.

CYSPP connection keys are not set in the factory default configuration; CYSPP Peripheral advertisements contain a “0” key. To change this, use the `p_cyspp_set_parameters (.CYSPPSP, ID=10/3)` API command, and specifically the “local\_key”, “remote\_key”, and “remote\_mask” arguments of this command as described in the following sections.

#### 2.4.7.6 Using the CYSPP peripheral connection key

The CYSPP Peripheral connection key affects only the content of the advertisement packet while the module is in an advertising state. The CYSPP Peripheral role does not include any filtering behavior; filtering is left to the scanning device that is operating in the CYSPP Central role.

When the CYSPP profile is enabled, the platform-managed advertising packet contains a special Manufacturer Data field to hold the local connection key value. It is not stored elsewhere, such as in a GATT characteristic. This advertisement packet field has the structure shown in 0.

##### : CYSPP peripheral connection key manufacturer data field structure

Length	Type	Company ID	Connection key
07	FF	b0 b1	b0 b1 b2 b3

The Company ID value is a 16-bit value that the Bluetooth®SIG assigns to member companies that have requested them (see resources on [www.bluetooth.com](http://www.bluetooth.com)). The factory default value is the Cypress company identifier, 0x0131(Cypress is an Infineon Technologies Company now), but you can change this with the same command used to change other CYSPP parameters. Note that both the Company ID and the Connection Key values are broadcast in little-endian byte order.

Use the `p_cyspp_set_parameters (.CYSPPSP, ID=10/3)` API command and enter the desired 32-bit value for the “local\_key” argument to apply a new Peripheral connection key. Changes take effect immediately, even if the module is already advertising in the CYSPP peripheral role.

**EZ-Serial firmware platform for Ezurio Vela IF820 series module incorporates only the CYSPP Peripheral connection key into the advertising packet if you have not enabled user-defined advertisement content. If you have configured user-defined advertisement content instead as described in section [Customizing advertisement and scanning response data](#), then changing this value will have no effect. You must ensure that your user-defined advertisement packet contains an equivalent field to allow scanning devices to filter properly.**

##### : Example 1 - Update CYSPP peripheral key to 0x11223344

Direction	Text Content	Binary Content	Effect
TX→	.CYSPPSP, E=2, G=0, C=0131, L=11223344, R=0, M=0, P=1, S=0, F=2	C0 13 0A 03 02 00 31 01 44 33 22 11 00 00 00 00 00 00 00 01 00 02 5A	Apply new CYSPP configuration
←RX	@R, 000E, .CYSPPSP, 0000	C0 02 0A 03 00 00 68	Response indicates success



#### 2.4.7.7 Using the CYSPP Central Connection key and mask

The CYSPP central connection key affects the scanning operation that occurs when CYSPP is active in the central role and has not yet connected to a remote peer. The central connection key has two parts:

**remote\_key:** The value used for comparison with the peripheral key from the advertisement packet.

**remote\_mask** – The bitmask used to strip away any irrelevant bits from the peripheral key before comparison.

For EZ-Serial firmware platform for Ezurio Vela IF820 series module to initiate a connection to a CYSPP peripheral device, the “remote\_key” value must match with advertised peripheral connection key after a logical AND operation with the “remote\_mask” value. A mask with all bits set (“FFFFFFFF”) will require an exact match between the two keys, while a mask with no bits set (“00000000”) will match any device. The factory default configuration is the all-zero mask, so any CYSPP-capable peer will match. The mask values between these two extremes provide the option to connect only to devices within specific segments of the connection key space, much like an IP-based network. 0 provides examples of each case.

##### Connection key and mask examples

Remote key	Remote mask	Key and mask	Result
11223344	FFFFFFFF	11223344	Connect to a device whose key is exactly “11223344”
55667788	FFFFFF00	55667700	Connect to any device whose key begins with “556677”
12345789	FFFF0000	12340000	Connect to any device whose key begins with “1234”
18F7A9CC	FFFF00FF	18F700CC	Connect to any device whose key begins with “18F7” and ends with “CC”
Any	00000000	00000000	Connect to any device

Use the `p_cyspp_set_parameters` (.CYSPPSP, ID=10/3) API command and enter the desired 32-bit values for the “remote\_key” and “remote\_mask” arguments to apply a new central connection key and mask. Changes to these values will take effect immediately, even if the module is already scanning in the CYSPP central role.

**Note:** If an advertising peripheral device is broadcasting the CYSPP service UUID but does not have a Manufacturer Data field containing a connection key in the same advertisement packet, the value “0” will be substituted for an actual key for the purpose of filtering on the scanning device.

##### Example 1 - Update CYSPP central key to 0x11223344 and require exact matching

Direction	Content	Effect
TX→	.CYSPPSP, R=11223344, M=FFFFFFFF	Apply new CYSPP configuration
←RX	@R, 000E, .CYSPPSP, 0000	Response indicates success

#### 2.4.7.8 CYSPP configuration and pin states

0 describes the relationship between the state of the CYSPP pin and the CYSPP firmware configuration managed with the `p_cyspp_set_parameters` (.CYSPPSP, ID=10/3) API command. Note the following two key behaviors concerning hardware control versus software control:

Asserting the CYSPP pin externally always triggers automatic CYSPP. This occurs even if you have disabled the profile in software.

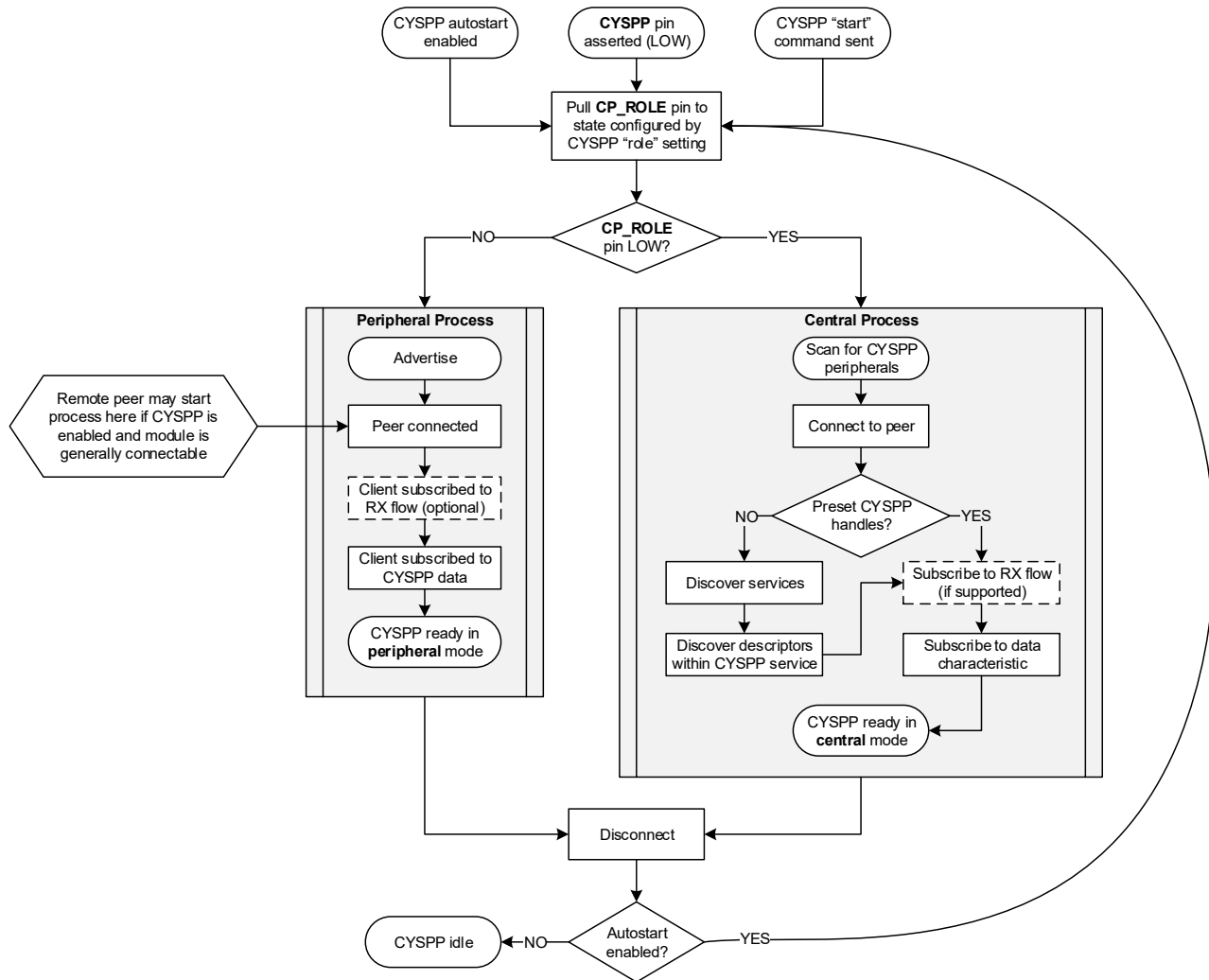
CYSPP data mode (where the API is suppressed, and all serial data is channeled to the remote peer) ultimately depends on the state of the CYSPP pin. EZ-Serial firmware platform for Ezurio Vela IF820 series module pulls this pin to the appropriate logic level based on internal CYSPP state changes when CYSPP is enabled, but you can override the pulled state with an external host or hardware design feature.

## : CYSPP configuration and pin relationship

CYSPP pin state	CYSPP “enable” value in configuration	CYSPP operation
Floating (assumed default)	Disabled	<b>Inactive.</b> All advertising, scanning, connections, GATT subscriptions, GATT transfers, and so on, occur via API commands and events. CYSPP GATT structure is not visible to a remote Client.
	Enabled	Idle until start. When started via the <code>p_cyspp_start (.CYSPPSTART, ID=10/2)</code> API command, the module begins advertising. API events (boot, stage changes, connections, etc.) are <b>visible</b> over UART until the CYSPP data connection is opened between the local device and remote peer. The CYSPP pin is pulled LOW when this occurs, at which point the API is suppressed and the serial interface may be used only for CYSPP data pipe. This mode continues until the remote host disconnects or unsubscribes.
	Autostart (factory default)	Automatic. Same behavior as in the “Enabled” case, except that CYSPP operation begins automatically at boot time and restarts upon disconnection.
Externally driven HIGH (de-asserted)	Disabled	<b>Inactive.</b> All advertising, connections, GATT subscriptions, GATT transfers, and so on occur via API commands and events. CYSPP GATT structure is not visible to a remote Client.
	Enabled	Idle until start, command mode retained. When started via the <code>p_cyspp_start (.CYSPPSTART, ID=10/2)</code> API command, module begins advertising. API events (BOOT, stage changes, connections, etc.) are visible over UART. API communication continues throughout the process; CYSPP data from the remote host is never raw/transparent unless the host asserts the CYSPP pin.
	Autostart	Automatic. The same behavior as in the “Enabled” case, except that CYSPP operation begins automatically at boot time and restarts upon disconnection. API events continues to be visible while <b>CYSPP</b> pin is de-asserted (HIGH).
Externally driven LOW (asserted)	Doesn’t matter	Active regardless of firmware configuration. Automatic advertising begins at boot time. API events (boot, state changes, connections, etc.) are not be visible over UART, because API communication is always suppressed when CYSPP pin is asserted.

#### 2.4.7.9 CYSPP state machine

0 shows the way EZ-Serial firmware platform for Ezurio Vela IF820 series module manages CYSPP operation, depending on firmware configuration and the logic states of the CYSPP and CP\_ROLE pins.



#### : CYSPP state machine

Note that EZ-Serial firmware platform for Ezurio Vela IF820 series module pulls the CP\_ROLE pin to the state configured by the `p_cyspp_set_parameters (.CYSPPSP, ID=10/3)` API command, but if the host or hardware design drives it to a different state, CYSPP will operate in the pin-defined state and not the firmware-defined state.

#### 2.4.8 Bluetooth® classic SPP

EZ-Serial firmware platform for Ezurio Vela IF820 series module also supports Bluetooth® SPP service profile. See our Migration Application Notes for additional information on how to setup SPP Incoming and Outgoing SPP Connections.

- Application Note - EZ-Serial Incoming Legacy SPP Connections (BT 2.0) - Vela IF820 Series
- Application Note - EZ-Serial Outgoing Legacy SPP Connections (BT 2.0) - Vela IF820 Series
- Application Note - EZ-Serial Incoming SSP SPP Connections (BT2.1+) - Vela IF820 Series
- Application Note - EZ-Serial Outgoing SSP SPP Connections (BT2.1+) - Vela IF820 Series
- Application Note - EZ-Serial UWTerminalX Commands - Vela IF820 Series

## 2.5 Configuration settings, storage, and protection

The EZ-Serial firmware platform for Ezurio Vela IF820 series module provides methods to customize its many built-in functions. It is important to understand how these settings are stored and changed in different contexts to avoid unexpected behavior.

### 2.5.1 Factory, boot, and runtime settings

Ezurio Vela IF820 series module implements three different “layers” of configuration data, each of which serves a unique purpose 0 describes each type of configuration storage in detail.

#### : Configuration setting storage layers

Layer	Details
Factory (FLASH)	<p>Description:</p> <p>Factory-level settings are hard-coded into the firmware image and stored in flash and cannot be changed independently by the user. They are used for runtime-level settings until/unless customized boot-level values exist. Using the <code>system_factory_reset (/RFAC, ID=2/5)</code> API command reverts to these values.</p> <p>Content:</p> <p>These values contain only platform configuration settings, but not custom GATT structure definitions or value data.</p> <p>Data retention during chipset reset: YES</p> <p>These values <u>are retained</u> upon power cycles and chipset reset conditions.</p>
Boot (FLASH)	<p>Description:</p> <p>Boot-level settings are set by the user and stored in flash, and applied to the runtime-level area for active use when the module boots. (If no customized boot-level settings have been set by the user, the factory-level settings are applied instead upon first boot.) These values can be modified using API commands, and they are erased when performing a factory reset.</p> <p>Content:</p> <p>These values contain both platform configuration settings and any custom GATT structure definitions. Actual GATT characteristic values such as those written by a remote Client are not included in this data.</p> <p>Data retention during chipset reset: YES</p> <p>These values <u>are retained</u> during power cycles and chipset reset conditions.</p>
Runtime (RAM)	<p>Description:</p> <p>Runtime-level settings are used as the active configuration set that controls EZ-Serial firmware platform for Ezurio Vela IF820 series module’s behavior at all times, with a few exceptions as noted in the “Automatic” section below. API commands that set or get configuration values access this layer of configuration data unless explicitly noted otherwise.</p> <p>Content:</p> <p>These values contain platform configuration settings, custom GATT structure definitions, and GATT characteristic values written from a remote Client.</p> <p>Data retention during chipset reset: NO</p>

Layer	Details
	These values <u>are not retained</u> during power cycles and chipset reset conditions. Any runtime settings or GATT database structure definitions should be written to flash with the relevant API command(s) before performing a reset.
Automatic (RAM)	<p>Description:</p> <p>Automatic settings are set by the firmware based on detected external behavior, and EZ-Serial firmware platform for Ezurio Vela IF820 series module uses these values to augment the settings in the runtime configuration block. Currently, only one setting falls into this category:</p> <p>API parse mode (binary or text mode depending on initial packet byte)</p> <p>Content:</p> <p>These values contain a very limited subset of auto-detected configuration settings, and do not include most configuration data or any GATT structure or value data.</p> <p>Data retention during chipset reset: NO</p> <p>These values <u>are not retained</u> during power cycles and chipset reset conditions.</p> <p>Data retention during DFU: NO</p> <p>These values <u>are not retained</u> during the OTA process, which involves a chipset reset prior to image transfer.</p>

### 2.5.2 Saving runtime settings in flash

Storing settings in flash memory is critical to allow predictable, long-term customized behavior without needing to reconfigure each time. EZ-Serial firmware platform for Ezurio Vela IF820 series module provides two ways to accomplish this:

1. Use the `system_store_config (/SCFG, ID=2/4)` API command to write all current runtime-level settings to the boot-level configuration. This applies a snapshot of the current configuration to flash in one step. This method should be used if you are unsure which settings have changed between boot-level and runtime-level values, or if you want to test out a new set of options before making them permanent.
2. Set the “flash” memory scope bit in the binary command packet header when writing new configuration values with relevant commands, or append the ‘\$’ character to command names in text mode. This is simpler than the alternative if you know exactly which settings need to be changed, since it does not require the final use of the `system_store_config (/SCFG, ID=2/4)` API command afterward.

Note that while the flash memory scope bit may be used with any command; doing so is relevant only for commands that either read or write configuration values directly. For other commands, these flags will be silently ignored. See the API reference material in API protocol reference for details.

To ensure the longest flash memory life, writes to flash should be as infrequent as possible in production-ready designs. Settings that must be changed frequently should be modified in RAM and only written to flash when required. Note that the internal chipsets used in the Ezurio Vela IF820 series modules that run EZ-Serial firmware platform for Ezurio Vela IF820 series module have a minimum flash endurance rating of 100,000 cycles.

### 2.5.3 Protected configuration settings

A small number of configuration values have the potential to put the module into a state where it is no longer possible to communicate over the serial interface as intended. To help avoid this potential problem, a few settings are classified as protected. This means that the values of the settings must be changed at the runtime level only (RAM) before they may be applied to the boot-level (flash) area. Currently, only one command affects protected settings: `system_set_uart_parameters (STU, ID=2/25)`.

The changes that are most likely to cause an unintended communication lockout are serial transport reconfigurations, such as selecting a baud rate that is not supported by the host. To store new values in flash for protected configuration settings, you must either send the same command twice with the flash memory scope bit/character used only the second time, or else use the `system_store_config (/SCFG, ID=2/4)` API command to write all runtime-level settings to the boot level after first setting the new value in RAM only. This forces the flash write to occur using the new configuration, which can only occur if communication is still possible.

## 2.6 Finding related material

This guide refers to firmware images and example source code files that must be accessed separately from this document.

### 2.6.1 Latest EZ-Serial firmware platform for Ezurio Vela IF820 series module image

You can find the latest available EZ-Serial firmware platform for Ezurio Vela IF820 series module image files on the Ezurio Github repository for Vela IF820 firmware: [https://github.com/LairdCP/Vela\\_IF820\\_Firmware/releases](https://github.com/LairdCP/Vela_IF820_Firmware/releases)

### 2.6.2 Latest host API protocol library

You can find the latest host API protocol library source code examples here: [https://github.com/LairdCP/Vela\\_IF820\\_Firmware](https://github.com/LairdCP/Vela_IF820_Firmware)

### 2.6.3 Comprehensive API reference

While this guide contains many specific functional examples, these are not intended to provide a full reference to all possible functionality provided by the API. See API protocol reference of this document for detailed material concerning the API structure and protocol.

## 3 Operational examples

EZ-Serial firmware platform for Ezurio Vela IF820 series module provides a great platform on which you can build a wide variety of BLE applications. This section describes many common operations that you can experiment with or combine to create the behavior needed for your application.

### 3.1 System setup examples

These examples demonstrate basic platform behavior and configuration of the system.

**Note:** The first example (see [Identifying the running firmware and BLE stack version](#)) provides low-level detail and explanation of some API protocol formatting features, while all other examples assume a basic understanding of the mechanics of the protocol and will only show example snippets in text format. For details on the API methods used in each case and the binary equivalents of each command, response, and event, see API protocol reference.

#### 3.1.1 Identifying the running firmware and BLE stack version

The EZ-Serial firmware platform for Ezurio Vela IF820 series module firmware, BLE stack, and protocol version details can be obtained from the API event generated at boot time, or on demand using an API command.

##### 3.1.1.1 Getting version details from boot event

Capture and process the `system_boot (BOOT, ID=2/1)` API event that occurs when the module is powered on or reset. This event includes the application version, stack version, protocol version, boot cause, and unique Bluetooth®MAC address.

If the protocol parser/generator is in text mode (factory default), the `system_boot (BOOT, ID=2/1)` API event looks like this:

```
@E,003B,BOOT,E=01000215,S=030200FA,P=0102,H=05,C=01,A=00A050421A63
```

If the protocol parser is in binary mode, this event will be similar to that shown below, expressed in hexadecimal notation:

##### : Version Details from Boot Event

Header	Payload	Checksum
80 12 02 01	19 00 01 01 35 00 03 03 03 01 05 01 63 1A 42 50 A0 00	3D

To simplify manual interpretation in this guide, individual parameters within the payload are separately underlined.

**Note:** In text mode, multi-byte integer data is expressed in big-endian notation, while in binary mode, multi-byte integer data is transmitted in little-endian order.

The payload data in the event text/binary examples shown above is described in [0](#).

**: Payload detail for boot event**

Text code	Text data	Binary data	Details	Interpretation
E	"01040C0C"	0C 0C 04 01	EZ-Serial firmware platform for Ezurio Vela IF820 series module version	Version 1.1.0 build 25 (0x19)
S	"03010000"	03 01 00 00	BLE stack version	Version 3.3.0 build 53 (0xFA)
P	"0104"	04 01	API protocol version	Version 1.3
H	"F1"	F1	Hardware ID	CY20820
C	"00"	00	Cause for boot event	Not supported
A	"00A050421A63"	63 1A 42 50 A0 00	MAC address	00:A0:50:42:1A:63

**3.1.1.2 Getting version details on demand**

Use the **system\_query\_firmware\_version (/QFV, ID=2/6)** API command to request version details at any time. The response to this command contains the same initial information in the **system\_boot (BOOT, ID=2/1)** API event, but it does not include the boot cause or the module's Bluetooth®MAC address.

The text-mode response to this API command is as shown below:

```
@R,002C,/QFV,0000,E=01040C0C,S=03010000,P=0104,H=F1
```

The binary-mode response packet is as shown below:

**: Version Details on Demand**

Header	Payload	Checksum
C0 0D 02 06	<u>06 00 00 1C 02 01 01 55 03 02 02 03 01 B1</u>	<u>9F</u>

To simplify manual interpretation in this guide, individual parameters within the payload are separately underlined.

**3.1.2 Changing the serial communication parameters**

Use the **system\_set\_uart\_parameters (STU, ID=2/25)** API command to reconfigure the serial interface used for host communication. This command affects protected settings, and therefore the protected setting must be applied in RAM first before it can be written to flash.

All data entered via text mode must be expressed in hexadecimal notation. 0 lists common baud rates and their hexadecimal equivalents:

**: Common UART baud rates and hex equivalents**

Baud rate	Hex equivalent
9,600	2580
14,400	3840
19,200	4B00
28,800	7080
38,400	9600

Baud rate	Hex equivalent
57,600	E100
115,200 (default)	1C200
230,400	38400
460,800	70800
921,600	E1000

**Note:** EZ-Serial firmware platform for Ezurio Vela IF820 series module supports non-standard baud rates not listed in **0**, and should remain below 3% clock error due to the use of an internal fractional clock divider. While this is within the tolerance level required by many UART interfaces, you should measure the actual bit timing with an oscilloscope or logic analyzer to verify that the baud rate is operating within required tolerance for your host device.

**: Example 1 - Set UART to 38400 baud, even parity, flow control enabled, and store in flash**

Direction	Text content	Binary content	Effect
TX→	STU,B=9600,A=0,C=0,F=1,D=8,P=0,S=1	C0 0A 02 19 00 96 00 00 00 00 01 08 00 01 1E	Set new UART parameters (RAM only) – “38400” decimal is “9600” hex
←RX	@R,0009,STU,0000	C0 02 02 19 00 00 76	Response indicates success

Change host UART parameters to match new settings here before sending additional data

TX→	STU\$,B=9600,A=0,C=0,F=1,D=8,P=0,S=1	D0 0A 02 19 00 96 00 00 00 00 01 08 00 01 2E	Write UART settings to flash
←RX	@R,000A,STU\$,0000	D0 02 02 19 00 00 86	Response indicates success

**: Example 2 - Set UART to 115200 baud, no parity, flow control disabled, and store in RAM only**

Direction	Text content	Binary content	Effect
TX→	"STU,B=1C200,A=0,C=0,F=0,D=8,P=0,S=1	C0 0A 02 19 00 C2 01 00 00 00 00 08 00 01 4A	Apply new UART parameters
←RX	@R,0009,STU,0000	C0 02 02 19 00 00 76	Response indicates success



### 3.1.3 Changing device name and appearance

Use the `gap_set_device_name` (SDN, ID=4/15) API command to set a new friendly device name at any time, and the `gap_set_device_appearance` (SDA, ID=4/17) API command to set a new appearance value.

EZ-Serial firmware platform for Ezurio Vela IF820 series module supports different device names for BLE and BT Classic communication. By default, the BT Classic Device Name starts with "BT" as a suffix.

EZ-Serial firmware platform for Ezurio Vela IF820 series module uses the device name and appearance to populate the GAP service's name and appearance characteristic values in the GATT database. If EZ-Serial firmware platform for Ezurio Vela IF820 series module is allowed to automatically manage the advertisement and scan response data content (default behavior), then it also includes up to 29 bytes of the device name in the scan response packet. (The limit of 29 bytes is due to a BLE specification limit on the maximum scan response payload, which is 31 bytes; the other two bytes are needed for the field length and field type values that are part of the device name field.)

**Note:** EZ-Serial firmware platform for Ezurio Vela IF820 series module limits the device name length to 64 bytes to minimize internal SRAM requirements.

Using EZ-Serial firmware platform for Ezurio Vela IF820 series module's special macro codes, described in section Macro Definitions you can enter a single text string which is expanded internally to include module-specific values—in this case, the Bluetooth® MAC address. This is shown in 0.

The device appearance value is a 16-bit field made up of a 10-bit and 6-bit subfield. Allowed values are defined by the Bluetooth® SIG and can be found at [developer.bluetooth.org](https://developer.bluetooth.org).

Changes made to the device name and appearance values take effect immediately. They are written to the local GATT characteristics for these two values (always present), and the device name is updated in the scan response packet if user-defined advertisement content has not been enabled with the `gap_set_adv_parameters` (SAP, ID=4/23) API command.

#### : Example 3 - Set device name with partial MAC address incorporation

Direction	Text content	Binary content	Effect
TX→	SDN\$,N=EZ-Serial %M4:%M5:%M6	D0 16 04 0F 15 45 5A 2D 53 65 72 69 61 6C 20 25 4D 34 3A 25 4D 35 3A 25 4D 36 5C	Set new device name in flash using 4th, 5th, and 6th MAC bytes (module-specific)
←RX	@R,000A,SDN\$,0000	D0 02 04 0F 00 00 7E	Response indicates success

This configured name results in an actual name of "EZ-Serial 1A:21:D3" assuming that the module in use has a MAC address of 20:73:7A:1A:21:D3).

#### : Example 4 - Set device appearance to "Generic Computer" (0x0080)

Direction	Text content	Binary content	Effect
TX→	SDA\$,A=0080	D0 02 04 11 80 00 00	Set new appearance value in flash
←RX	@R,000A,SDA\$,0000	D0 02 04 11 00 00 80	Response indicates success

### 3.1.4 Changing output power

Use the `system_set_tx_power` (STXP, ID=2/21) API command to set a new radio transmit power level. The argument to this command is not the dBm value directly, but rather a set of predefined values representing a fixed range. STXP can set the max power of BR/EDR/BLE individually. The power array is 3\*8 bytes for power level. User can use STXP,P=0,D=xx to change power level array. But it does not suggest change it. User can use STXP,P=1~8 to set power level. The default is 7.

Current power level array of 20820 chip in default is:

BR: {-2,0,2,4,6,8,10,12},

EDR: {-2,0,2,4,6,8,10,12},

BLE: {-2,0,2,4,6,8,10,10},

**: Supported TX power output options**

Argument value	Power level BR	Power level EBR	Power level BLE	Comments
1	-2 dBm	-2 dBm	-2 dBm	
2	0 dBm	0 dBm	0 dBm	
3	2 dBm	2 dBm	2 dBm	
4	4 dBm	4 dBm	4 dBm	
5	6 dBm	6 dBm	6 dBm	
6	8 dBm	8 dBm	8 dBm	
7	10 dBm	10 dBm	10 dBm	Default Value
8	12 dBm	12 dBm	10 dBm	

**Note:** This table is just for example. The predefined table can be changed as BR,EBR and BLE.

Changes to the configured output power will take effect immediately.

**: Example 5 - Set output power to -8 dBm**

Direction	Text content	Binary content	Effect
TX→	STXP,P=3	C0 01 02 15 03 74	Set new TX power (RAM only)
←RX	@R,000A,STXP,0000	C0 02 02 15 00 00 72	Response indicates success

### 3.1.5 Managing sleep states

EZ-Serial firmware platform for Ezurio Vela IF820 series module manages transitions between active and sleep states according to the LP\_Mode pin logic level and the system sleep level configurations. It chooses the mode requiring the lowest safe power consumption according to the current operational state and configuration, including transitioning into sleep mode between BT Classic and BLE radio events.

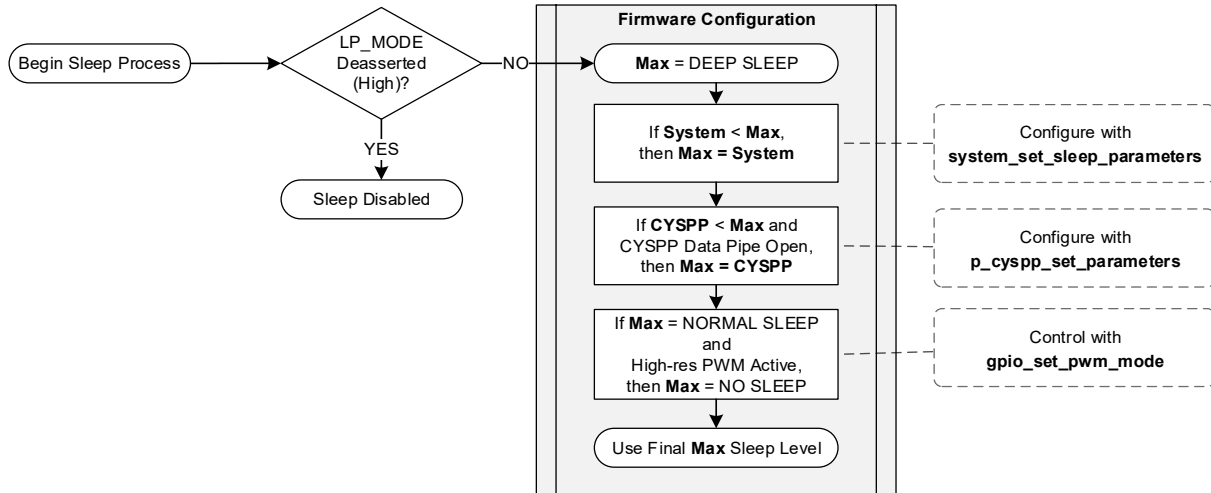
0 provides a high-level summary of the four power states used by the platform.

**: EZ-Serial firmware platform for Ezurio Vela IF820 series module power states**

Power mode	Current range (typical), Vdd = 3.3 V	Description
Active	5 mA to 7 mA	CPU and all peripherals are active. All functionality is possible with no delay.
Sleep	0.23 mA to 3.x mA	In PDS Mode, UART may have missed communication. However, it can still receive data from BLE or BT link.
Deep Sleep	1.2 µA	In HID-Off Mode, no active resources are available until the FW restarts.

EZ-Serial firmware platform for Ezurio Vela IF820 series module uses the maximum allowed sleep level based on combined data from the system-wide sleep setting, CYSPP data mode sleep setting (if CYSPP data mode is active), PWM output state, and LP\_MODE pin state.

0 describes the sleep level determination logic.



### : EZ-Serial firmware platform for Ezurio Vela IF820 series module sleep state behavior

In outline form, the sleep state logic follows this process:

1. If the **LP\_MODE** pin is de-asserted to high, the module will remain in Active mode, otherwise select the lowest value (0 = no sleep, 1 = normal sleep, 2 = deep sleep) from the following methods to configure the system-wide sleep setting:
  - a. The system sleep level configured with **system\_set\_sleep\_parameters** (SSLP, ID=2/19) API command.
  - b. The CYSPP-specific sleep level configured with the **p\_cyspp\_set\_parameters** (.CYSPPSP, ID=10/3) API command, if the CYSPP data pipe is open (connected and in CYSPP data mode).
  - c. No sleep if high-resolution PWM output is enabled with the **gpio\_set\_pwm\_mode** (SPWM, ID=9/11) API command.

**Note:** EZ-Serial firmware platform for Ezurio Vela IF820 series module does not allow changes to the sleep level hierarchical order. For example, if CYSPP sleep level is “1” (sleep) but system-wide sleep is level “0” (no sleep), then the system-wide setting will override the CYSPP setting because it is a lower value. EZ-Serial firmware platform for Ezurio Vela IF820 series module will always select the lowest applicable value for the current operational state.

#### 3.1.5.1 Configuring the system-wide sleep level

Configure the system-wide sleep level using the **system\_set\_sleep\_parameters** (SSLP, ID=2/19) API command. When sleep is not prevented by de-asserting the **LP\_MODE** pin, this value is the first “default” sleep level limit applied when calculating which sleep mode to use.

Active PWM output limits the effective sleep level in any state to no sleep (value = 0). If the CYSPP data pipe is open (connected and in CYSPP data mode), then the CYSPP-specific sleep level may further limit the effective maximum sleep level. 0 shows how EZ-Serial firmware platform for Ezurio Vela IF820 series module determines the sleep level to use.

EZ-Serial firmware platform for Ezurio Vela IF820 series modules allows normal sleep (value = 1) as the factory default system-wide sleep level and sets **LP\_MODE** to high by default to provide a simpler out-of-the-box UART communication experience. However, you can change this to allow Deep Sleep to improve average current consumption.

#### : Example 6. Change system-wide sleep level to Deep Sleep

Direction	Text content	Binary content	Effect
TX→	SSLP,L=2	C0 01 02 13 02 71	Set new system sleep level to “Deep Sleep”
←RX	@R,000A,SSLP,0000	C0 02 02 13 00 00 70	Response indicates success

In normal sleep mode the module cannot receive commands; the host needs proper use of the **LP\_MODE** pin as described in section **Preventing sleep with the LP\_MODE pin** before transmitting the command.

### 3.1.5.2 Configuring the CYSPP data mode sleep level

Use the `p_cyspp_set_parameters (.CYSPPSP, ID=10/3)` API command to set the CYSPP data mode sleep level. When sleep is enabled by the LP\_MODE pin, the CYSPP data mode sleep level value is the second limit to determine which sleep mode to use. The system-wide sleep level takes precedence over the CYSPP sleep level. Furthermore, PWM output limits the sleep level in any state to no sleep (value = 0), regardless of other settings. 0 shows how EZ-Serial firmware platform for Ezurio Vela IF820 series module determines the sleep level to use.

#### : Example 7 - Limit CYSPP-specific sleep level to normal sleep

Direction	Text content	Binary content	Effect
TX→	.CYSPPSP,E=2,G=0,C=0131,L=0,R=0,M=0,P=1, S=1,F=2	C0 13 0A 03 02 00 31 01 00 00 00 00 00 00 00 00 00 00 00 00 01 01 02 B1	Set new CYSPP sleep level to "normal sleep"
←RX	@R,000E,.CYSPPSP,0000	C0 02 0A 03 00 00 68	Response indicates success

### 3.1.5.3 Preventing sleep with the LP\_MODE pin

De-assert the LP\_MODE control pin (HIGH) to prevent the module from sleeping under any circumstances. Properly asserting and de-asserting this pin surrounding host-to-module UART transmissions provides efficient power consumption while still allowing normal sleep at all other times.

### 3.1.5.4 Managing host and module sleep simultaneously

In applications that include both an external host MCU and a Bluetooth® module, typically both components need to sleep to save as much power as possible. The DATA\_READY pin is asserted (LOW) whenever there is UART data in the output buffer and not yet fully clocked out of the module. Using this pin as the wakeup signal for the MCU is the recommended way to allow the module to alert the host whenever some interaction needs to occur.

In certain situations, an external MCU takes may take so long to wake that it loses the first few bits or bytes of the incoming UART data from the module. If the host needs extra time to wake and RTS/CTS flow control is unavailable on the host MCU, you can still enable UART flow control in EZ-Serial firmware platform for Ezurio Vela IF820 series module with the `system_set_uart_parameters (STU, ID=2/25)` API command and then control the module's CTS pin from a host GPIO. When CTS is held in the de-asserted (HIGH) state, the module waits to send any outgoing UART data. The host can complete its wakeup process and then assert (LOW) the module's CTS pin to allow serial data transmission when ready.

True flow control support on the host MCU is not necessary in this case, and you can leave the module's RTS pin disconnected. However, you must still enable flow control within EZ-Serial firmware platform for Ezurio Vela IF820 series module to accomplish this. **Flow control with EZ-Serial firmware platform for Ezurio Vela IF820 series module is not enabled by default.**

To summarize the complete cycle:

1. Host sets the module CTS pin HIGH to prevent UART transmission.
2. Host enables the DATA\_READY pin falling-edge interrupt.
3. Host puts the CPU to sleep.
4. Module asserts (LOW) its DATA\_READY pin when relevant activity occurs.
5. Host CPU wakes up.
6. Host sets the module CTS pin LOW to allow UART transmission.
7. Module transmits data to the host for processing.

### 3.1.6 Performing a factory reset

You can perform a factory reset using `system_factory_reset (/RFAC, ID=2/5)` API command.

EZ-Serial firmware platform for Ezurio Vela IF820 series module generates the `system_factory_reset_complete (RFAC, ID=2/3)` API event immediately after erasing all settings, and before performing the final module reset to boot to the factory default state. The platform generates this event using the previously configured parser and transport mode. While this event is typically not processed by an external host during a hardware-triggered factory reset, it helps to verify the intended flow when controlling the module via software.

After the reset completes, the `system_boot (BOOT, ID=2/1)` API event occurs.

To trigger a factory reset over the serial interface, use the `system_factory_reset (/RFAC, ID=2/5)` API command.

#### : Example 8 - Perform a factory reset

Direction	Text content	Binary content	Effect
TX→	/RFAC	C0 00 02 05 60	Trigger factory reset
←RX	@R,000B,/RFAC,0000	C0 02 02 05 00 00 62	Response indicates success
←RX	@E,0005,RFAC	80 00 02 03 1E	Event indicates factory reset completed
Short delay while chipset reset and boot process occurs, ~150 ms			
←RX	@E,003B,BOOT,E=01010700,S=05020016,P=0103,H=D1,C=00,A=ECBF17C8FD39	80 12 02 01 00 07 01 01 16 00 02 05 03 01 D1 00 39 FD C8 17 BF EC E9	

## 3.2 Cable replacement examples with CYSPP

EZ-Serial firmware platform for Ezurio Vela IF820 series module's CYSPP implementation provides a simple way to use a BLE connection to manage a bidirectional stream of serial data. Both ends of the connection must support CYSPP, including the ability to either provide or make use of the CYSPP GATT structure for data flow. The EZ-Serial firmware platform for Ezurio Vela IF820 series module can operate as either a GAP peripheral and CYSPP server device (typical when communicating with a smartphone) or as a GAP central and CYSPP client device (typical when communicating with a second module running EZ-Serial firmware platform for Ezurio Vela IF820 series module).

See section [Using CYSPP mode](#) for a description of how CYSPP mode behaves generally and how it affects API communication.

### 3.2.1 Getting started in CYSPP mode with zero custom configuration

The factory default configuration enables the CYSPP profile in "auto-start" mode. With this configuration, the module begins advertising or scanning as soon as it has power, depending on the state of the CP\_ROLE pin.

1. Connect the **CP\_ROLE** pin to either logic LOW (central) or logic HIGH (peripheral) to define the role used. If left floating, EZ-Serial firmware platform for Ezurio Vela IF820 series module will use the role configured in firmware using the [p\\_cyspp\\_set\\_parameters \(.CYSPPSP, ID=10/3\)](#) API command. EZ-Serial firmware platform for Ezurio Vela IF820 series module uses the peripheral role with factory default settings.
2. Connect the module's **UART\_RX** pin to the external host's **UART\_TX** pin.
3. Connect the module's **UART\_TX** pin to the external host's **UART\_RX** pin.
4. *OPTIONAL*: Assert (LOW) the **CYSPP** pin to force CYSPP data mode in hardware, preventing API usage or output.
5. Apply power to the module, or reset it with the hardware reset pin.
6. If you have asserted (LOW) the **CYSPP** pin externally:
  - a. Monitor the **CONNECTION** pin to detect when the remote peer has connected and GATT data subscription is complete.
  - b. Once the **CONNECTION** pin goes low, you can send and receive data from the host to the remote peer over the module's serial connection.
7. If the CP\_ROLE pin is left floating:
  - a. Wait for the [p\\_cyspp\\_status \(.CYSPP, ID=10/1\)](#) API event to appear with the LSB set indicating the data channel is ready. The final status event should appear as one of the following:

@E, 000C, .CYSPP, S=05 (running in peripheral role)  
  
@E, 000C, .CYSPP, S=35 (running in central role)
  - b. Send and receive data as desired.

**Note:** If you externally de-assert (HIGH) the **CYSPP** pin, then EZ-Serial firmware platform for Ezurio Vela IF820 series module will never enter CYSPP data mode. The remote peer may use CYSPP on its end normally, but all data transfers and status updates will appear on the local EZ-Serial firmware platform for Ezurio Vela IF820 series module end as API events to be processed normally. mode even if a remote peer has connected and all CYSPP mode data pipe preparations have completed.

### 3.2.1.1 Starting CYSPP out of the box in peripheral mode

EZ-Serial firmware platform for Ezurio Vela IF820 series module's factory default configuration automatically starts CYSPP operation in the Peripheral role after booting. To establish a CYSPP data pipe, simply scan and connect from a remote device, then subscribe to RX flow control (optional) and the desired acknowledged or unacknowledged data characteristic as described in section [Sending and receiving data in CYSPP data mode](#).

A second EZ-Serial firmware platform for Ezurio Vela IF820 series module running in CYSPP Central/Client mode will perform all required client-side steps automatically. EZ-Serial firmware platform for Ezurio Vela IF820 series module shows all GATT events relating to CYSPP setup until the CYSPP data pipe is fully opened.

#### : Example 9 - Complete boot and CYSPP connection process in peripheral mode

Direction	Text content	Binary content	Effect
←RX	@E,0076,BOOT,E=01040C0C,S=03010000,P=0104,H=F1,C=00,A=FB56F7AC98D7,F=EZ-Serial-VELA_IF820_INT V1.4.12.12 Sep 11 2023 10:17:21	80 12 02 01 1C 02 01 01 55 03 02 02 03 01 B1 00 D3 21 1A 7A 73 20 7A	Boot event
←RX	@E,000E,ASC,S=00,R=03	80 02 04 02 01 03 25	CYSPP-triggered advertisement started
←RX	@E,0035,C,C=40,A=00A050422A0F,T=00,I=0006,L=0000,O=0064,B=00	80 0F 04 05 40 0F 2A 42 50 A0 00 00 06 00 00 00 64 00 00 46	Connection established with remote device
←RX	@E,001A,W,C=40,H=0015,T=00,D=0200	80 08 05 02 40 15 00 00 02 00 02 00 81	Remote client writes [02 00] to Client Characteristic Configuration Descriptor (CCCD) for RX flow control to enable indications from that characteristic.
←RX	@E,000C,.CYSPP,S=04	80 01 0A 01 04 29	CYSPP status update (0x04):  0x04: Subscribed to RX flow control
←RX	@E,001A,W,C=40,H=0012,T=00,D=0100	80 08 05 02 40 12 00 00 02 00 01 00 7D	Remote client writes [01 00] to CCCD for unacknowledged data to enable notifications from that characteristic.
←RX	@E,000C,.CYSPP,S=05	80 01 0A 01 05 2A	CYSPP status update (0x05):  0x04: Subscribed to RX flow control 0x01: Subscribed to unacknowledged data

Host may now send data to the module for delivery to the remote peer, received data comes from peer.

### 3.2.1.2 How to start CYSPP out of the box in central mode

Starting CYSPP client mode with factory default settings also requires no reconfiguration, since CYSPP mode will start automatically. However, you must assert (LOW) the CP\_ROLE pin at boot time or set G=1 using the `p_cyspp_set_parameters (.CYSPPSP, ID=10/3)`.

*: Example 1 - Complete boot and CYSPP connection process in central mode*

Direction	Content	Effect
←RX	@E,003B,BOOT,E=0101011A,S=03030035,P=0103, H=05,C=01,A=00A050E3835F	Boot event
←RX	@E,000E,SSC,S=02,R=03	CYSPP-triggered scan started
←RX	@E,0062,S,R=00,A=00A050421650,T=00,S=D1,B=00,D=020106 110700A10C2000089A9EE21115A13333336507 FF310100000000	Scan result (advertisement fields separated for easier interpretation)
←RX	@E,000E,SSC,S=00,R=03	CYSPP-triggered scan stopped
←RX	@E,0035,C,C=04,A=00A050421650,T=00, I=0006,L=0000,O=0064,B=00	Connection established with remote device
←RX	@E,0029,DR,C=04,H=0001,R=0007,T=2800,P=00,U=0018	GATT discovery result (0x1800)
←RX	@E,0029,DR,C=04,H=0008,R=000B,T=2800,P=00,U=0118	GATT discovery result (0x1801)
←RX	@E,0045,DR,C=04,H=000C,R=0015,T=2800,P=00, U=00A10C2000089A9EE21115A133333365	GATT discovery result (CYSPP service)
←RX	@E,0010,RPC,C=04,R=060A	Remote procedure complete
←RX	@E,0029,DR,C=04,H=000C,R=0000,T=2800,P=00,U=0028	GATT discovery result (service declaration)
←RX	@E,0029,DR,C=04,H=000D,R=0000,T=2803,P=00,U=0328	GATT discovery result (characteristic declaration)
←RX	@E,0045,DR,C=04,H=000E,R=0000,T=0000,P=00, U=01A10C2000089A9EE21115A133333365	GATT discovery result (CYSPP ack'd data)
←RX	@E,0029,DR,C=04,H=000F,R=0000,T=2902,P=00,U=0229	GATT discovery result (configuration descriptor)
←RX	@E,0029,DR,C=04,H=0010,R=0000,T=2803,P=00,U=0328	GATT discovery result (characteristic declaration)
←RX	@E,0045,DR,C=04,H=0011,R=0000,T=0000,P=00, U=02A10C2000089A9EE21115A133333365	GATT discovery result (CYSPP unack'd data)
←RX	@E,0029,DR,C=04,H=0012,R=0000,T=2902,P=00,U=0229	GATT discovery result (configuration descriptor)
←RX	@E,0029,DR,C=04,H=0013,R=0000,T=2803,P=00,U=0328	GATT discovery result (characteristic declaration)
←RX	@E,0045,DR,C=04,H=0014,R=0000,T=0000,P=00, U=03A10C2000089A9EE21115A133333365	GATT discovery result (CYSPP RX flow control)
←RX	@E,0029,DR,C=04,H=0015,R=0000,T=2902,P=00,U=0229	GATT discovery result (configuration descriptor)
←RX	@E,0010,RPC,C=04,R=0000	Remote descriptor discovery complete

Direction	Content	Effect
←RX	@E,000C,.CYSPP,S=10	CYSPP status update (0x10): 0x10: CYSPP peer support verified
←RX	@E,0017,WRR,C=04,H=0015,R=0000	Remote server acknowledged the write operation that enabled indications on RX flow control characteristic.
←RX	@E,000C,.CYSPP,S=14	CYSPP status update (0x14): 0x10: CYSPP peer support verified 0x04: Subscribed to RX flow control
←RX	@E,0018,D,C=04,H=0014,S=02,D=00	Remote server pushes a "flow allowed" value via an indication from the RX flow control characteristic.
←RX	@E,0017,WRR,C=04,H=0012,R=0000	Remote server acknowledged write operation which enabled notifications on unacknowledged data characteristic
←RX	@E,000C,.CYSPP,S=15	CYSPP status update (0x15): 0x10: CYSPP peer support verified 0x04: Subscribed to RX flow control 0x01: Subscribed to unacknowledged data

Host may now send data to the module for delivery to the remote peer, received data comes from peer

### 3.3 GAP peripheral examples

GAP Peripheral operation is one of the most common use cases for BLE designs because it is usually the simplest way to communicate with a smartphone operating as a Central device.

The Bluetooth® specification defines different types of roles for the devices on each end of a BLE link:

#### Link layer

Master – Device that initiates a connection (always GAP Central)

Slave – Device that accepts a connection (always GAP Peripheral)

#### GAP layer

Central – Device that initiated a connection (always LL master)

Peripheral – Device that accepted a connection (always LL slave)

Broadcaster – Device that is advertising in a non-connectable state

Observer – Device that is scanning without initiating a connection

#### GATT layer

Client – device which accesses data from a remote GATT Server

Server – device which provides Attribute data to be accessed remotely

Link layer roles are defined when a connection is initiated based on which side initiates the connection.

The GAP layer provides four different roles, two of which involve connections (Central and Peripheral) and two of which are connectionless (Broadcaster and Observer). The link layer and GAP layer roles are closely related, particularly when a connection is involved.

The GATT layer role is independent of other behavior. A single device may even perform GATT duties in both the client and server roles. A common example of this is an iOS device providing the Apple Notification Center Service as a GATT Server, even though it is connected to a Peripheral device and acting as a GATT Client to that device.

EZ-Serial firmware platform for Ezurio Vela IF820 series module only supports slave link layer role, Peripheral or Broadcaster GAP roles, and GATT Server functionality.



### 3.3.1 Advertising as peripheral device

Advertising is the BLE activity which allows scanning devices to observe and connect to Peripherals. It is required for a connection to be initiated, but it may also be done in a non-connectable way (called "broadcasting"). EZ-Serial firmware platform for Ezurio Vela IF820 series module supports non-connectable broadcasting even while connected.

EZ-Serial firmware platform for Ezurio Vela IF820 series module gives you full control over when and how to advertise by using the `gap_start_adv (/A, ID=4/8)` API command and the `gap_set_adv_parameters (SAP, ID=4/23)` API command.

When the advertising state changes, the `gap_adv_state_changed (ASC, ID=4/2)` API event occurs. This event includes the new state as well as a code showing the reason why the state changed.

**Note:** If you do not have any automatic advertisement timeout set, then advertisements continue until you explicitly stop them, or a remote device initiates a connection.

#### : Example 10 - Start advertising with custom parameters

Direction	Text content	Binary content	Effect
TX→	/A,M=02,T=03,C=07,H=0030, D=001E,L=0800,O=003C, F=00,A=000000000000,Y=0	C0 13 04 08 02 03 07 30 00 1E 00 00 08 3C 00 00 00 00 00 00 00 00 16	Begin advertising with custom arguments
←RX	@R,0008,/A,0000	C0 02 04 08 00 00 67	Response indicates success
←RX	@E,000E,ASC,S=03,R=00	80 02 04 02 03 00 24	Event indicates advertising state changed to "active"

### 3.3.2 Stopping advertising as a peripheral device

To explicitly stop advertising, use the `gap_stop_adv (/AX, ID=4/9)` API command, or open a connection to the module from a remote BLE Central device.

#### : Example 1 - Stop advertising

Direction	Text content	Binary content	Effect
TX→	/AX	C0 00 04 09 66	Stop advertising
←RX	@R,0009,/AX,0000	C0 02 04 09 00 00 68	Response indicates success
←RX	@E,000E,ASC,S=00,R=00	80 02 04 02 00 00 21	Event indicates advertising state changed to "inactive" due to user request

### 3.3.3 Customizing advertisement and scanning response data

You can customize the content of the main advertisement payload and scan response payload with the `gap_set_adv_data (SAD, ID=4/19)` and `gap_set_sr_data (SSRD, ID=4/21)` API commands, respectively.

**Note:** If you intend to use user-defined advertisement content, you must explicitly enable this in the advertisement parameters. Normally, the EZ-Serial firmware platform for Ezurio Vela IF820 series module manages the content in the advertisement and scans response packets automatically based on the platform configuration, including the device name and the profiles that are enabled. If you set custom content but do not configure EZ-Serial firmware platform for Ezurio Vela IF820 series module to use that content, advertisement and scan response payloads remain automatically managed.

Key features and requirements for customizing data:

Each advertisement and scan response packet payloads may have a maximum of 31 bytes. This is a BLE specification limit.

Advertisement data in both packets should follow the correct [Length, Type, Value...] format required by the Bluetooth® specification.

Malformed data within advertisements can prevent proper scanning by remote devices. The **Length** value does not include itself but does include the **Type** byte and all bytes in the remaining **Value** data.

Each packet may contain as many fields as will fit in 31 bytes. Place multiple fields one right after the other with no special separator. Since each field begins with a "length" value, a scanning device is always able to properly identify the end of each field.

Advertisement packets include the Bluetooth® connection address (public or random) outside of the payload data. This does not count towards the 31-byte limit.

The main advertisement packet is always transmitted while advertising. It typically includes things like connectable flags, important supported service UUIDs, and a custom manufacturer data field. Place any data that is critical for the remote device to see inside the main advertisement packet.

The scan response packet is transmitted only when a remote device is performing an active scan. During an active scan, the scanning device send a scan request to any discovered advertising device immediately after receiving the main advertisement packet. The scan response packet typically includes the friendly name of the advertising device, and occasionally also includes transmit power, more manufacturer data, or other useful but less critical data that a remote scanning device may not need to see.

Detailed information on approved field types and their intended contents can be found the Bluetooth® specification.

0 lists the most commonly used fields;

#### : Common advertisement field types

Type	Description	Value
0x01	Flags field – 1 byte of data	1 byte (bitfield)
0x02	Partial list of 16-bit UUIDs for supported GATT services	2*N bytes (UUIDs)
0x03	Complete list of 16-bit UUIDs for supported GATT services	2*N bytes (UUIDs)
0x04	Partial list of 32-bit UUIDs for supported GATT services	4*N bytes (UUIDs)
0x05	Complete list of 32-bit UUIDs for supported GATT services	4*N bytes (UUIDs)
0x06	Partial list of 128-bit UUIDs for supported GATT services	16*N bytes (UUIDs)
0x07	Complete list of 128-bit UUIDs for supported GATT services	16*N bytes (UUIDs)
0x08	Shortened local name	0-29 bytes (Text string)
0x09	Complete local name	0-29 bytes (Text string)
0x0A	TX power level	1 byte (dBm as signed integer)
0xFF	Manufacturer data	3-29 bytes (company ID + data)

EZ-Serial firmware platform for Ezurio Vela IF820 series module does not validate advertisement or scan response payload content, and the underlying BLE stack has only limited validation on the Flags field. You must ensure that any customized data within either of these packets is correctly formatted. While the module will transmit whatever payload data is configured, scanning devices may not correctly identify your device if the data is malformed or missing (especially the Flags field).

The stack requires that the Flags field, if present, must have the final two bits set so that they match the Discovery Mode setting used when starting advertisements. For BLE-only devices that do not support "classic" BR/EDR Bluetooth® behavior, this means that the flags byte will almost always be one of these three values:

0x04: Non-discoverable/broadcast-only (common for beacon-only devices)

0x05: Limited discoverable

0x06: General discoverable (most common for connectable devices)

See [gap\\_start\\_adv \(/A, ID=4/8\)](#) API command for additional reference on discoverable modes.

0 provides examples for reference.

*: Examples of well formed advertisement fields*

Byte content	Field description
02 01 06	Length: 2 bytes Type: Flags (0x01) Value: LE General Discoverable Mode, BT Classic Not Supported
05 02 09 18 0D 18	Length: 5 bytes Type: Complete list of 16-bit UUIDs for supported GATT services (0x02) Value: 0x1809 (Health Thermometer), 0x180D (Heart Rate)
07 08 57 69 64 67 65 74	Length: 7 bytes Type: Shortened local name (0x08) Value: "Widget"
09 FF 31 01 AA BB CC DD EE FF	Length: 9 bytes Type: Manufacturer data (0xFF) Value: Company ID = 0x0131 (Cypress Semiconductor, an Infineon Technologies company) Data = [AA BB CC DD EE FF]

These four example fields require 25 bytes when combined, including each of the four Length values. The bytes can be placed in a single advertisement packet if desired:

02 01 06 05 02 09 18 0D 18 07 08 57 69 64 67 65 74 09 FF 31 01 AA BB CC DD EE FF

Here, the shortened name is included in the same packet as the more critical information. This is uncommon, but not prohibited. The name typically goes in the scan response packet because it cannot fit into the advertisement packet, but any field may be in any location if the scanning device knows what to expect.

*: Example 2 - Set custom advertisement and scan response data*

Direction	Text content	Binary content	Effect
TX→	SAP, M=02, T=03, C=07, H=0030, D=001E, L=0800, O=003C, F=02, A=00000000000000, Y=0	C0 13 04 17 02 03 07 30 00 1E 00 00 08 3C 00 02 00 00 00 00 00 00 00 27	Enable user-defined advertisement and scan response content
←RX	@R, 0009, SAP, 0000	C0 02 04 17 00 00 7	Response indicates success
TX→	SAD, D=020106060209180D18	C0 0A 04 13 09 02 01 06 06 02 09 18 0D 18 DA C0 02 04 13 00 00 72	Set new advertisement content (RAM only), Flags and 16-bit UUID fields
←RX	@R, 0009, SAD, 0000	C0 02 04 13 00 00 72	Response indicates success
TX→	SSRD, D=0708576964676574	C0 09 04 15 08 07 08 57 69 64 67 65 74 F6	Set new scan response content (RAM only), Complete local name field
←RX	@R, 000A, SSRD, 0000	C0 02 04 15 00 00 74	Response indicates success

: Example 3 - Set advertisement and scan response data to value similar to factory defaults

Direction	Text content	Binary content	Effect
TX→	SAP, M=02, T=03, C=07, H=0030, D=001E, L=0800, O=003C, F=02, A=000000000000, Y=0	C0 13 04 17 02 03 07 30 00 1E 00 00 08 3C 00 02 00 00 00 00 00 00 00 27	Enable user-defined advertisement and scan response content
←RX	@R, 0009, SAP, 0000	C0 02 04 17 00 00 76	Response indicates success
TX→	SAD, D=020106110700a10c2000089a9e e21115a133333365	C0 16 04 13 15 02 01 06 11 07 00 A1 0C 20 00 08 9A 9E E2 11 15 A1 33 33 33 65 70	Set new advertisement content (RAM only)
←RX	@R, 0009, SAD, 0000	C0 02 04 13 00 00 72	Response indicates success
TX→	SSRD, D=1309455a2d53657269616c204 5333a38333a3546	C0 15 04 15 14 13 09 45 5A 2D 53 65 72 69 61 6C 20 45 33 3A 38 33 3A 35 46 D5	Set new scan response content (RAM only)
←RX	@R, 000A, SSRD, 0000	C0 02 04 15 00 00 74	Response indicates success

## 3.4 GAP central examples

Running as a GAP Central allows you to scan for and connect to remote Peripheral devices. You can also operate as a GAP Observer by scanning without any subsequent connection attempts. For further discussion of various link-layer, GAP, and GATT roles, see the material at the beginning of Section [GAP peripheral examples](#).

### 3.4.1 How to scan peripherals

Use the `gap_start_scan (/S, ID=4/10)` API command to begin scanning for devices. Scanning is not required before initiating a connection, but doing so helps to identify potential connection targets or ensure that known or compatible Peripherals are nearby and connectable

**Note:** If you do not have any automatic scan timeout set, scanning will continue until you explicitly stop it. Scanning **will not** automatically resume when a connection is terminated unless CYSPP is enabled in the central role. Otherwise, you must implement this behavior in your application logic as needed.

**Note:** You must stop scanning before you can initiate an outgoing connection to a remote peer. Requesting a connection with `gap_connect (/C, ID=4/1)` while scanning will result in an error.

In text mode, all arguments to the `gap_start_scan (/S, ID=4/10)` API command are optional. Any supplied arguments will be used only for the immediate scan started as a result of the command, while any omitted arguments will fall back to the values configured by the `gap_set_scan_parameters (SSP, ID=4/25)` API command. You can see these values at any time by using the `gap_get_scan_parameters (GSP, ID=4/26)` API command.

After you start scanning, EZ-Serial firmware platform for Ezurio Vela IF820 series module will begin generating `gap_scan_result (S, ID=4/4)` API events each time a new advertisement packet is seen from a remote device. The same advertising device will generate multiple scan results until duplicate filtering is enabled in the scan parameters.

### Passive vs. Active Scanning:

During a **passive** scan, EZ-Serial firmware platform for Ezurio Vela IF820 series module will not send scan requests to devices to ask for the “follow-up” scan response packet. In this mode, each device generates only one event for each detected advertisement packet. Passive scans use less power on average because the transmitter remains inactive and the receiver is not intentionally re-activated for a second time for the same device.

During an **active** scan, EZ-Serial firmware platform for Ezurio Vela IF820 series module sends a scan request to obtain additional information from the remote Peripheral. In this mode, the BT stack may generate two events for each device detected during a scan. However, the remote device may not send the scan response packet, or the local device may not receive it due to adverse RF conditions, so a second scan result event is not guaranteed. Active scans use more power than passive scans, and result in brief transmission bursts in between receive operations.

*Due to the precise timing required by the BLE protocol and the way active scans behave, a large number of actively scanning devices in the same vicinity can result in none of the scanning devices successfully obtaining a scan response from an advertising device. If two or more scanning devices transmit a scan request on the same channel within the same ~150  $\mu$ s window immediately after the main advertisement packet, the advertising device will not be able to parse the request and will not send a response to either device. This unlikely but possible issue does not occur while performing a passive scan.*

#### : Example 1 - Start passive scanning with preconfigured default parameters

Direction	Content	Effect
TX→	/S	Begin scanning with preconfigured defaults
←RX	@R,0008,/S,0000	Response indicates success
←RX	@E,000E,SSC,S=01,R=00	Event indicates scanning state has changed to "active" due to user request
←RX	@E,0052,S,R=00,A=00A050E3835E,T=00,S=D1,B=00,D=0201061107CA366D7D5BCC0288B14DE541D9FF652F	Event indicates scan result from 00:A0:50:E3:83:5E, normal adv packet, RSSI -47 dBm (0xB1), Flags field and 128-bit UUID

#### : Example 2 - Start 5-second active scan with duplicate filtering enabled

Direction	Content	Effect
TX→	/S,M=2,A=1,D=1,O=5	Begin "observation" scanning, active mode, 5-second timeout, duplicate filter enabled
←RX	@R,0008,/S,0000	Response indicates success
←RX	@E,000E,SSC,S=01,R=00	Event indicates scanning state has changed to "active" due to user request
←RX	@E,0052,S,R=00,A=00A050E3835E,T=00,S=D1,B=00,D=0201061107CA366D7D5BCC0288B14DE541D9FF652F	Event indicates scan result from 00:A0:50:E3:83:5E, adv packet, RSSI -47 dBm (0xB1), Flags field and 128-bit UUID
←RX	@E,004E,S,R=04,A=00A050E3835E,T=00,S=D1,B=00,D=1209426C7565666C6F772037383A46353A4236	Event indicates scan result from 00:A0:50:E3:83:5E, scan response packet, RSSI -47 dBm, Local name field
←RX	@E,000E,SSC,S=00,R=02	Event indicates scanning state has changed to "stopped" due to configured timeout (5 seconds)

### 3.4.2 How to stop scanning for peripheral devices

To explicitly stop scanning, use the `gap_stop_scan (/SX, ID=4/11)` API command, or initiate a connection request to a remote device using the `gap_connect (/C, ID=4/1)` API command.

*It is possible for additional `gap_scan_result (S, ID=4/4)` API events to occur between a successful response to the `gap_stop_scan` command and the `gap_scan_state_changed` event (SSC in text mode), due to the brief amount of time that it takes the stack to process the request and change states. Ensure that your application logic will not fail in this case.*

**: Example 1 - Stop scanning**

Direction	Content	Effect
TX→	/SX	Stop scanning
←RX	@R, 0009, /SX, 0000	Response indicates success
←RX	@E, 000E, SSC, S=00, R=00	Event indicates scanning state has changed to "inactive" due to user request

**3.4.3 How to connect to a peripheral device**

Use the `gap_connect (/C, ID=4/1)` API command to initiate a connection to a remote device based on its Bluetooth® connection address. The Bluetooth® connection address (also commonly referred to as a MAC address) is made up of the 6-byte device address and a 1-byte value indicating the address type. To initiate a connection, the module must be in a disconnected state (not advertising, scanning, connecting, or connected).

**Note:** At this time, the Infineon Bluetooth® stack supports one active connection at a time. To transfer data to and from multiple devices quickly, you must establish and tear down connections in rapid succession. With a fast advertisement interval on peripheral devices and a fast connection interval while connected, it is possible to perform many connect-transfer-disconnect cycles per second.

Addresses may be either public or random. Public addresses do not change, while random addresses change on some period determined by the device employing privacy measures (typically at least every few minutes). The use of random addresses, also called private addresses, reduces the possibility of passive profiling by a remote device. For example, iOS devices always use random addressing for BLE operations. EZ-Serial firmware platform for Ezurio Vela IF820 series module supports both types and uses public address by default.

When a BLE device initiates a connection request, it does not immediately transmit anything. Rather, it must first scan until it receives a connectable advertisement packet from the target device. This is why a Peripheral device must be in an advertising state to accept a connection. The full connection process includes the following steps:

1. Target Peripheral device is advertising in a connectable state.
2. Central device begins scanning for advertisements packets from target Peripheral device.
3. Central device detects advertisement and initiates a connection request to a target Peripheral device.
4. Target Peripheral device receives connection request and responds with connection response.
5. Connection is successfully established at this point.

The API command used to initiate a connection includes arguments for scan parameters, as scanning is the first operation that the stack must perform on the GAP Central device during a connection process.

**: Example 1 - Connect to a remote device using default connection parameters**

Direction	Content	Effect
TX→	/C, A=00A050E3835E	Initiate connection
←RX	@R, 000D, /C, 0000, C=00	Response indicates success
←RX	@E, 0030, C, C=01, A=00A050E3835E, T=00, I=0006, L=0000, O=0064, B=0	Event indicates connection opened

**3.4.4 How to cancel a pending connection to a peripheral device**

Use the `gap_cancel_connection (/CX, ID=4/2)` API command to cancel a pending outgoing connection request. This applies only when the connection is not yet open and you have not received the `gap_connected (C, ID=4/5)` API event. If you need to close an open connection, use the `gap_disconnect (/DIS, ID=4/5)` API command.

**: Example 1 - Cancel a pending connection to a remote device**

Direction	Content	Effect
TX→	/CX,	Cancel pending connection

Direction	Content	Effect
←RX	@R,0009,/CX,0000	Response indicates success
←RX	@E,0010,DIS,C=00,R=0900	Event indicates connection canceled

### 3.4.5 How to disconnect from a peripheral device

Use the [gap\\_disconnect \(/DIS, ID=4/5\)](#) API command to close an active connection to a remote device. This applies only when the connection is already fully established; this should not be used to cancel a pending outgoing connection. In that case, use the [gap\\_cancel\\_connection \(/CX, ID=4/2\)](#) API command.

#### : Example 1 - Disconnect from a remote device

Direction	Content	Effect
TX→	/DIS	Disconnect from peer
←RX	@R,000A,/DIS,0000	Response indicates success
←RX	@E,0010,DIS,C=01,R=0900	Event indicates connection closed, reason=0x0900 (unknown reason)

## 3.5 GATT server examples

BLE data transfer operations between two connected devices most often occur through the GATT layer, with a server on one side and a client on the other side. The GATT Server makes use of a pre-defined attribute structure, which the client may remotely discover and use as needed. The GATT Server defines what data is available and how it may be accessed and has limited ability to push data to the client if the client has subscribed to receive these types of updates.

### 3.5.1 Defining custom local GATT services and characteristics

EZ-Serial firmware platform for Ezurio Vela IF820 series module implements a dynamic GATT structure that can be modified at runtime and stored in flash. Note that the structure itself and values stored within data characteristics (other than default values defined when creating new entries) are stored in RAM only, and is not stored to flash until explicitly calling command `gatts_store_db (/SGDB, ID=5/4)` or `system_store_config (/SCFG, ID=2/4)`.

EZ-Serial firmware platform for Ezurio Vela IF820 series module implements a dynamic GATT structure that can be modified at runtime. The structure and its values are stored in RAM only when be created or modified. The structure and its values will not be stored to flash until you call the command explicitly, `gatts_store_db (/SGDB, ID=5/4)` or `system_store_config (/SCFG, ID=2/4)`.

The EZ-Serial firmware platform for Ezurio Vela IF820 series module contains a few pre-defined GATT elements in the factory default configuration. EZ-Serial firmware platform for Ezurio Vela IF820 series module requires these GATT elements for correct operation, and the elements cannot be removed or modified. However, additional structural elements are entirely customizable.

A GATT structure is fundamentally made up of individual attributes, each of which has a unique numeric handle, a UUID that is 16 bits, 32 bits, or 128 bits wide, and a value container. Attribute handles start at 1 and may go up to 0xFFFF (65535). No two attributes may have the same handle. Ezurio Vela IF820 series module internally uses three structures to store individual attribute:

- `gatts_db[]`: An array of GATT entry structures containing the fixed-length portion of each entry (type, permissions, length, and the 16-bit length prefix value from the data array).
- `gatts_db_const_data[]`: An array of UINT8 bytes containing the variable-length portion of each entry (the payload from the data array).
- `gatts_external_data[]`: An array of UINT8 bytes containing the writable values of each entry.

EZ-Serial firmware platform for Ezurio Vela IF820 series module provides the `gatts_create_attr (/CAC, ID=5/1)` API command to create a new custom attribute, which in the EZ-Serial firmware platform for Ezurio Vela IF820 series module takes the following arguments:

```
uint8 type
uint8 permissions
uint16 length
longuint8a data
```

The first six bytes of this packed structure (through the 16-bit length prefix on data) is a match for the GATT entry structure. Any payload data in the data structure goes in the constant data pool instead.

To use the custom Local GATT Services and Characteristics correctly, you must have some prior knowledge of correct GATT structures, especially in the case of a characteristic declaration which includes additional metadata beyond just the value attribute's UUID. The following demonstrates how you would use this command to add one service, one characteristic, one characteristic value, and one CCCD:

// syntax: /CAC,type, permissions, length, data[]	
/CAC, T=00, P=02, L=12, D=0028D0002D121E4B0FA4994ECEB531F40579	1.
/CAC, T=00, P=02, L=15, D=0328301F00BD1DA299E625588CD94201630D12BF9F	2.
/CAC, T=01, P=89, L=40, D=1122334455667788	3.
/CAC, T=00, P=0A, L=04, D=02290000	4.

1. Create a service descriptor, which contains the 0x2800 structural UUID, 0xD0 properties byte, the 16-bit attribute handle corresponding to the value attribute, and 128-bits UUID. Note that the attribute handle is automatically generated and EZ-Serial firmware platform for Ezurio Vela IF820 series module requires the value attribute to be present immediately after the declaration.
2. Create a characteristic descriptor, which contains the 0x2803 structural UUID, 0x030 properties byte, the 16-bit attribute handle corresponding to the value attribute, and 128-bit UUID.
3. Create a characteristic value descriptor, which contains the initial value 0x1122334455667788 and reserve 0x40 length room to contain value.
4. Create a CCCD, which contains 0x9202 structural UUID and a value 0x0000.

**Modifications to the custom GATT structure require flash write operations, which can potentially disrupt BLE connectivity. Therefore, you should only make changes to the GATT database while there is no active BLE connection to avoid the possibility of a connection loss.**



### 3.5.1.1 Understanding custom GATT limitations

The dynamic GATT implementation in EZ-Serial firmware platform for Ezurio Vela IF820 series module contains some built-in entries to provide required EZ-Serial firmware platform for Ezurio Vela IF820 series module functionality, leaving the remaining space available for custom entries. Each entry is assembled by three structures:

1. GATT attribute entry: Containing the fixed-length portion of each entry (type, permissions, length, and the 16-bit length prefix value from the data array)
2. GATT data array: Containing the variable-length portion of each entry
3. GATT external read/write data: Containing the writable values of each entry

0 lists each relevant value on both platforms:

#### : Dynamic GATT Structural Limitations

Category	Built-in	Vela IF820	
		Total	Available
SRAM reserved for GATT attribute entries	21*6 = 126 bytes	128*6=768 bytes	107*6=642 bytes
SRAM reserved for GATT data arrays	38+87 = 125 bytes	768 bytes	643 bytes
SRAM reserved for GATT external data arrays	107 bytes	512 bytes	405
Flash memory room reserved for storing GATT data base	358 bytes	2048 bytes	1690 bytes

Attempting to create a new custom attribute which exceeds any of the bounds listed in 0 will generate an error result indicating the nature of the limitation. See section **Error codes** for details.

### 3.5.1.2 Building custom services and characteristics

The GATT database is made up of one or more primary services. Each primary service has a service declaration (UUID 0x2800) and includes one or more characteristics. Each characteristic has a characteristic declaration (UUID 0x2803) and a value attribute (any UUID not in the above list), and often has additional characteristic-related descriptors in the 0x2900 range.

UUIDs indicate the purpose of each attribute, but may be (and often are) repeated through the complete database. For example, a database containing three services will contain three separate attributes which all have the UUID 0x2800, which is the official "Primary Service Declaration" UUID defined by the Bluetooth® SIG. 0 lists notable pre-defined structural definition UUIDs from the Bluetooth® SIG.

#### : Bluetooth® SIG structural UUIDs

UUID	Description
0x2800	Primary Service Declaration
0x2801	Secondary Service Declaration
0x2802	Include Declaration
0x2803	Characteristic Declaration
0x2900	Characteristic Extended Properties
0x2901	Characteristic User Description
0x2902	Client Characteristic Configuration
0x2903	Server Characteristic Configuration
0x2904	Characteristic Format
0x2905	Characteristic Aggregate Format

For more details on these and other official identifiers, see the [Bluetooth® SIG website](https://www.bluetooth.com).

When defining GATT elements at runtime, you must enter each attribute in the correct order based on the desired structure. Any entries that do not conform to the correct order requirement will be rejected with a validation error. The only case where a validation warning is allowed is when you define a new service or characteristic declaration and have not yet entered the subsequent attributes which must follow. You can use the `gatts_validate_db (/VGDB, ID=5/3)` API command at any time to perform an integrity check on the current GATT structure to see whether additional attributes are expected.

The required order for each complete characteristic definition (declaration, value, and optional descriptors) is dictated by the internal BLE stack as follows:

#### : Required characteristic attribute order

Order	UUID	Description	Required
#1	0x2803	Characteristic Declaration	Yes
#2	< custom >	Characteristic Value	Yes
#3	0x2900	Characteristic Extended Properties	No
#4	0x2901	Characteristic User Description	No
#5	0x2902	Client Characteristic Configuration	No
#6	0x2903	Server Characteristic Configuration	No
#7	0x2904	Characteristic Format	No
#8	0x2905	Characteristic Aggregate Format	No

Any optional attributes may be omitted if all provided attributes are supplied in the order mentioned in 0.

For details on how to use custom GATT creation API commands to add support for Bluetooth® SIG official services such as Device Information, Health Thermometer, and others, see section [Adopted Bluetooth SIG GATT profile structure snippets](#) and the API reference material for `gatts_create_attr (/CAC, ID=5/1)`.

#### 3.5.1.3 Choosing correct GATT permissions

It is critical to use correct permissions when defining any custom GATT structural elements. See section [Adopted Bluetooth SIG GATT profile structure snippets](#) for example definitions, and you may notice certain patterns. Here are the recommended guidelines for the most common entries:

```

Service declarations (UUID = 0x2800)
    PERM = 0x02
PERM_READABLE
    Characteristic properties are not needed because they do not apply.
Characteristic declarations (UUID = 0x2803)
    PERM = 0x02
PERM_READABLE
    Characteristic properties = < actual properties >
Characteristic value attributes (type = 0x0000)
    PERM = 0x89
PERM_VARIABLE_LENGTH
PERM_WRITE_REQ
PERM_SERVICE_UUID_128 (if this service has a 128-bit UUID)
    Characteristic properties value is not required because it has been defined in previous characteristic declarations.
Characteristic user description attributes (UUID = 0x2901)
    PERM = 0x02
PERM_READABLE
    Characteristic properties = 0x02 (read)
Client characteristic configuration attributes (UUID = 0x2902)
    PERM = 0x0A
PERM_READABLE
PERM_WRITE_REQ
    Characteristic properties = 0x0A (read + write)

```

In general, structural elements such as service and characteristic declarations should be read-only, but should have no particular security restrictions on them. This ensures that a connected client is able to discover the database structure correctly, even if additional security is

required to execute read and/or write operations on the characteristic value attributes. Some Android devices are known to have problems during discovery if the declaration descriptors themselves have extra security requirements.

### 3.5.2 Listing local GATT services, characteristics, and descriptors

Listing the local GATT structure can be helpful in certain cases, even though it is typically the remote GATT structure that requires discovery. This is especially true because you can dynamically change the local GATT structure at runtime. EZ-Serial firmware platform for Ezurio Vela IF820 series module provides three commands for local discovery.

#### 3.5.2.1 Discovering local GATT services

Use the `gatts_discover_services (/DLS, ID=5/6)` API command to obtain a list of services in the local GATT database.

##### : Example 4 - Local GATT service discovery with factory default structure (no custom attributes)

Direction	Text content	Binary content	Effect
TX→	/DLS, B=0, E=0	C0 04 05 06 00 00 00 00 68	Request to discover all local services
←RX	@R, 0011, /DLS, 0000, C=0003	C0 04 05 06 00 00 03 00 6B	Response indicates success, 3 records to follow
←RX	@E, 0024, DL, H=0001, R=0007, T=2800, P=00, U=0018	80 0A 05 01 01 00 07 00 00 28 00 02 00 18 73	Service 0x1800, start=1, end=7
←RX	@E, 0024, DL, H=0008, R=000B, T=2800, P=00, U=0118	80 0A 05 01 01 00 07 00 00 28 00 02 00 18 73	Service 0x1801, start=8, end=11 (0x0B)
←RX	@E, 0040, DL, H=000C, R=0015, T=2800, P=00, U=00A10C2000089A9EE21115A133333365	80 18 05 01 0C 00 15 00 00 28 00 10 00 A1 0C 20 00 08 9A 9E E2 11 15 A1 33 33 33 65 44	Service 0x6533...A100, start=12 (0x0C), end=21 (0x15)

#### 3.5.2.2 Discovering local GATT characteristics

Use the `gatts_discover_characteristics (/DLC, ID=5/7)` API command to obtain a list of characteristics in the local GATT database.

##### : Example 5 - Local GATT characteristic discovery with factory default structure (no custom attributes)

Direction	Text content	Binary content	Effect
TX→	/DLC, B=0, E=0, S=0	C0 06 05 07 00 00 00 00 00 00 00 6B	Request to discover all local characteristics
←RX	@R, 0011, /DLC, 0000, C=0007	C0 04 05 07 00 00 07 00 70	Response indicates success, 7 records to follow
←RX	@E, 0024, DL, H=0002, R=0003, T=2803, P=02, U=002A	80 0A 05 01 02 00 03 00 03 28 02 02 00 2A 87	Char 0x2A00, decl handle=2, value handle=3, perm=0x02
←RX	@E, 0024, DL, H=0004, R=0005, T=2803, P=02, U=012A	80 0A 05 01 02 00 03 00 03 28 02 02 00 2A 87	Char 0x2A01, decl handle=4, value handle=5, perm=0x02
←RX	@E, 0024, DL, H=0006, R=0007, T=2803, P=02, U=042A	80 0A 05 01 04 00 05 00 03 28 02 02 01 2A 8C	Char 0x2A04, decl handle=6, value handle=7, perm=0x02

Direction	Text content	Binary content	Effect
←RX	@E,0024,DL,H=0009,R=000A,T=2803,P=22,U=052A	80 0A 05 01 09 00 0A 00 03 28 22 02 05 2A BA	Char 0x2A05, decl handle=9, value handle=10, perm=0x22
←RX	@E,0040,DL,H=000D,R=000E,T=2803,P=28, U=01A10C2000089A9EE21115A133333365	80 18 05 01 0D 00 0E 00 03 28 28 10 01 A1 0C 20 00 08 9A 9E E2 11 15 A1 33 33 33 65 6A	Char 0x6533...A101, decl handle=13, value handle=14, perm=0x28
←RX	@E,0040,DL,H=0010,R=0011,T=2803,P=14, U=02A10C2000089A9EE21115A133333365	80 18 05 01 10 00 11 00 03 28 14 10 02 A1 0C 20 00 08 9A 9E E2 11 15 A1 33 33 33 65 5D	Char 0x6533...A102, decl handle=16, value handle=17, perm=0x14
←RX	@E,0040,DL,H=0013,R=0014,T=2803,P=20, U=03A10C2000089A9EE21115A133333365	80 18 05 01 13 00 14 00 03 28 20 10 03 A1 0C 20 00 08 9A 9E E2 11 15 A1 33 33 33 65 70	Char 0x6533...A103, decl handle=19, value handle=20, perm=0x20

### 3.5.2.3 Discovering local GATT descriptors

Use the `gatts_discover_descriptors (/DLD, ID=5/8)` API command to obtain a list of descriptors in the local GATT database.

#### Example 6 - Local GATT descriptor discovery with factory default structure (no custom attributes)

Direction	Text content	Binary content	Effect
TX→	/DLD, B=0, E=0, S=0, C=0	C0 08 05 08 00 00 00 00 00 00 00 00 6E	Request to discover all local descriptors
←RX	@R, 0011, /DLD, 0000, C=0015	C0 04 05 08 00 00 15 00 7F	Response indicates success, 21 records to follow
←RX	@E, 0024, DL, H=0001, R=0007, T=2800, P=00, U=0028	80 0A 05 01 01 00 07 00 00 28 00 02 00 28 83	UUID 0x2800 (Primary Service), start=1, end=7
←RX	@E, 0024, DL, H=0002, R=0003, T=2803, P=02, U=0328	80 0A 05 01 02 00 03 00 03 28 02 02 03 28 88	UUID 0x2803 (Characteristic), decl=2, value handle=3
←RX	@E, 0024, DL, H=0003, R=0000, T=0000, P=02, U=002A	80 0A 05 01 03 00 00 00 00 00 02 02 00 2A 5A	UUID 0x2A00 (Device Name), handle=3, perm=0x02
Additional records omitted for brevity			
←RX	@E, 0024, DL, H=000C, R=0015, T=2800, P=00, U=0028	80 0A 05 01 0C 00 15 00 00 28 00 02 00 28 9C	UUID 0x2800 (Primary Service), start=12, end=21
←RX	@E, 0024, DL, H=000D, R=000E, T=2803, P=28, U=0328	80 0A 05 01 0D 00 0E 00 03 28 28 02 03 28 C4	UUID 0x2803 (Characteristic), decl=13, value handle=14, perm=0x28
←RX	@E, 0040, DL, H=000E, R=0000, T=0000, P=28, U=01A10C2000089A9EE21115A133333365	80 18 05 01 0E 00 00 00 00 00 28 10 01 A1 0C 20 00 08 9A 9E E2 11 15 A1 33 33 33 65 32	UUID 0x6533...A101 (Acknowledged Data Characteristic:), handle=14, perm=0x28
←RX	@E, 0024, DL, H=000F, R=0000, T=2902, P=0A, U=0229	80 0A 05 01 0F 00 00 00 02 29 0A 02 02 29 9A	UUID 0x2902 (CCCD), handle=15, perm=0x0A
←RX	@E, 0024, DL, H=0010, R=0011, T=2803, P=14, U=0328	80 0A 05 01 10 00 11 00 03 28 14 02 03 28 B6	UUID 0x2803 (Characteristic), decl=16, value handle=17, perm=0x28
←RX	@E, 0040, DL, H=0011, R=0000, T=0000, P=14, U=02A10C2000089A9EE21115A133333365	80 18 05 01 11 00 00 00 00 00 14 10 02 A1 0C 20 00 08 9A 9E E2 11 15 A1 33 33 33 65 22	UUID 0x6533...A102 (Unacknowledged Data Characteristic), handle=17, perm=0x28

Direction	Text content	Binary content	Effect
←RX	@E, 0024, DL, H=0012, R=0000, T=2902, P=0A, U=0229	80 0A 05 01 12 00 00 00 02 29 0A 02 02 29 9D	UUID 0x2902 (CCCD), handle=18, perm=0x0A
←RX	@E, 0024, DL, H=0013, R=0014, T=2803, P=20, U=0328	80 0A 05 01 13 00 14 00 03 28 20 02 03 28 C8	UUID 0x2803 (Characteristic), decl=19, value handle=20, perm=0x28
←RX	@E, 0040, DL, H=0014, R=0000, T=0000, P=20, U=03A10C2000089A9EE21115A133333365	80 18 05 01 14 00 00 00 00 00 20 10 03 A1 0C 20 00 08 9A 9E E2 11 15 A1 33 33 33 65 32	UUID 0x6533...A103 (RX Flow Characteristic), handle=20, perm=0x20
	@E, 0024, DL, H=0015, R=0000, T=2902, P=0A, U=0229	80 0A 05 01 15 00 00 00 02 29 0A 02 02 29 A0	UUID 0x2902 (CCCD), handle=21, perm=0x0A

### 3.5.3 Reading and writing local GATT attribute values

Read and write local GATT values using the `gatts_read_handle (/RLH, ID=5/9)` and `gatts_write_handle (/WLH, ID=5/10)` API commands, respectively.

It is always possible to locally read any attribute, and locally write any attribute that supports the write operation. Some attributes, such as service and characteristic declarations, contain only constant data (stored in flash) that is not meant to be modified with a typical GATT write command. If you intend to change the structure of the GATT database itself, use the `gatts_create_attr (/CAC, ID=5/1)` and `gatts_delete_attr (/CAD, ID=5/2)` API commands.

#### 3.5.3.1 Reading local GATT data

You can read the value of a local attribute using the `gatts_read_handle (/RLH, ID=5/9)` API command. EZ-Serial firmware platform for Ezurio Vela IF820 series module will return the current value in the response.

#### Example 7. Read local device name characteristic

Direction	Text content	Binary content	Effect
TX→	/RLH, H=3	C0 02 05 09 03 00 6C	Read attribute with handle = 3
←RX	@R, 0031, /RLH, 0000, D=455A2D53657269616C2031 413A32313A4433	C0 16 05 09 00 00 12 00 45 5A 2D 53 65 72 69 61 6C 20 31 41 3A 32 31 3A 44 33 9B	Response indicates success, hex data is "EZ- Serial 1A:21:D3"

#### 3.5.3.2 Writing local GATT data

You can write the value of a local attribute using the `gatts_write_handle (/WLH, ID=5/10)` API command. This command replaces any existing data in the attribute and is limited by the maximum length of the attribute in the GATT structure.

Writing data does not automatically push a notification or indication packet to a remote client, even if the client has subscribed to either of these types of pushed updates. See section [Notifying and indicating data to a remote client](#) for details on how to push data.

### : Example 8 - Write "ABCD" at beginning of local Device Name characteristic

Direction	Text content	Binary content	Effect
TX→	/WLH, H=3, D=41424344	C0 08 05 0A 03 00 04 00 41 42 43 44 81	Write "ABCD" (hex) into attribute with handle = 3
←RX	@R, 000A, /WLH, 0000	C0 02 05 0A 00 00 6A	Response indicates success
TX→	/RLH, H=3	C0 02 05 09 03 00 6C	Read attribute with handle = 3 to verify
←RX	@R, 0031, /RLH, 0000, D=41424344	C0 08 05 09 00 00 04 00 41 42 43 44 7D	Response indicates success, data shows expected value

### 3.5.4 Notifying and indicating data to a remote client

Notifying and indicating allow a server to push updates to a client without the client specifically requesting the latest values. These transfer mechanisms provide an efficient way to send real-time updates without constant polling from the client side, saving power for use cases such as remote sensors or any interrupt-driven activities.

Notifications and indications both transmit data from the server to the client, but notifications are unacknowledged, while indications are acknowledged. You can transmit multiple notifications during a single connection interval, but you can only transmit one indication every two connection intervals (one interval for the transmission and one for the acknowledgement).

Although the server decides when to push data to the client using these methods, the client retains ultimate control over whether the server may transmit at all, via the use of "subscription" bits for each type of transfer. All GATT characteristics which support either the "notify" or "indicate" operation must have a CCCD within the set of attributes making up the complete characteristic structure. For example, the "Service Changed" characteristic (UUID 0x2A05) within the "Generic Attribute" service (UUID 0x1801) is made up of three separate attributes as listed in 0.

### : Service changed GATT characteristic structure

Handle	UUID	Description
0x0009	0x2803	Characteristic declaration
0x000A	0x2A05	Service change value attribute
0x000B	0x2902	Client Characteristic Configuration Descriptor (CCCD)

This characteristic supports the "indicate" operation. For a client to subscribe to indications, it must set Bit 1 (0x02) of the value in the CCCD. This descriptor holds a 16-bit value, so the correct operation on the client side is to write [ 02 00 ] to the 0x000B handle.

For characteristics that support the "notify" operation, the correct subscription flag is Bit 0 (0x01).

Notification and indication subscriptions do not persist across multiple connections.

#### 3.5.4.1 Notifying data to a remote client

Use the `gatts_notify_handle (/NH, ID=5/11)` API command to notify data to a remote Client. You must use a handle corresponding to a value attribute for a characteristic for which the remote client has already subscribed to notifications by writing 0x0001 to the relevant CCCD. First, you need create a CCCD value as shown here.

**Note:** Notifying data to a client requires an active connection.

### : Example 9 - Notify a four-byte value to a Client manually using the customized characteristic with CCCD

Direction	Text Content	Binary content	Effect
TX→	/CAC, T=00, P=2, L=0012, D=002800 B10C2000089A9EE21115A1333336 5	C0 18 05 01 00 02 12 00 12 00 00 28 00 B1 0C 20 00 08 9A 9E E2 11 15 A1 33 33 33 65 89	Create a new CCCD value as follows:  First, create new service, UUID=.
←RX	@R, 0018, /CAC, 0000, H=0016, V=00 01	C0 06 05 01 00 00 16 00 01 00 7C	Response indicates success.
TX→	/CAC, T=00, P=2, L=0015, D=032828 180001B10C2000089A9EE21115A13 3333365	[C0 1B 05 01 00 02 15 00 15 00 03 28 28 18 00 01 B1 0C 20 00 08 9A 9E E2 11 15 A1 33 33 33 65 D6	Then, create a characteristic.
←RX	@R, 0018, /CAC, 0000, H=0017, V=00 01	C0 06 05 01 00 00 17 00 01 00 7D	Response indicates success.
TX→	/CAC, T=01, P=B9, L=0014, D=	C0 06 05 01 01 89 14 00 00 00 03	Create a value for the above characteristic.
←RX	@R, 0018, /CAC, 0000, H=0018, V=00 00	C0 06 05 01 00 00 18 00 00 00 7D	Response indicates success.
TX→	/CAC, T=00, P=0A, L=04, D=0229	C0 08 05 01 00 0A 04 00 02 00 02 29 A2	Create CCCD.
←RX	@R, 0018, /CAC, 0000, H=0019, V=00 00	C0 06 05 01 00 00 19 00 00 00 7E	
←RX	@E, 0035, C=40, A=00A05012C722, T =00, I=0007, L=0000, O=000A, B=0	80 0F 04 05 40 22 C7 12 50 A0 00 00 07 00 00 00 0A 00 00 6D	Connected from peer device
←RX	@E, 001A, W, C=40, H=0019, T=00, D= 0100	80 08 05 02 40 19 00 00 02 00 01 00 84	Subscribe service by peer device.
TX→	/NH, C=40, H=18, D=41424344	C0 08 05 0B 40 18 00 04 41 42 43 44 D7	Notify "ABCD" (hex) via attribute with handle = 17 (0x11).
←RX	@R, 0009, /NH, 0000	C0 02 05 0B 00 00 6B	Response indicates success.

#### 3.5.4.2 Indicating data to a remote client

Use the `gatts_indicate_handle (/IH, ID=5/12)` API command to indicate data to a remote client. You must use a handle corresponding to a value attribute for a characteristic for which the remote client has already subscribed to indications by writing 0x0002 to the relevant CCCD.

**Note:** Indicating data to a client requires an active connection.



: Example 10 - Indicate a start/end handle range to a client through the service changed characteristic

Direction	Text content	Binary content	Effect
←RX	@E, 001A, W, C=40, H=000B, T=00, D=0200	80 08 05 02 40 0B 00 00 02 00 02 00 77	Remote Client writes 0x002 to handle 0x0B to subscribe the Service Changed Characteristic.
TX→	/IH, C=40, H=A, D=1D002500	C0 08 05 0C 40 0A 00 04 1D 00 25 00 02	Write 1D002500 via attribute with handle = 10 (0x0A)
←RX	@R, 0009, /IH, 0000	C0 02 05 0C 00 00 6C	Response indicates success.
←RX	@E, 000F, IC, C=40, H=000A	80 03 05 03 40 0A 00 6E	Event indicates Client has confirmed receipt of data.

### 3.5.5 Detecting and processing written data from a remote client

Write operations from a remote GATT Client generates the `gatts_data_written` (W, ID=5/2) API event, containing the handle and value data as well as the remote connection handle from the device that initiated the request. This event occurs only if the write succeeds and was not blocked due to incorrect permissions, insufficient encryption or authentication levels, or invalid length or offset.

**Note:** EZ-Serial firmware platform for Ezurio Vela IF820 series module does not currently implement an API event for read requests.

## 3.6 GATT client examples

EZ-Serial firmware platform for Ezurio Vela IF820 series module provides GATT Client operational support through a variety of API methods. All methods described in the sections below require an active connection to a remote peer device and will generate an error result if attempted without an active connection.

### 3.6.1 How to discover a remote server's GATT structure

EZ-Serial firmware platform for Ezurio Vela IF820 series module's remote GATT discovery methods function in the same way as local discovery methods, with an addition of a connection handle in the discovery result output. For an overview of behavioral differences between local and remote GATT discovery, see [Listing local GATT services, characteristics, and descriptors](#).

**Note:** Any attribute that requires authentication (bonding) must also require encryption. If you enable the authentication bit, make sure that you also enable the encryption bit. If not, the command will be rejected with an error result.

**Note:** Remote discovery procedures often complete with a final result code of 0x060A rather than 0x0000. This does not indicate a problem, but only means that the final internal request to find more data in the specified start/end range yielded no further results. This is a logical indicator to the Client that it should terminate the discovery process. You can avoid this result code by specifying start and end range values in the discovery request command, which do not result in a final search in an empty range on the server. However, these start and end values are typically not available before performing the discovery in the first place.

#### 3.6.1.1 Discovering remote GATT services

Use the `gattc_discover_services` (/DRS, ID=6/1) API command to obtain a list of services in the remote GATT database on a connected peer device.

: Example 1 - Remote GATT service discovery on an EZ-Serial firmware platform for Ezurio Vela IF820 series module peer device with factory default configuration

Direction	Content	Effect
TX→	/DRS	Request to discover all remote services

Direction	Content	Effect
←RX	@R, 000A, /DRS, 0000	Response indicates success
←RX	@E, 0029, DR, C=04, H=0001, R=0007, T=2800, P=00, U=0018	Service 0x1800, start=1, end=7
←RX	@E, 0029, DR, C=04, H=0008, R=000B, T=2800, P=00, U=0118	Service 0x1801, start=8, end=11 (0x0B)
←RX	@E, 0045, DR, C=04, H=000C, R=0015, T=2800, P=00, U=00A10C2000089A9EE21115A133333365	Service 0x6533...A100, start=12 (0x0C), end=21 (0x15)

### 3.6.1.2 Discovering remote GATT characteristics

Use the [gattc\\_discover\\_characteristics \(/DRC, ID=6/2\)](#) API command to obtain a list of characteristics in the remote GATT database on a connected peer device.

*Example 1 - Remote GATT characteristic discovery on an EZ-Serial firmware platform for Ezurio Vela IF820 series module peer device with factory default configuration*

Direction	Content	Effect
TX→	/DRC	Request to discover all remote characteristics
←RX	@R, 000A, /DRC, 0000	Response indicates success
←RX	@E, 0029, DR, C=04, H=0002, R=0003, T=2803, P=02, U=002A	Char 0x2A00, decl handle=2, value handle=3, perm=0x02
←RX	@E, 0029, DR, C=04, H=0004, R=0005, T=2803, P=02, U=012A	Char 0x2A01, decl handle=4, value handle=5, perm=0x02
←RX	@E, 0029, DR, C=04, H=0006, R=0007, T=2803, P=02, U=042A	Char 0x2A04, decl handle=6, value handle=7, perm=0x02
←RX	@E, 0029, DR, C=04, H=0009, R=000A, T=2803, P=22, U=052A	Char 0x2A05, decl handle=9, value handle=10, perm=0x22
←RX	@E, 0045, DR, C=04, H=000D, R=000E, T=2803, P=28, U=01A10C2000089A9EE21115A133333365	Char 0x6533...A101, decl handle=13, value handle=14, perm=0x28
←RX	@E, 0045, DR, C=04, H=0010, R=0011, T=2803, P=14, U=02A10C2000089A9EE21115A133333365	Char 0x6533...A102, decl handle=16, value handle=17, perm=0x14
←RX	@E, 0045, DR, C=04, H=0013, R=0014, T=2803, P=20, U=03A10C2000089A9EE21115A133333365	Char 0x6533...A103, decl handle=19, value handle=20, perm=0x20
←RX	@E, 0045, DR, C=04, H=0017, R=0018, T=2803, P=28, U=01A20C2000089A9EE21115A133333365	Char 0x6533...A201, decl handle=23, value handle=24, perm=0x28
←RX	@E, 0045, DR, C=04, H=001A, R=001B, T=2803, P=28, U=02A20C2000089A9EE21115A133333365	Char 0x6533...A202, decl handle=26, value handle=27, perm=0x28
←RX	@E, 0010, RPC, C=04, R=060A	Remote procedure complete, 0x060A = no attributes found in last search request

### 3.6.1.3 Discovering remote GATT descriptors

Use the `gattc_discover_descriptors (/DRD, ID=6/3)` API command to obtain a list of descriptors in the remote GATT database on a connected peer device.

*Example 1 - Remote GATT descriptor discovery on an EZ-Serial firmware platform for Ezurio Vela IF820 series module peer device with factory default configuration*

Direction	Content	Effect
TX→	/DRD	Request to discover all remote descriptors
←RX	@R,000A,/DRD,0000	Response indicates success
←RX	@E,0024,DR,H=0001,R=0000,T=2800,P=00,U=0028	UUID 0x2800 (Primary Service), start=1
←RX	@E,0024,DR,H=0002,R=0000,T=2803,P=00,U=0328	UUID 0x2803 (Characteristic), decl=2
←RX	@E,0024,DR,H=0003,R=0000,T=0000,P=00,U=002A	UUID 0x2A00 (Device Name), handle=3
<i>Additional records omitted for brevity</i>		
←RX	@E,0029,DR,C=04,H=0016,R=0000,T=2800,P=00,U=0028	UUID 0x2800 (Primary Service), start=22
←RX	@E,0029,DR,C=04,H=0017,R=0000,T=2803,P=00,U=0328	UUID 0x2803 (Characteristic), decl=23
←RX	@E,0045,DR,C=04,H=0018,R=0000,T=0000,P=00,U=01A20C2000089A9EE21115A133333365	UUID 0x6533...A201 (CYCommand Challenge), handle=24
←RX	@E,0029,DR,C=04,H=0019,R=0000,T=2902,P=00,U=0229	UUID 0x2902 (CCCD), handle=25
←RX	@E,0029,DR,C=04,H=001A,R=0000,T=2803,P=00,U=0328	UUID 0x2803 (Characteristic), decl=26
←RX	@E,0045,DR,C=04,H=001B,R=0000,T=0000,P=00,U=02A20C2000089A9EE21115A133333365	UUID 0x6533...A202 (CYCommand Data), handle=27
←RX	@E,0029,DR,C=04,H=001C,R=0000,T=2902,P=00,U=0229	UUID 0x2902 (CCCD), handle=28
←RX	@E,0010,RPC,C=04,R=060A	Long remote procedure complete, 0x060A = no attributes found in last search request

### 3.6.2 How to read and write remote GATT attribute values

Reading and writing local GATT values can be done with the `gattc_read_handle (/RRH, ID=6/4)` and `gattc_write_handle (/WRH, ID=6/5)` API commands, respectively.

### 3.6.3 How to detect notified or indicated values from a remote GATT server

A remote GATT Server may push data updates to a GATT Client at unpredictable times if the client has subscribed to notifications or indications on a supported remote GATT Server characteristic. When this occurs, EZ-Serial firmware platform for Ezurio Vela IF820 series module generates the `gattc_data_received` (D, ID=6/3) API event with the connection handle, attribute handle, and value data.

To receive notifications or indications from a remote GATT server, you must first subscribe to the relevant type of data updates by writing a special value to the attribute called Client Characteristic Configuration Descriptor (CCCD). This attribute always has a UUID of 0x2902, and is a separate attribute relative to the characteristic declaration (UUID 0x2803) or characteristic value (custom UUID).

Usually, the CCCD attribute has a handle value that is +1 or +2 from the characteristic value attribute. You can use the `gattc_discover_descriptors` (/DRD, ID=6/3) API command to obtain a list of descriptors and identify which attributes you need to use. For example, a remote server structure might contain something like the following:

```
Handle 0x0017, UUID 0x2803: Characteristic Declaration Descriptor
Handle 0x0018, UUID 0x2A46: Characteristic Value Descriptor ("New Alert" characteristic)
Handle 0x0019, UUID 0x2902: Client Characteristic Configuration Descriptor
```

With this structure, you can subscribe to notifications for this characteristic by writing the 16-bit value 0x0001 to the attribute with handle 0x0019. Remember that you must write this value as a little-endian integer [ 01 00 ]. To unsubscribe from receiving notifications, simply write the value 0x0000 to the same CCCD attribute.

Subscribing to indications requires the same procedure, but you must use the value 0x0002 instead of 0x0001.

The CCCD attribute with UUID 0x2902 will only be present for a characteristic which supports either notifications or indications. Whether you should enable notifications or indications depends on which of those two GATT methods is implemented on the GATT Server side. For official, adopted characteristics, you can find this information on the Bluetooth® SIG developer website. For proprietary/custom characteristics, see the documentation or reference material available from the product developer.

## 3.7 Security and encryption examples

EZ-Serial firmware platform for Ezurio Vela IF820 series module supports built-in Bluetooth® security technologies for safeguarding sensitive data transmitted wirelessly, including privacy and encryption.

### 3.7.1 Bonding with or without MITM protection

Bonding between two devices requires generating and exchanging encryption keys, and then permanently storing encryption data along with the information required to identify the bonded device and reuse the same keys again in the future. The mechanism of pairing depends on which side (master or slave) initiates the pairing request, and the I/O capabilities of each side.

**Note:** While the Bluetooth® specification allows pairing (generation and exchange of encryption keys) without bonding (permanent storage of encryption data), most common smartphones, tablets, and computer operating systems require performing both at the same time if you need encryption. The encryption-only arrangement (no bonding) is supported only between modules that support pairing without bonding.

EZ-Serial firmware platform for Ezurio Vela IF820 series module supports pairing with or without MITM protection enabled. The factory default settings apply the so-called "just works" method, with no passkey entry and no MITM protection.

#### 3.7.1.1 Pairing in "Just Works" mode without MITM protection (BLE)

The simplest way to bond requires no special passkey entry or display. If your device has no input or output capabilities, you must use this mode for pairing since MITM protection requires numeric display or entry (or both) to function correctly.

0 assumes that you have already connected to a remote peer device. An active connection is required for any type of pairing operation to succeed. However, configuration of security settings may be done either before or after connecting.

*: Example 11 - Configure simple pairing without MITM protection, then initiate pairing*

Direction	Text content	Binary content	Effect
TX→	SSBP, M=40, B=1, K=10, P=0, I=3, F=1	C0 06 07 0B 40 01 10 00 03 01 C6	Set "No Input / No Output" I/O (Factory default).
←RX	@R, 000A, SSPB, 0000	C0 02 07 0B 00 00 6D	Response indicates success.

Direction	Text content	Binary content	Effect
TX→	/P,C=01,B=0,K=10,M=40,P=0	C0 05 07 03 01 40 00 10 00 B9	Initiate pairing request to remote peer.
←RX	@R,0008,/P,0000	C0 02 07 03 00 00 65	Response indicates success.
←RX	@E,001B,P,C=01,M=00,B=00,K=00,P=00	80 05 07 02 01 00 00 00 28	Event indicates pairing process request.
←RX	@E,000F,PR,C=01,R=0000	80 03 07 03 01 16 00 3D	Event indicates pairing process completed successfully.
←RX	@E,000E,ENC,C=01,S=00	80 02 07 04 01 00 27	Event indicates encryption status changed successfully.

### 3.7.1.2 Pairing with a fixed passkey(BLE) (Obsolete, not supported)

EZ-Serial firmware platform for Ezurio Vela IF820 series module supports the configuration of a fixed passkey to be used during the pairing process instead of either no passkey or a random one. You can choose a fixed 6-digit value between 000000 and 999999 by using the [smp\\_set\\_fixed\\_passkey \(SFPK, ID=7/13\)](#) API command and configuring the local I/O capabilities to the "Display Only" value with the [smp\\_set\\_security\\_parameters \(SSBP, ID=7/11\)](#) API command.

**Note:** The fixed passkey takes effect only if you enable fixed passkey use by setting Bit 1 (0x02) of the security flags parameter and set the "Display Only" I/O capabilities value (0x00) using the [smp\\_set\\_security\\_parameters \(SSBP, ID=7/11\)](#) API command. If both conditions are not met, the stack reverts to the default behavior of using a random passkey.

0 assumes that the module is already connected to a remote peer device. An active connection is required for any type of pairing operation to succeed. However, configuration of security settings may be done either before or after connection.

In CYSmart, you need to set its IO ability to keyboard only.

#### : Example 12 - Configure "123456" fixed passkey value and required I/O capabilities, then pair from remote peer

Direction	Text content	Binary content	Effect
TX→	SSBP,M=4D,B=1,K=10,P=0,I=0,F=3	C0 06 07 0B 4D 01 10 00 02 03 D4	Set "Display Only" I/O, enable fixed passkey use flag bit (0x02).
←RX	@R,000A,SSPB,0000	C0 02 07 0B 00 00 6D	Response indicates success.
TX→	SFPK,P=1E240	C0 04 07 0D 40 E2 01 00 94	Set fixed passkey value (1E240 hex = <b>123456</b> dec).
←RX	@R,000A,SFPK,0000	C0 02 07 0D 00 00 6F	Response indicates success.
←RX	@E,001B,P,C=01,M=00,B=00,K=00,P=00	80 05 07 02 01 00 00 00 28	Event indicates pairing process request.
←RX	@E,000F,PR,C=01,R=0000	80 03 07 03 01 00 00 27	Event indicates encryption status changed (peer entered key).
←RX	@E,000E,ENC,C=01,S=00	80 02 07 04 01 00 27	Event indicates encryption status changed successfully.

### 3.7.1.3 Pairing with a random passkey (BLE)

0 shows how to generate a random passkey and that peer device compares the passkey and accept pairing.

In CYSmart, you need to set its IO ability to display Yes or No.

: Example 13 - Configure random passkey value and required I/O capabilities, then pair from remote peer

Direction	Text content	Binary content	Effect
TX→	SSBP,M=4D,B=1,K=10,P=0,I=4,F=3	C0 06 07 0B 4D 01 10 00 04 03 D6	Set "Keyboard + Display" I/O, enable fixed passkey use flag bit (0x02).
←RX	@R,000A,SSPB,0000	C0 02 07 0B 00 00 6D	Response indicates success.
←RX	@E,001B,P,C=01,M=00,B=00,K=00,P=00	80 05 07 02 01 00 00 00 00 28	Event indicates pairing process request.
←RX	@E,0014,PKD,C=01,P=0000EA26	80 05 07 05 01 26 EA 00 00 3B	Event shows the random passkey
			Peer device compare passkey and click yes
←RX	@E,000F,PR,C=01,R=0000	80 03 07 03 01 00 00 27	Event indicates encryption status changed (peer entered key).
←RX	@E,000E,ENC,C=01,S=00	80 02 07 04 01 00 27	Event indicates encryption status changed successfully.

### 3.7.1.4 Pairing with a random passkey (BT classic)

0 illustrates how to enter a Passkey to accept pairing with Bluetooth® Classic.

: Example 14 - Enter random key to accept the pair from remote peer

Direction	Text content	Binary content	Effect
TX→	SSBP,M=4D,B=1,K=10,P=0,I=2,F=3	C0 06 07 0B 4D 01 10 00 02 03 D4	Set "Display Only" I/O, enable fixed passkey use flag bit (0x02).
←RX	@R,000A,SSPB,0000	C0 02 07 0B 00 00 6D	Response indicates success.
←RX	@E,001B,P,C=00,M=00,B=00,K=00,P=00	80 05 07 02 00 00 00 00 00 27	Event indicates pairing process request.
←RX	@E,0015,BTPIN,A=E4A471C2FDFC	80 06 07 07 FC FD C2 71 A4 E4 E1	Pin entry request from peer device
TX→	/BTPIN,P=C8CEC,C=0	C0 05 07 11 00 EC 8C 0C 00 FA	Send BT PIN code to peer device as it displays
←RX	@R,000C,/BTPIN,0000	C0 02 07 11 00 00 73	Response indicates success.
←RX	@E,000F,PR,C=00,R=0000	80 03 07 03 00 00 00 26	Event indicates encryption status changed (peer entered key).

Direction	Text content	Binary content	Effect
←RX	@E, 000E, ENC, C=00, S=00	80 02 07 04 00 00 26	Event indicates encryption status changed successfully.

0 shows how to compare the passkey and uses a yes/no indication for pairing via an end product display. The peer device compares the key and accepts pairing.

: Example 15 - Display random passkey value for Peer Device and select Yes/No to accept pairing

Direction	Text content	Binary content	Effect
TX→	SSBP, M=5D, B=1, K=10, P=0, I=0, F=3	C0 06 07 0B 4D 01 10 00 02 03 D4	Set "Display Only" I/O, enable fixed passkey use flag bit (0x02).
←RX	@R, 000A, SSPB, 0000	C0 02 07 0B 00 00 6D	Response indicates success.
←RX	@E, 001B, P, C=00, M=00, B=00, K=00, P=00	80 05 07 02 00 00 00 00 00 27	Event indicates pairing process request.
←RX	@E, 0014, PKD, C=01, P=0000EA26	80 05 07 05 01 26 EA 00 00 3B	Event shows the random passkey
			Peer device compare passkey and click yes
←RX	@E, 000F, PR, C=00, R=0000	80 03 07 03 00 00 00 26	Event indicates encryption status changed (peer entered key).
←RX	@E, 000E, ENC, C=00, S=00	80 02 07 04 00 00 26	Event indicates encryption status changed successfully.

## 3.8 Performance testing examples

This section covers techniques to achieve optimal performance in specific contexts.

### 3.8.1 Maximizing throughput to a remote peer

Throughput concerns how much data you can move across a link within a specific period, usually expressed in bytes per second or bits per second (8 bits per byte). In the case of BLE, the following guidelines help improve the average throughput:

**Minimize the connection interval.** The BLE specification allows 7.5 ms minimum connection interval. Data transfers are specifically timed during BLE connections, and more frequent transfers mean higher potential throughput.

When operating in the **GAP Peripheral** role, the remote Central determines the initial interval, and you must request an update with the [gap\\_update\\_conn\\_parameters \(UCP, ID=4/3\)](#) API command after connecting. The remote peer (master/central device) may either accept or reject this request. Note that if the remote peer rejects the request, it does not notify the requesting device; the only evidence of the rejection is the lack of a subsequent [gap\\_connection\\_updated \(CU, ID=4/8\)](#) API event.

**Maximize the payload size for GATT transfers.** It takes much longer to send 20 one-byte packets than one 20-byte packet, due to the low transmission duty cycle required by the BLE protocol. If your application has five 16-bit sensor measurement values that are used to the remote peer on the same interval, use a single characteristic to send all 10 bytes at once rather than using five separate characteristics.

**Use unacknowledged transfers.** You can push more unacknowledged data through in a single connection interval than you can with acknowledged transfers. A typical acknowledged data transfer requires two full connection intervals to complete (one for the transfer and one for the acknowledgement), but multiple unacknowledged transfers can be used in sequence within the same interval—up to one packet every 1.25 ms, if supported by the remote client. Typically, standalone full-stack modules cannot buffer and process data quite this fast, but it is often possible to achieve something near this level of throughput. Note that making this change may require additional application logic to provide a packet delivery/retry request mechanism.

For **client-to-server** transfers, use the "write-no-response" operation instead of "write."

For **Server-to-Client** transfers, use the "notify" operation instead of "indicate."

These actions help increase the observed throughput, but simultaneously increase power consumption. Keep this trade-off in mind to choose the right balance between power consumption and throughput.

**Example 16 - Request a connection parameter update to 7.5-ms interval, no latency, 1-second timeout**

Direction	Text content	Binary content	Effect
TX→	/UCP,C=40,I=6,L=0,O=64	C0 07 04 03 40 06 00 00 00 64 00 11	Request connection update to 7.5 ms (6 * 1.25 ms), no slave latency, 1-second supervision timeout.
←RX	@R,000A,/UCP,0000	C0 02 04 03 00 00 62	Response indicates success; request sent to remote peer.
←RX	@E,001D,CU,C=40,I=0006,L=0000,O=0064	80 07 04 08 40 06 00 00 00 64 00 D6	Event indicates new connection parameters accepted.

### 3.8.1.1 Maximizing throughput to an iOS device

Apple devices began supporting BLE technology with the iPhone 4S and iOS 5. iOS devices have additional limitations on top of those mandated in the Bluetooth® specification.

The following additional guidelines apply for maximizing iOS throughput:

- When operating in the GAP Central role, the latest iOS devices limit the minimum connection interval of 30 ms (or 11.25 ms when connecting to HID devices). If the peripheral requests a shorter connection interval than this, the iOS device rejects the request.
- iOS devices limit unacknowledged GATT data transfers (write-no-response or notify) to a maximum of four per connection interval, according to widespread observations.
- iOS 5 added support for GAP Peripheral role operation, which includes support for 7.5-ms intervals as required by the Bluetooth® specification. However, switching GAP roles may not be suitable depending on other application requirements, and requires a notably different mobile app development approach with its own side effects. See the [Core Bluetooth® Programming Guide](#) on the Apple Developer website for official guidelines.

**Example 17 - Request a connection parameter update to 30-ms interval, no latency, 1-second timeout**

Direction	Text content	Binary content	Effect
TX→	/UCP,C=40,I=18,L=0,O=64	C0 07 04 03 40 18 00 00 00 64 00 23	Request connection update to 30 ms (24 * 1.25 ms), no slave latency, 1-second supervision timeout.
←RX	@R,000A,/UCP,0000	C0 02 04 03 00 00 62	Response indicates success; request sent to remote peer.
←RX	@E,001D,CU,C=40,I=0018,L=0000,O=0064	80 07 04 08 40 18 00 00 00 64 00 E8	Event indicates new connection parameters accepted.

### 3.8.1.2 Maximizing throughput to an Android device

Android devices officially began supporting BLE technology with the Android 4.3 release, though Android 4.4 and onward greatly improved stability and supported functionality.

The following additional guidelines apply for maximizing Android throughput:

- Android 4.4.2 and earlier releases only support a single connection interval of 48.75 ms.
- Android 4.4.3 and later releases support intervals down to 7.5 ms when requested by the remote device, even though the default interval is still 48.75 ms when first establishing the connection.
- Newer Android handsets allow up to six unacknowledged GATT transfers in a single connection interval.



### 3.8.1.3 Minimizing power consumption

You can reduce power consumption by making the BLE radio active as infrequently as your application allows. Specific actions described in this section help decrease average consumption, but also decreases the potential throughput. Keep this trade-off in mind to choose the right balance between power consumption and throughput.

If you have not already done so, ensure that the best possible CPU sleep mode for your application is configured as described in section [Managing sleep states](#). This will ensure that the CPU is not taking more power than necessary. If the CPU is fully or partially awake more often than necessary, relative improvements possible using the methods described below may not make a notable difference.

### 3.8.1.4 Minimizing power consumption while broadcasting

To reduce power consumption in an advertising state:

**Maximize the advertisement interval while broadcasting.** The BLE specification allows advertising at any interval between 20 ms and 10240 ms. Increasing the interval means fewer transmissions within a given period. For example, a device advertising at 500 ms will use roughly 20% of the power required by that same device advertising at 100 ms. Use the [gap\\_set\\_adv\\_parameters](#) (SAP, ID=4/23) API command to change the default advertisement interval, or the [gap\\_start\\_adv](#) (IA, ID=4/8) API command to use a non-default interval at the moment you enter an advertising state.

Side effects:

Scanning devices are less likely to detect each advertisement packet, due to the reduced probability of the scanning device actively receiving on the same channel at the same time as the advertisement transmission occurs.

Connections may take longer to establish, because this process begins with the same scanning process and requires detection of a connectable advertisement packet from the target device.

**Do not use all three advertisement channels.** The BLE spectrum dedicates three channels to advertisement packets, spread across the 2.4-GHz Bluetooth® RF spectrum to help ensure reception in busy RF environments. Most BLE devices advertise on all three channels, but you can selectively advertise on only one or two of these channels using the [gap\\_set\\_adv\\_parameters](#) (SAP, ID=4/23) or [gap\\_start\\_adv](#) (IA, ID=4/8) API commands. Advertising on only one channel requires roughly 33% of the power needed when using all three.

Side effects:

Scanning devices are less likely to detect advertisement packets for the same reason as above—there are fewer advertisement packets being transmitted, which reduces the probability of actively receiving on the correct channel at the correct time.

The advertising device cannot combat RF interference as effectively. If you enable only one advertisement channel, but that portion of the RF spectrum is extremely congested, then a scanning device may not be able to detect advertisement packets at all even if the timing lines up correctly.

**If connections are not required, use a non-connectable/non-scannable mode.** When a Peripheral device is connectable (accepting new connections) or scannable (accepting scan request packets while advertising), the BLE radio switches to a receiving state for approximately 150 µs after every advertisement packet to listen for a connection request or scan request packet. When using all three advertising channels, this means three complete TX-RX cycles occur repeatedly at the configured advertisement interval. If a Peripheral device needs to broadcast only, you can configure a broadcast-only advertising mode with the [gap\\_set\\_adv\\_parameters](#) (SAP, ID=4/23) or [gap\\_start\\_adv](#) (IA, ID=4/8) API commands. This prevents the radio from switching into a receiving state after each transmission, saving both time and power.

Side effects:

Any data configured in the scan response packet payload is never transmitted. Most often, this is the friendly device name.

**Minimize the advertisement and/or scan response data payload length.** Regardless of the configured advertisement interval, the advertisement payload also has a significant effect on the amount of time spent on transmissions. The advertisement payload may be between 0 and 31 bytes, and the BLE RF protocol uses a symbol rate of 1 Mbit/sec, which translates to 8 µs per byte. The fixed encapsulation and overhead data in every advertisement or scan response packet takes roughly 140 µs to transmit, but the payload can add up to 248 µs to this duration. In other words, a 31-byte payload (~390 µsec) requires twice as much transmission time as a 7-byte payload (~195 µs).

In most cases, the application design requires very specific content in the advertisement payload. However, you should optimize this as much as possible if low power consumption is critical for the application. You can configure custom advertisement data content with the [gap\\_set\\_adv\\_data](#) (SAD, ID=4/19) and [gap\\_set\\_adv\\_parameters](#) (SAP, ID=4/23) API commands, as described in section [Customizing advertisement and scanning response data](#).

### 3.8.1.5 Minimizing power consumption while connected

To reduce power consumption in a connected state:

**Maximize the connection interval.** The BLE specification allows a connection interval from 7.5 ms to 4000 ms.

When operating in the **GAP Peripheral** role, the remote Central determines the initial interval; you must request an update after connecting if you need to change it. The remote peer may either accept or reject this request.

**Use non-zero slave latency.** While this affects only power consumption on the slave or peripheral device during a connection, the slave latency setting can drastically improve power efficiency in many applications. This setting controls how many connection intervals the slave may

skip if it has no data to send to the connected master device. Once the allowed number of intervals have occurred, the slave must respond regardless of whether it has any new data to send. The slave may respond at any interval.

With the default "0" slave latency setting, the slave must acknowledge the master's connection maintenance packets at every interval. In applications requiring infrequent data transfers, this wastes a great deal of power. Increasing the slave latency value to "3" allows the slave to respond every four intervals instead of every interval, for an average power reduction of 75% while connected. Applications such as environmental sensors and human input devices can benefit greatly from non-zero slave latency.

The slave latency value may not be higher than the maximum number that allows the calculated value for `[conn_interval * slave_latency]` to remain below the `supervision_timeout` value, because otherwise the connection would time out regularly. Side effects:

If the slave has no data to send, the master must wait until the slave latency period passes before it can send or request data to or from the slave. The slave will not be aware of any requests from the master until it enables its radio again. This can result in noticeable delays especially when using long connection intervals. For example, a 500-ms connection interval and slave latency setting of "3" could create a master-to-slave response delay of up to two full seconds. To mitigate this, select a balanced combination of connection interval and slave latency values that provides acceptable master-side delay and slave-side power consumption.

Non-zero slave latency interval increases the possibility of a connection timeout in non-optimal RF environments. The master triggers a supervision timeout condition if it does not receive an acknowledgement from the slave before the timeout period elapses. The master resends any connection maintenance packet that is not acknowledged, but if the slave has already switched back to a low-power state between required response intervals, the master's attempted retries may be ignored for too long. To mitigate this, select a longer supervision timeout, shorter connection interval, and/or lower slave latency value to achieve required connection stability in the target environment.

**Use unacknowledged transfers.** Acknowledged transfers involve more data sent over the air to handle the acknowledgement. This results in higher average consumption. If you do not need application-level data transfer confirmations, use unacknowledged methods instead.

For **client-to-server** transfers, use the "write-no-response" operation instead of "write."

For **Server-to-Client** transfers, use the "notify" operation instead of "indicate."

## 3.9 Device firmware update examples

See section [Latest EZ-Serial firmware platform for Ezurio Vela IF820 series module image](#) for information on where to find the latest EZ-Serial firmware platform for Ezurio Vela IF820 series module firmware images.

### 3.9.1 Updating firmware locally using UART

If you have access to the HCI UART interface, you can use standard the [ModusToolbox™](#) Software and an UART interface to flash a new firmware image onto the module. Details about how to do this are available on the [Infineon website](#).

Updating firmware via this method always returns to factory default settings and removes any bonding data and custom GATT structure.

## 3.10 GPIO operation examples

EZ-Serial firmware platform for Ezurio Vela IF820 series module supports reading and configuring GPIO states including during system start up, before entering, or after exiting a low-power state. It also supports reading and configuring GPIO interrupts.

### 3.10.1 Get current GPIO status

EZ-Serial firmware platform for Ezurio Vela IF820 series module supports reading the current status and configuration of GPIO for example input, output. It also supports reading current status and configuration of GPIO interrupt:

Direction	Content	Effect
Get current GPIO status		
TX→	GIOL, P=2, D=0	Get the <b>input</b> status and configuration of Pin 2
←RX	@R, 0020, GIOL, 0000, L=00000001, H=00004000	Input status is <b>HIGH</b> and configuration is 0x4000 (out_enable) for Pin 2
TX→	GIOL, P=2, D=1	Get the <b>output</b> status and configuration Pin 2

Direction	Content	Effect
←RX	@R, 0020, GIOL, 0000, L=00000001, H=00004000	Output status is <b>HIGH</b> and configuration is <b>0x4000</b> for Pin 2
TX→	GIOL, P=2, D=2	Get the <b>interrupt</b> status and configuration of Pin 2
←RX	@R, 0020, GIOL, 0000, L=00000001, H=00004000	Interrupt status is <b>set</b> and configuration is <b>0x4000</b> for Pin 2.

**Note:** The interrupt status works only when interrupt is configured; L=1 here does not have any effect.

### 3.10.2 GPIO configuration when entering or exiting Low-Power state

To support the low-power scenario, the system may need to change GPIO state to LOW or HIGH when it enters or exits a low-power state (sleep level =1).

Direction	Content	Effect
Set GPIO behavior when system enters/exits low power		
TX→	SIOD, P=2, C=4200, L=0, O=0	To set Pin 2 to <b>low</b> with GPIO configuration set to <b>Pull down immediately</b>
←RX	@R, 000A, SIOD, 0000	Response indicates success. LED should be ON.
TX→	SIOD, P=2, C=4400, L=1, O=1	To set Pin 2 to <b>high</b> with configuration set to <b>Pull up</b> when system <b>enters low power</b> state
←RX	@R, 000A, SIOD, 0000	Response indicates success
TX→	SIOD, P=2, C=4200, L=0, O=2	To set Pin 2 to low with configuration set to <b>Pull down</b> when system <b>exits low power</b> state
←RX	@R, 000A, SIOD, 0000	Response indicates success

With above configuration, LED will turn ON when device enters Low-Power state and turn OFF when device exits Low-Power state (setting LP\_MOD pin to high or low).

### 3.10.3 GPIO pin configuration

GPIO pin configuration is a 32-bit value which corresponds to the internal WICED SDK API definition. The following are the details of configuration from WICED SDK API:

```
// Trigger Type
// GPIO configuration bit 0, Interrupt type defines
GPIO_EDGE_TRIGGER_MASK    = 0x0001,
GPIO_EDGE_TRIGGER         = 0x0001,
GPIO_LEVEL_TRIGGER         = 0x0000,

// Negative Edge Triggering
// GPIO configuration bit 1, Interrupt polarity defines
GPIO_TRIGGER_POLARITY_MASK = 0x0002,
GPIO_TRIGGER_NEG           = 0x0002,
```

```
// Dual Edge Triggering
// GPIO configuration bit 2, single/dual edge defines
GPIO_DUAL_EDGE_TRIGGER_MASK = 0x0004,
GPIO_EDGE_TRIGGER_BOTH      = 0x0004,
GPIO_EDGE_TRIGGER_SINGLE    = 0x0000,

// Interrupt Enable
// GPIO configuration bit 3, interrupt enable/disable defines
GPIO_INTERRUPT_ENABLE_MASK  = 0x0008,
GPIO_INTERRUPT_ENABLE       = 0x0008,
GPIO_INTERRUPT_DISABLE      = 0x0000,

// Interrupt Config
// GPIO configuration bit 0:3, Summary of Interrupt enabling type
GPIO_EN_INT_MASK            = GPIO_EDGE_TRIGGER_MASK | GPIO_TRIGGER_POLARITY_MASK |
GPIO_DUAL_EDGE_TRIGGER_MASK | GPIO_INTERRUPT_ENABLE_MASK,
GPIO_EN_INT_LEVEL_HIGH      = GPIO_INTERRUPT_ENABLE | GPIO_LEVEL_TRIGGER,
GPIO_EN_INT_LEVEL_LOW       = GPIO_INTERRUPT_ENABLE | GPIO_LEVEL_TRIGGER |
GPIO_TRIGGER_NEG,
GPIO_EN_INT_RISING_EDGE      = GPIO_INTERRUPT_ENABLE | GPIO_EDGE_TRIGGER,
GPIO_EN_INT_FALLING_EDGE     = GPIO_INTERRUPT_ENABLE | GPIO_EDGE_TRIGGER |
GPIO_TRIGGER_NEG,
GPIO_EN_INT_BOTH_EDGE        = GPIO_INTERRUPT_ENABLE | GPIO_EDGE_TRIGGER |
GPIO_EDGE_TRIGGER_BOTH,

// GPIO Output Buffer Control and Output Value Multiplexing Control
// GPIO configuration bit 4:5, and 14 output enable control and
// muxing control
GPIO_INPUT_ENABLE           = 0x0000,
GPIO_OUTPUT_DISABLE         = 0x0000,
GPIO_OUTPUT_ENABLE          = 0x4000,
GPIO_KS_OUTPUT_ENABLE       = 0x0010, // Keyscan Output enable
GPIO_OUTPUT_FN_SEL_MASK     = 0x0030,
GPIO_OUTPUT_FN_SEL_SHIFT    = 4,

// Global Input Disable
// GPIO configuration bit 6, "Global_input_disable" Disable bit
// This bit when set to "1" , P0 input_disable signal will control
// ALL GPIOs. Default value (after power up or a reset event) is "0".
GPIO_GLOBAL_INPUT_ENABLE     = 0x0000,
GPIO_GLOBAL_INPUT_DISABLE    = 0x0040,

// Pull-up/Pull-down
// GPIO configuration bit 9 and bit 10, pull-up and pull-down enable
// Default value is [0,0]--means no pull resistor.
GPIO_PULL_UP_DOWN_NONE      = 0x0000,    //[0,0]
GPIO_PULL_UP                 = 0x0400,    //[1,0]
GPIO_PULL_DOWN               = 0x0200,    //[0,1]
GPIO_INPUT_DISABLE           = 0x0600,    //[1,1] // input disables the GPIO

// Drive Strength
// GPIO configuration bit 11
GPIO_DRIVE_SEL_MASK          = 0x0800,
GPIO_DRIVE_SEL_LOWEST        = 0x0000,    // 2mA @ 1.8V
GPIO_DRIVE_SEL_MIDDLE_0      = 0x0000,    // 4mA @ 3.3v
GPIO_DRIVE_SEL_MIDDLE_1      = 0x0800,    // 4mA @ 1.8v
GPIO_DRIVE_SEL_HIGHEST       = 0x0800,    // 8mA @ 3.3v

// Input Hysteresis
// GPIO configuration bit 13, hysteresis control
GPIO_HYSTERESIS_MASK         = 0x2000,
GPIO_HYSTERESIS_ON           = 0x2000,
GPIO_HYSTERESIS_OFF          = 0x0000,
```

### 3.11 Init command examples

The init commands feature allows you to store EZ-Serial firmware platform for Ezurio Vela IF820 series module commands in the pre-allocated section. During FW startup, FW loads these init commands from pre-allocated section and executes them in a sequence. It is useful for the use cases which do not have host MCU. Currently, the stored command is only for text format command. It is the extend feature for the command "/WUD" and "/RUD". Refer [system\\_write\\_user\\_data \(/WUD, ID=2/11\)](#) and [system\\_read\\_user\\_data \(/RUD, ID=2/12\)](#).

FW reserves some flash slots (3 slots in current FW implementation) as a list to save Init commands. Each slot is 255 bytes. User can not add additional Init commands after reaching maximum capacity (Maximum capacity for existing FW is 765 bytes (3\*255=765 Bytes)).

EZ-Serial firmware platform for Ezurio Vela IF820 series module provides command to Add, Delete and Display Init commands.

#### 3.11.1 Add Init command

Current EZ-Serial firmware platform for Ezurio Vela IF820 series module provides two methods to add commands into Init command list: use history command information or simply use prefix '&'.

Direction	Content	Effect
Add command to Init command list with two methods		
TX→	/PING	Sent to ping the local module to verify proper communication
←RX	@R,001D,/PING,0000,R=00000115,F=3DBF	Response indicates success
TX→	/WUD,O=1,D=00,M=3	Use history command info and store it to flash. O = 1 means the <b>previous</b> command. 'D' is ignored.
←RX	@R,000A,/WUD,0000	Response indicates success
TX→	&GBA	Store "GBA" to Init command list using '&' before the command
←RX	@R,0018,GBA,0000,A=E755F205D0D8	Response indicates success

#### 3.11.2 Display current Init commands

Direction	Content	Effect
Display current commands in the Init command list		
TX→	/RUD,O=00,M=5,L=1	<b>Display</b> all Init commands in the Init command list. 'O' and 'L' is ignored in this command.
←RX	Init cmd list(Enabled): 00:[/ping] 01:[GBA] end of list @R,000D,/RUD,0000,D=	Response indicates success and shows "/PING" and "GBA" is stored. 'D' is ignored here.

#### 3.11.3 Check Init command is executed at system start up

Direction	Content	Effect
Command in the Init command list will be executed after system start up		

Direction	Content	Effect
TX→	/RBT	Reset the module ( or you can press reset button to reset module)
←RX	@R,000A,/RBT,0000 @E,003B,BOOT,E=01021313,S=05020016,P=0103,H=D1,C=00,A=E755F205D0D8 @E,000E,ASC,S=01,R=03 Start executing init cmd: --->/ping @R,001D,/PING,0000,R=00000000,F=023E ---> <a href="#">GBA</a> @R,0018,GBA,0000,A=E755F205D0D8 Finish executing init cmd!	Commands are executed sequentially after system start up. Observe that GBA is command is executed in the startup sequence.

### 3.11.4 Delete Init command

Direction	Content	Effect
Delete command one by one or remove all commands		
TX→	/WUD,O= <a href="#">1</a> ,D=00,M= <a href="#">4</a>	<a href="#">Delete</a> Init command <a href="#">1</a> from the list 'D' is ignored
←RX	@R,000A,/WUD,0000	Response indicates success
TX→	/RUD,O=00,M= <a href="#">5</a> ,L=1	<a href="#">display</a> all Init commands in the Init command list
←RX	Init cmd list (Disabled): 00:[/PING] end of list @R,000D,/RUD,0000,D=	Command <a href="#">1</a> (GBA) is removed
TX→	/WUD,O= <a href="#">1</a> ,D=00,M= <a href="#">5</a>	<a href="#">Remove all commands</a> from the Init command list 'D' is ignored
←RX	@R,000A,/WUD,0000	Response indicates success
TX→	/RUD,O=00,M= <a href="#">5</a> ,L=1	<a href="#">Display</a> current Init commands in the Init command list
←RX	@R,000D,/RUD,0000,D=	No commands in the list now

### 3.11.5 Enable/disable Init command

Init command execution is enabled by default in EZ-Serial firmware platform for Ezurio Vela IF820 series module. EZ-Serial firmware platform for Ezurio Vela IF820 series module provides commands to disable/enable Init command execution based on users requirement.

Direction	Content	Effect
Disable/Enable Init command operation		

Direction	Content	Effect
TX→	/WUD,O=FFFF,M=5,D=1	<b>Disable</b> Init command. D= 1 is disable. D=0 is enable
←RX	@R,000A,/WUD,0000	Response indicates success
TX→	/RUD,O=00,M=5,L=1	<b>Display</b> all Init commands in the Init command list. 'O' and 'L' is ignored.
←RX	Init cmd list( <b>Disabled</b> ): 00:[/PING] 01:[GBA] end of list @R,000D,/RUD,0000,D=	Response indicates success and Init command list is Disabled. If you restart system, these commands will not be executed.
TX→	/WUD,O=FFFF,M=5,D=0	<b>Enable</b> Init command. D= 1 is disable. D=0 is enable
←RX	@R,000A,/WUD,0000	Response indicates success

## 4 Application design examples

Examples in this section describe the hardware design and platform configuration necessary for some common types of applications. You can use any of these exactly as described for your design or modify as needed.

### 4.1 Smart MCU host with 4-Wire UART and full GPIO connections

This application design example allows maximum functionality with an external host microcontroller, including efficient sleep state control and optional CYSPP/SPP communication.

#### 4.1.1 Hardware design

Include the following design elements in your hardware:

- Module UART\_TX pin to host UART RX pin
- Module UART\_RX pin to host UART TX pin
- Module UART\_CTS pin to host UART RTS pin
- Module UART\_RTS pin to host UART CTS pin
- Module CYSPP, and LP\_MODE pins to digital output host GPIOs
- Module CONNECTION pins to high-impedance digital input host GPIO

#### 4.1.2 Module configuration

Most configuration settings will depend on your communication requirements. However, you may wish to make one or more of the following changes:

- Change Device Name with `gap_set_device_name` (SDN, ID=4/15)
- Change CYSPP connection key and/or security requirements with `p_cyspp_set_parameters` (.CYSPPSP, ID=10/3)
- Enable system-wide Deep Sleep with `system_set_sleep_parameters` (SSLP, ID=2/19)
- Enable flow control and optionally change UART parameters with `system_set_uart_parameters` (STU, ID=2/25)

#### 4.1.3 Host configuration

The external host must match EZ-Serial firmware platform for Ezurio Vela IF820 series module's configured UART communication. The factory default settings are 115200,8/N/1 with no flow control. However, you should enable and use flow control if the host supports it.

Use the host API library examples described in Host API Library to facilitate easy API communication between the host and the module, making sure to properly assert and de-assert the module's LP\_MODE pin as described in section [Connecting GPIO pins](#).

Monitor the CONNECTION signal for a simple indicator of BLE/BT connectivity without needing to parse all possible API events from the module. This can be especially helpful when using CYSPP/SPP mode.

### 4.2 Dumb terminal host with CYSPP and simple GPIO state indication

This application design example takes advantage of the factory-default EZ-Serial firmware platform for Ezurio Vela IF820 series module configuration and support for automatic CYSPP connectivity. It is best suited for applications where the external host cannot or does not need to impose any control over the EZ-Serial firmware platform for Ezurio Vela IF820 series module via API commands or events.

#### 4.2.1 Hardware design

Include the following design elements in your hardware:

- Module CYSPP pin to GND (force CYSPP data mode always, no API communication)
- Module UART\_TX pin to host UART RX pin
- Module UART\_RX pin to host UART TX pin
- Optional for flow control:
  - Module UART\_CTS pin to host UART RTS pin
  - Module UART\_RTS pin to host UART CTS pin
- Optional for connectivity status:
  - Module CONNECTION pin to LED (active LOW)

#### 4.2.2 Module configuration

The factory default configuration provides most of the behavior required. However, you may wish to make one or more of the following changes:

- Change device name with `gap_set_device_name` (SDN, ID=4/15)



Change CYSPP connection key and/or security requirements with `p_cyspp_set_parameters` (.CYSPPSP, ID=10/3)  
Change system sleep settings with `system_set_sleep_parameters` (SSLP, ID=2/19)  
Change UART baud or other parameters with `system_set_uart_parameters` (STU, ID=2/25)

#### 4.2.3 Host configuration

The external host must match EZ-Serial firmware platform for Ezurio Vela IF820 series module's configured UART communication. The factory-default settings are 115200,8/N/1 with no flow control. However, you should enable and use flow control if the host supports it.

If the host supports a simple "enable" control line for whether it is safe to send data, use the module's CONNECTION pin. This signal is asserted (LOW) only when the CYSPP data pipe is fully established.

### 4.3 Module-Only application with Beacon functionality

This application design example requires no special external hardware and only minimal initial configuration to define the type of beaconing desired.

#### 4.3.1 Hardware design

For correct operation, the module only requires power to the supply pins. You may also wish to include test pad or header access to the UART interface and status pins such as LP\_STATUS or CONNECTION during prototyping, because this can greatly simplify debugging if necessary.

#### 4.3.2 Module configuration

Make the following changes from the factory default configuration:

Disable CYSPP mode with `p_cyspp_set_parameters` (.CYSPPSP, ID=10/3)  
Enable system-wide sleep mode with `system_set_sleep_parameters` (SSLP, ID=2/19)  
Configure non-connectable (broadcast-only) with `gap_set_adv_parameters` (SAP, ID=4/23)  
Configure custom advertisement data with the appropriate beacon content using `gap_set_adv_data` (SAD, ID=4/19)

#### 4.3.3 Host configuration

The simple automatic beacon design does not require any host hardware, and therefore needs no host configuration.

## 5 Host API library

The host library implements a protocol parser/generator that communicates with the EZ-Serial firmware platform for Ezurio Vela IF820 series module using the API protocol. The library is usually written in standard C and wraps all API methods into easy-to-use command functions or response/event callbacks.

This section uses the [online host API library](#) for EZ-Serial firmware platform for Ezurio Vela IF820 series module to describe how to use the library as designed, how to port it to other platforms, or how to create your own library if the provided code is not suited for direct use or porting for any reason.

### 5.1 Host API library overview

#### 5.1.1 High level architecture

The host library communicates with the EZ-Serial firmware platform for Ezurio Vela IF820 series module, providing the host side of the command, response, or event communication mechanism that the module implements. The host must perform the following over the UART interface:

- Read and parse incoming data (may be either response or event packets)
- Validate packets using checksum
- Trigger application-defined callbacks when incoming packets arrive
- Generate and send outgoing data (command packets)

The protocol parser and generator on the module side strictly follow these rules:

- Events may be generated by the module at any time
- Every command received from the host immediately generates a response
- An event generated (for example, by a GPIO interrupt) while a command is being processed does not interrupt the command-response packet flow, but is sent out after the response packet is sent

The parser and generator on the host side must operate under these assumptions.

### 5.1.2 Host library design

Host communication with an EZ-Serial firmware platform for Ezurio Vela IF820 series module requires that only the incoming module-to-host byte stream is processed correctly, and that the outgoing host-to-module byte stream is properly formatted. To simplify this and provide a convenient layer of abstraction, the host API library provides a simple “parse” function for incoming bytes, and “wrapper” command functions that convert named parameter lists into binary packets ready for transmission.

Other than expecting standard C compiler functionality and little-endian byte order, the library is intentionally platform-agnostic. The source of incoming data does not matter; the internal methods process the data only after it arrives. The destination of outgoing data also does not matter; the internal methods perform only packetization and buffering of data so that it is ready to transmit. This improves portability because UART peripherals are accessed differently on different platforms, and a single library cannot provide support across all (or even very many) platforms if the UART peripheral implementation is built into the library itself.

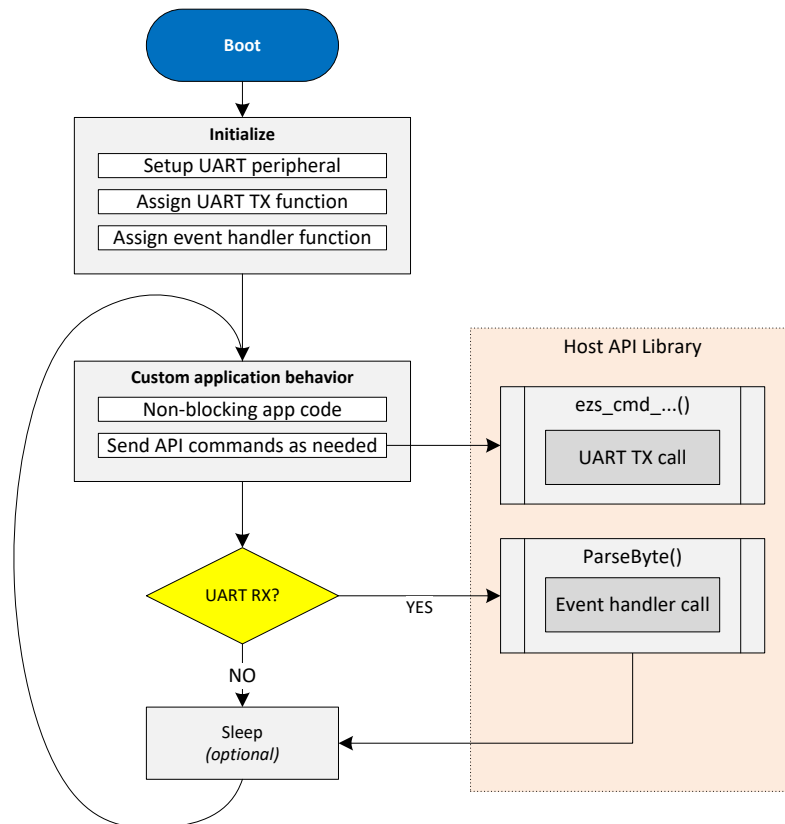
## 5.2 Implementing a project using the Host API library

### 5.2.1 Basic application architecture

Any host application that uses the EZ-Serial firmware platform for Ezurio Vela IF820 series module API library must follow the same basic behavior:

- Set up UART peripheral for incoming and outgoing data
- Assign hardware-specific input/output callback methods
- Monitor UART for incoming data, and send to parser
- Handle event/response packets sent to callback handler
- Call command wrapper functions as needed for application

This process is shown in [Figure 6](#).



**Figure 6: EZ-Serial firmware platform for Ezurio Vela IF820 series module Host API library application flow**

The host API library contains the core parsing and generating functions necessary to translate incoming data into callbacks and command function calls into binary packets.

## 5.2.2 Exposed API functions

The generic host API implementation written in C provides the following methods:

Function	Description
<code>EZSerial_Init</code>	Initializes parser and callback functions used for event handling, serial output, and serial input
<code>EZSerial_Parse</code>	Processes incoming bytes and triggers the event callback function when response or event packet is successfully processed
<code>EZSerial_FillPacketMetaFromBinary</code>	Fills binary packet metadata in <code>ezs_packet_t</code> structure based on the 4-byte binary packet header content (used internally within <code>EZSerial_Parse</code> )
<code>EZSerial_SendPacket</code>	Sends binary packet and checksum byte using the host-specific output callback function
<code>EZSerial_WaitForPacket</code>	Reads the data using the host-specific input callback function in a blocking or non-blocking way depending on the timeout argument (calls <code>EZSerial_Parse</code> as part of its functionality)

The application is responsible for providing implementation functions for three methods, assigned to the function pointers below:

Function	Description
<code>EZSerial_AppHandler</code>	Called whenever a valid incoming packet is observed.  This is strictly required in all cases. It is a core element of abstracting incoming packets into callback functions.
<code>EZSerial_HardwareOutput</code>	Called whenever the API generator needs to send data to the module over UART.  This is required if you intend to use the <code>EZSerial_SendPacket</code> method, or the <code>ezs_cmd_...</code> macros which also use that method. If you are manually sending well-formed binary command packet data directly from your own application, this may be assigned as NULL.
<code>EZSerial_HardwareInput</code>	Called whenever the API parser needs to read data from the module over UART.  This is required if you intend to use the <code>EZSerial_WaitForPacket</code> method, or the <code>EZS_WAIT...</code> or <code>EZS_CHECK...</code> macros which also use that method. If you are manually calling the <code>EZSerial_Parse</code> method after reading bytes in over UART, this may be assigned as NULL.

### 5.2.3 Command macros

To simplify binary packet creation, the library implements packet builder macros that match the protocol definitions for each command method. For example:

```
ezs_cmd_system_ping()
ezs_cmd_system_reboot()
ezs_cmd_gap_start_adv(mode, type, interval, channels, filter, timeout)
```

Commands which fall into the SET/GET categories and may access flash memory for retrieving or storing setting data have two separate command functions for each:

```
RAM: ezs_cmd_gatts_set_parameters(flags)
Flash: ezs_fcmd_gatts_set_parameters(flags)
```

To substantially reduce flash usage, the above commands are defined as macros that make use of a single function that accepts variable arguments:

```
ezs_output_result_t ezs_cmd_va(uint16 index, uint8 memory, ...)
```

This single method uses the supplied command table index (defined in the library header file as an enumerated list) and the packed binary protocol structure definition to determine the number of arguments needed for any given command and their data types.

This macro-based approach means that it is not possible to perform type checking at compile time, but it also means that the entire command generator implementation uses a tiny quantity of flash memory (well under 1KB as measured on one 8-bit MCU).

### 5.2.4 Convenience macros

If the hardware-specific input and output functions are correctly defined, the library also provides macros to further abstract common behavior into simpler code.

Function	Description
EZS_SEND_AND_WAIT (CMD, TIMEOUT)	Sends a command and then calls EZS_WAIT_FOR_RESPONSE
EZS_WAIT_FOR_PACKET (TIMEOUT)	Calls EZSerial_WaitForPacket with type set to any
EZS_WAIT_FOR_RESPONSE (TIMEOUT)	Calls EZSerial_WaitForPacket with type set to response
EZS_WAIT_FOR_EVENT (TIMEOUT)	Calls EZSerial_WaitForPacket with type set to event
EZS_CHECK_FOR_PACKET ()	Wrapper for EZS_WAIT_FOR_PACKET (0), a non-blocking attempt to read data

The assignable “return value” (evaluated expression result) for all these macros is a pointer to an `ezs_packet_t` object. If the process fails at any point for any reason—timeout, command transmission failure, incoming packet in progress, and so on—then the pointer value will be 0 (NULL).

## 5.3 Porting the Host API library to different platforms

The API protocol uses a packet byte stream, so the API host library expects matching byte ordering and packet structure mapping to avoid any extra processing overhead. The module (and low-level Bluetooth® spec) uses little-endian byte ordering, so the host must as well for all multi-byte integer data.

The example application code provided with the library to demonstrate EZ-Serial firmware platform for Ezurio Vela IF820 series module API usage includes a block of code that can verify proper support and configuration of byte ordering and structure packing. While it is not possible to provide a single, comprehensive cross-platform implementation of a structure packing macro due to variations between compilers, it is possible to definitively test whether the existing code will work properly. This can quickly identify and avoid potential problems that are otherwise very difficult to troubleshoot.

No special C extensions are used; tested compilers are GCC or GCC-compliant and follow the default C89 ruleset because no additional extensions are enabled.

## 5.4 Using the API definition JSON file to create a custom library

The JSON schema used for the API definition has the following structure:

- `info` (single dictionary)
- `date` – Definition revision date
- `version` – API protocol definition version
- `groups` (list of dictionaries) [...]
- `id` – Numeric ID assigned to group
- `name` – Alpha name assigned to group (for example, "gap")
- `commands` (list of dictionaries) [...]
- `id` – Numeric ID assigned to command
- `name` – Alpha name assigned to command (for example, "start\_adv")
- `flashopt` – Boolean flag indicating flash storage for settings
- `parameters` (list of dictionaries) [...]
- `type` – Data type (for example, "uint16")
- `name` – Alpha name assigned to parameter (for example, "mode")
- `textname` – text-mode equivalent (for example, "M")
- `required` – Boolean flag indicating optional or required parameter
- `format` – Intended data presentation format (for example, "string" or "hex")
- `default` – Fixed default value if optional parameter
- `returns` (list of dictionaries) [...see parameters...]
- `references` (single dictionary)
- `commands` (dictionary)
- `events` (dictionary)
- `events` (list of dictionaries) [...see commands...]

## 6 Troubleshooting

EZ-Serial firmware platform for Ezurio Vela IF820 series module is designed to be as robust and intuitive as possible, but it is always possible for something to go wrong. The instructions below can help narrow down the cause of failure in identify solutions in some cases.

### 6.1 UART communication issues

If you are unable to send or receive data as expected over the UART interface, perform the following steps:

1. Ensure that VDD and GND pins are properly connected (VDDR also requires power).
2. Ensure that VDD has a stable supply within the supported range (typically 3.3 V).
3. Ensure that UART data pins are properly connected:
  - a. Module UART\_RX to host TX
  - b. Module UART\_TX to host RX
4. If flow control is enabled or expected, ensure that the UART flow control pins are properly connected:
  - a. Module UART\_RTS to host CTS
  - b. Module UART\_CTS to host RTS
5. Ensure that the CYSPP pin is floating or HIGH to avoid entry into CYSPP mode. When CYSPP is active, API communication is disabled, and this can appear as a non-communicative state until a connection is established.
6. Drive or strongly pull the LP\_MODE pin HIGH to disable normal sleep mode. This is not necessary in most cases, but it can help eliminate potential uncertainty during testing.
7. Reset the module and monitor the UART\_TX pin during the boot process. If the module boots normally (CYSPP pin de-asserted), the **system\_boot** (BOOT, ID=2/1) API event should occur at the configured baud rate. With factory default settings, these values are 115200 baud and text mode. If possible, verify activity using an oscilloscope or a logic analyzer.
8. If attempting to communicate using the API protocol, ensure that your command packet structures are correct per the definitions in Section **Protocol structure and communication flow**.
9. If you are sending commands in binary mode and the commands in use have any variable-length arguments (data type of uint8a or longuint8a), ensure that the argument has the correct `<length> [data0, data1, ..., dataN]` format. Omitting the length byte will cause the API parser to interpret the packet incorrectly.

## 6.2 BLE connection issues

If you are unable to connect from a remote device, perform the following steps:

1. Ensure that the module is advertising in a connectable state. Start advertising specifically in the “connectable, undirected” mode using the `gap_start_adv` (IA, ID=4/8) API command, and watch for the expected `gap_adv_state_changed` (ASC, ID=4/2) API event indicating that the state actually changed to “active.”
2. Ensure you have set properly formed custom advertising data with `gap_set_adv_data` (SAD, ID=4/19) if you have disabled automatic advertising packet management with `gap_set_adv_parameters` (SAP, ID=4/23). Advertisement packets without a standard “Flags” field (usually [ 02 01 06 ]) do not appear in a generic scan. See section [Customizing advertisement and scanning response data](#).

## 6.3 GPIO signal issues

If you do not observe the expected behavior for GPIO input and/or output signals, perform the following steps:

1. Ensure that the pins that you have connected are correct based on your chosen module. See section [GPIO pin map](#) for per-device pin map details.
2. If a special-function output pin is not sufficiently driving a connected external device's input logic, ensure that the external device is not also attempting to drive or strongly pull the pin in the opposite direction at the same time.

## 7 API protocol reference

This section describes the API protocol that EZ-Serial firmware platform for Ezurio Vela IF820 series module uses. This protocol allows an external host to control the module, in addition to any GPIO signals involved in the design. The protocol follows a strict set of rules to make deterministic host-side behavior possible.

### 7.1 Protocol structure and communication flow

#### 7.1.1 API protocol formats

EZ-Serial firmware platform for Ezurio Vela IF820 series module implements a unified set of functionalities that can be accessed using binary API communication. Cypress text-based protocol APIs are also provided for ease of reading, as well as to generate binary API commands via the provided Python script.

##### 7.1.1.1 Text format overview

The text protocol definition is comprised entirely of printable ASCII characters for ease of use in terminal software. Response and Event packets sent from the module shall end with "\r\n" characters (0x0D, 0x0A). Commands sent to the module may end with either or both. Unlike the binary mode described below, the text protocol does not contain any checksum data or have a command entry timeout.

##### 7.1.1.2 Binary format overview

The binary protocol uses a fixed packet structure for every transaction in either direction. This fixed structure comprises a 4-byte header, followed by an optional payload of up to 2047 bytes (length specifier field is 11 bits wide).

Currently defined binary packet does not contain more than 520 payload bytes at this time, and very few packets contain more than 48. The API reference material below lists every fixed or minimum/maximum length value for all commands, responses, and events within the protocol.

The payload carries information related to the command, response, or event. If present, this payload always comes immediately after the header. All data in the payload is contained within one or more of the datatypes specified in section [API protocol data types](#).

To simplify the implementation of parsers and generators both inside the firmware and on external host microcontrollers, any packet may have a maximum of one variable-length data member (byte array or string), and if present, it must be the last element in the payload.

#### 7.1.2 API protocol data types

The data types implemented for individual parameters/arguments in the API protocol are described below, including representative text and binary examples.

In both text and binary modes, all negative numbers are represented in two's complement form. In this form, the MSb is the sign bit, which indicates a negative number if set. The remaining bits count upward from the bottom of the selected (positive or negative) range. For example, the value 0x80 is the bottom of the "int8" range, -128.

##### : API protocol data types

Type	Bytes	Description	Example
uint8	1	Unsigned 8-bit integer. Range is 0 to 255.	Text Mode: "10" = 0x10, decimal 16 "9A" = 0x9A, decimal 154  Binary Mode: [ 10 ] = 0x10, decimal 16 [ 9A ] = 0x9A, decimal 154

Type	Bytes	Description	Example
int8	1	Signed 8-bit integer. Range is -128 to 127.	Text Mode: "10" = 0x10, decimal 16 "9A" = 0x9A, decimal -102 Binary Mode: [ 10 ] = 0x10, decimal 16 [ 9A ] = 0x9A, decimal -102
uint16	2	Unsigned 16-bit integer. Range is 0 to 65,535.	Text Mode: "1234" = 0x1234, decimal 4,660 "9ABC" = 0x9ABC, decimal 39,612 Binary Mode: (little-endian) [ 34 12 ] = 0x1234, decimal 4,660 [ BC 9A ] = 0x9ABC, decimal 39,612
int16	2	Signed 16-bit integer. Range is -32,768 to 32,767.	Text Mode: "1234" = 0x1234, decimal 4,660 "9ABC" = 0x9ABC, decimal -25,924 Binary Mode: (little-endian) [ 34 12 ] = 0x10, decimal 4,660 [ BC 9A ] = 0x9ABC, decimal -25,924
uint32	4	Unsigned 32-bit integer. Range is 0 to 4,294,967,295.	Text Mode: "12345678" = 0x12345678 decimal 305,419,896 "9ABCDEF0" = 0x9ABCDEF0, decimal 2,596,069,104 Binary Mode: (little-endian) [ 78 56 34 12 ] = 0x12345678 decimal 305,419,896 [ F0 DE BC 9A ] = 0x9ABCDEF0 decimal 2,596,069,104
int32	4	Signed 32-bit integer. Range is -2,147,438,648 to 2,147,483,647.	Text Mode: "12345678" = 0x12345678 decimal 305,419,896 "9ABCDEF0" = 0x9ABCDEF0, decimal -1,698,898,192 Binary Mode: (little-endian) [ 78 56 34 12 ] = 0x12345678 decimal 305,419,896 [ F0 DE BC 9A ] = 0x9ABCDEF0 decimal -1,698,898,192
macaddr	6	48-bit MAC address.	Text Mode: "112233AABBCC" = 11:22:33:AA:BB:CC Binary Mode: (little-endian) [ CC BB AA 33 22 11 ] = 11:22:33:AA:BB:CC
uint8a	1+	Array of uint8 bytes, with prefixed one-byte length value. Supported length is 0-255 bytes.	Text Mode: (length omitted, detected automatically) "41424344" = Length 4, Data [ 41 42 43 44 ] "1122334455" = Length 5, Data [ 11 22 33 44 55 ] Binary Mode: [ 04 41 42 43 44 ] = Ln. 4, [ 41 42 43 44 ] [ 05 11 22 33 44 55 ] = Ln. 5, [ 11 22 33 44 55 ]
longuint8a	2+	Array of uint8 bytes, with prefixed two-byte length value.	Text Mode: (length omitted, detected automatically) "41424344" = Length 4, Data [ 41 42 43 44 ] "1122334455" = Length 5, Data [ 11 22 33 44 55 ]



Type	Bytes	Description	Example
		Supported length is 0-65535 bytes.	Binary Mode: [ 04 00 41 42 43 44 ] = Length 4, Data [ 41 42 43 44 ] [ 05 00 11 22 33 44 55 ] = Length 5, Data [ 11 22 33 44 55 ]  Note the 16-bit length prefix in binary mode is transmitted in little-endian byte order, so the value 0x0005 is sent as [ 05 00 ].
string	1+	String of uint8 bytes, with prefixed one-byte length value. Length is 0-255 bytes.	These two datatypes are represented in binary the same way as uint8a and longuint8a data, but in text mode they are entered and displayed exactly as-is, with the assumption that they contain printable ASCII characters. An example of a string value entered and displayed in this way is the Device Name value.

### 7.1.3 Binary format details

#### 7.1.3.1 Byte ordering and structure packing

The protocol implements a collection of common data types representing signed and unsigned integers, arrays of binary bytes, arrays of printable characters, and certain technology-specific data (6-byte MAC address).

In text mode, all data except string/longstring values are represented as ASCII hexadecimal characters, without a leading "0x" or other prefix. For example, the decimal value 154 is shown or entered as "9A". Leading zeros may be omitted. Also, in text mode, all multi-byte integer and MAC address data shall be entered in big-endian byte order. For example, the value 0x1234 is entered or displayed as "1234". The MAC address 11:22:33:AA:BB:CC is entered or displayed as "112233AABBCC".

In binary mode, all multi-byte integers and MAC address data must be transmitted serially in little-endian byte order. For example, the value 0x1234 is two bytes and transmitted as [ 34 12 ], and the MAC address 11:22:33:AA:BB:CC is six bytes and transmitted as [ CC BB AA 33 22 11 ].

The Bluetooth® Low Energy specification mandates little-endian byte order internally, so data from the stack is naturally presented to the application layer in this byte order. Further, many common embedded processors use little-endian data storage. As a result, host MCU firmware can read in a serial byte stream into a contiguous SRAM buffer, and define a structure like the following:

```
typedef struct {
    uint16 app;
    uint32 stack;
    uint16 protocol;
    uint8 hardware;
    uint8 cause;
    macaddr address;
} ezs_evt_system_boot_t;
```

The host MCU application can directly map this structure onto the packet buffer in memory with no additional byte-swap operations. Accessing any one of the structure members gives correct access to the data in the packet. This arrangement allows for minimal flash usage and CPU execution time.

### 7.1.3.2 Binary packet header

The binary packet 4-byte header structure is described below.

#### : Binary packet header structure

Byte	Field(s)	Description
0	[7:6] - Type [5:4] - Memory [2:0] - Length MSB	<p><b>Type:</b> The "Type" field is a 2-bit value (MSb aligned) indicating whether the packet is a command, response, or event. Options are as follows:</p> <ul style="list-style-type: none"> <li>00: RESERVED</li> <li>01: RESERVED</li> <li>10: Event (module-to-host)</li> <li>11: Response (module-to-host) and Command (host-to-module)</li> </ul> <p>Protocol methods follow this convention when the "Type" value is aligned properly:</p> <ul style="list-style-type: none"> <li>Commands sent to the module begin with 0xC0</li> <li>Responses sent to the host begin with 0xC0</li> <li>Events sent to the host begin with 0x80</li> </ul> <p><b>Memory:</b> The "Memory" field is a 2-bit value (MSb aligned) indicating whether a sent command accesses the runtime value stored in RAM or the boot value stored in flash. This field is ignored for commands which do not read or write configuration data stored in either flash or RAM. Options are as follows:</p> <ul style="list-style-type: none"> <li>00: Runtime (RAM)</li> <li>01: Boot (Flash)</li> <li>10: RESERVED</li> <li>11: RESERVED</li> </ul> <p>The values stored in RAM and flash may be the same, if you have not modified the runtime value separately from the boot value since the last power-on or reset.</p> <p>Length MSB: The length MSB field contains the upper three bits of the payload length value (11 bits total). See below for length detail.</p> <p>The "Type", "Memory", and "Length MSB" bitfields are positioned within Byte 0 as follows:</p> <pre>0b TTMM 0LLL</pre> <p>The remaining bit in the middle is currently reserved and should always be set to zero.</p>
1	Length LSB	<p>This value indicates the number of bytes in the payload. It may be 0 to indicate no payload, or any value up to the 11-bit maximum of 2047 (combining the LSB and MSB fields together).</p> <p>Typically, packets fit easily within a 64-byte buffer. However, a few packets such as local GATT reads and writes may potentially be much longer than this. Protocol methods which may require or generate atypically long packets are documented specifically.</p>
2	Group ID	<p>All protocol methods are organized into logically separate groups, such as GAP, GATT Server, CYSPP, and so on. This byte represents the group ID, between 0 and 255.</p>

Byte	Field(s)	Description
		A single group ID applies to all commands, responses, and events within that group.
3	Method ID	Within each group and packet type, every protocol method has a unique ID between 0 and 255. Command/response pairs always have matching IDs. Command/response pairs and events are separate collections and may have overlapping method IDs, each in a set starting from 0.

## 7.2 API commands and responses

All commands and responses implemented in the API protocol are described in detail below. API events are documented separately in section 0. A master list of all possible error codes resulting from commands can be found in section [Error codes](#).

Important things to note about the reference material in the following sections:

The 16-bit “result” code is common to every response, and always occupies the same position in the packet (immediately after the binary header or text name). For simplicity, this “result” field is omitted from each list of response parameters in the tables below.

The “Text” column in each “Command Arguments” table contains the text code for each argument. Required arguments have a red asterisk (\*) next to their text codes. Optional arguments in text mode will not have a red asterisk.

All command arguments are required in binary mode, because binary parsing depends on predictable argument position and byte width for proper data identification and unpacking.

The “Command-Specific Result Codes” list appearing for some commands do not include some errors that may result from command entry or protocol format mistakes. These common errors include:

```

0x0203 - EZS_ERR_PROTOCOL_UNRECOGNIZED_COMMAND
0x0206 - EZS_ERR_PROTOCOL_SYNTAX_ERROR
0x0207 - EZS_ERR_PROTOCOL_COMMAND_TIMEOUT
0x0209 - EZS_ERR_PROTOCOL_INVALID_CHECKSUM
0x020A - EZS_ERR_PROTOCOL_INVALID_COMMAND_LENGTH
0x020B - EZS_ERR_PROTOCOL_INVALID_PARAMETER_COUNT
0x020C - EZS_ERR_PROTOCOL_INVALID_PARAMETER_VALUE
0x020D - EZS_ERR_PROTOCOL_MISSING_REQUIRED_ARGUMENT
0x020E - EZS_ERR_PROTOCOL_INVALID_HEXADEDECIMAL_DATA
0x020F - EZS_ERR_PROTOCOL_INVALID_ESCAPE_SEQUENCE
0x0210 - EZS_ERR_PROTOCOL_INVALID_MACRO_SEQUENCE

```

See section [Error codes](#) for details on these and other error codes.

Commands and responses are broken down into the following groups:

- Protocol group (ID=1)
- System group (ID=2)
- GAP Group (ID=4)
- GATT Server Group (ID=5)
- GATT Client Group (ID=6)
- SMP Group (ID=7)
- GPIO Group (ID=9)
- CYSPP Group (ID=10)
- BT group (ID=14)
- Spp group (ID=19)

### 7.2.1 Protocol group (ID=1)

Protocol methods allow you to change the way the API protocol operates while communicating with an external host over the serial interface.

Commands within this group are listed below:

```

protocol_set_parse_mode (SPPM, ID=1/1)
protocol_get_parse_mode (GPPM, ID=1/2)
protocol_set_echo_mode (SPEM, ID=1/3)
protocol_get_echo_mode (GPEM, ID=1/4)

```

Events within this group are documented in section [System Group \(ID=2\)](#).

### 7.2.1.1 *protocol\_set\_parse\_mode (SPPM, ID=1/1)*

Configure new protocol parse mode and transparent mode.

In binary mode, all API packets to and from the module must use a binary format with a fixed header and payload structure, as described in the reference material. In text mode, all commands, responses, and events use a human-readable format that is suitable for typing in a terminal. See section [Protocol structure and communication flow](#) for details.

In non-transparent mode, though SPP is connected, user still can use command mode. Data receive from SPP connection will arrive with event ".SPPD".

**Note:** When the protocol mode is changed with this command, the effect is immediate. The response packet returned will come in the newly configured format, not the previous format.

#### : Binary header

	Type	Length	Group	ID	Notes
CMD	C0	01	01	01	None
RSP	C0	02	01	01	None

#### Text info

Text name	Response length	Category	Notes
SPPM	0x000A	SET	None

#### Command arguments

Data type	Name	Text	Description
uint8	mode	M	<p>New parse and transparent mode:</p> <p>Bit 0 - Text or Binary Mode</p> <p>0 = Text mode (factory default)</p> <p>1 = Binary mode</p> <p>Bit 1: Disable Transparent transmission mode for SPP</p> <p>0: It does not include header for SPP</p> <p>1: It includes header for SPP</p>

#### Response parameters

None.

#### Related commands

- [protocol\\_get\\_parse\\_mode \(GPPM, ID=1/2\)](#)

### 7.2.1.2 *protocol\_get\_parse\_mode (GPPM, ID=1/2)*

Obtain current protocol parse mode.

#### : Binary header

	Type	Length	Group	ID	Notes
CMD	C0	00	01	02	None
RSP	C0	03	01	02	None

#### Text info

Text name	Response length	Category	Notes
GPPM	0x000F	GET	None

#### Command arguments

None.

#### Response parameters

Data type	Name	Text	Description
uint8	mode	M	<p>Current parse and transparent mode:</p> <p>Bit 0 - Text or Binary Mode</p> <p>0 = Text mode (factory default)</p> <p>1 = Binary mode</p> <p>Bit 1: Disable Transparent transmission mode for SPP/CYSPP mode</p> <p>0: It does not include header for SPP/CYSPP data</p> <p>1: It includes header for SPP/CYSPP data</p>

#### Related commands

*protocol\_get\_parse\_mode (GPPM, ID=1/2)*

### 7.2.1.3 *protocol\_set\_echo\_mode (SPEM, ID=1/3)*

Configure new protocol echo mode.

The protocol echo mode applies when using text mode API protocol over UART to communicate with the module. Enabling echo will result in each input byte being sent back to the host after it is parsed. Local echo may be desirable during a terminal session, but it is typically simpler disable it for MCU communication so that the MCU only needs to parse response and event data.

**Note:** Local echo does not apply in CYSPP data mode or CYCommand data mode, regardless of the protocol format in use. It only affects communication over the UART interface when using the API protocol in text mode.

#### : Binary header

	Type	Length	Group	ID	Notes
CMD	C0	01	01	03	None
RSP	C0	02	01	03	None

#### Text info

Text name	Response length	Category	Notes
SPEM	0x000A	SET	None

**Command arguments**

Data type	Name	Text	Description
uint8	mode	M	New echo mode: 0 = Disabled 1 = Enabled (factory default)

**Response parameters:**

None.

**Related commands:***protocol\_get\_echo\_mode (GPEM, ID=1/4)***7.2.1.4 *protocol\_get\_echo\_mode (GPEM, ID=1/4)***

Obtain current protocol echo mode.

**: Binary header**

	Type	Length	Group	ID	Notes
CMD	C0	00	01	04	None
RSP	C0	03	01	04	None

**Text info**

Text name	Response length	Category	Notes
GPEM	0x000F	GET	None

**Command arguments**

None.

**Response parameters**

Data type	Name	Text	Description
uint8	mode	M	Current echo mode: 0 = Disabled 1 = Enabled (factory default)

**Related commands:***protocol\_set\_echo\_mode (SPEM, ID=1/3)*

### 7.2.2 System group (ID=2)

System methods relate to the core device and describe functionality such as boot status, setting or obtaining device address information, and resetting to an initial state.

Commands within this group are listed below:

```
system_ping (/PING, ID=2/1)
system_reboot (/RBT, ID=2/2)
system_dump (/DUMP, ID=2/3)
system_store_config (/SCFG, ID=2/4)
system_factory_reset (/RFAC, ID=2/5)
system_query_firmware_version (/QFV, ID=2/6)
system_query_random_number (/QRND, ID=2/8)
system_write_user_data (/WUD, ID=2/11)
system_read_user_data (/RUD, ID=2/12)
system_set_bluetooth_address (SBA, ID=2/13)
system_get_bluetooth_address (GBA, ID=2/14)
system_set_sleep_parameters (SSLP, ID=2/19)
system_get_sleep_parameters (GSLP, ID=2/20)
system_set_tx_power (STXP, ID=2/21)
system_get_tx_power (GTXP, ID=2/22)
system_set_transport (ST, ID=2/23)
system_get_transport (GT, ID=2/24)
system_set_uart_parameters (STU, ID=2/25)
system_get_uart_parameters (GTU, ID=2/26)
```

Events within this group are documented in section [System Group \(ID=2\)](#).

#### 7.2.2.1 *system\_ping (/PING, ID=2/1)*

Test API communication.

Pinging the module verifies that the host and the module can communicate properly in API mode. The module should immediately generate a well-formed response to this command if communication is working correctly. Host-side initialization routines often begin with this step.

Runtime values returned in the response to this command are calculated based on the built-in 32768-Hz watch clock oscillator (WCO) that is used to manage low-power operation of the BLE stack. No external hardware is required for this functionality.

Pinging the module does not serve any purpose other than to verify proper communication, or to obtain runtime since reset. You do not need to ping at regular intervals to keep a connection alive or prevent the module from entering low-power states. The platform automatically maintains BLE connections unless commanded otherwise. See section [Managing sleep states](#) for details of sleep behavior.

#### : Binary header

	Type	Length	Group	ID	Notes
CMD	C0	00	02	01	None
RSP	C0	08	02	01	None

#### Text info

Text name	Response length	Category	Notes
/PING	0x000B	ACTION	None

#### Command arguments

None.

### Response parameters

Data type	Name	Text	Description
uint32	runtime	R	Number of seconds since boot
uint16	fraction	F	Fraction of a second (units are ms)

#### 7.2.2.2 *system\_reboot (/RBT, ID=2/2)*

Reboot module.

A module reboot takes effect immediately. Any configuration settings not stored in flash revert to their boot-level values, and any active connections are terminated without clean closure (remote peer will detect a supervision timeout). See section [Saving runtime settings in flash](#) for details about how to store settings in flash to make them persist across reboots and power cycles.

#### : Binary header

	Type	Length	Group	ID	Notes
CMD	C0	00	02	02	None
RSP	C0	02	02	02	None

#### Text info

Text name	Response length	Category	Notes
/RBT	0x0000A	ACTION	None

#### Command arguments

None.

#### Response parameters

None.

#### Related commands

*system\_store\_config (/SCFG, ID=2/4)* – Used to store all configuration items in flash before rebooting, if desired

#### Related events

*system\_boot (BOOT, ID=2/1)* – Occurs once the reboot process completes

#### 7.2.2.3 *system\_dump (/DUMP, ID=2/3)*

Dump current device configuration or state information.

Performing a system dump generates a sequence of *system\_dump\_blob (DBLOB, ID=2/5)* API events, each containing up to 16 bytes, until all data transmission is complete. You can provide this information for troubleshooting if requested by Infineon support staff.

#### : Binary header

	Type	Length	Group	ID	Notes
CMD	C0	01	02	03	None
RSP	C0	04	02	03	None

#### Text info



Text name	Response length	Category	Notes
/DUMP	0x0012	ACTION	None

## Command arguments

Data type	Name	Text	Description
uint8	type	T	Type of information to dump: 0 = Runtime configuration data (default) 1 = Boot-level configuration data 2 = Factory-level configuration data 3 = System state data

## Response parameters

Data type	Name	Text	Description
UInt16	Length	L	Number of bytes to be dumped: Configuration data is 674 bytes (0x02A2) State data is 1,955 bytes (0x07A3)

## Related commands

[system\\_store\\_config \(/SCFG, ID=2/4\)](#)

## Related events

[system\\_dump\\_blob \(DBLOB, ID=2/5\)](#)

#### 7.2.2.4 [system\\_store\\_config \(/SCFG, ID=2/4\)](#)

Store all configuration settings into flash.

This command applies all runtime settings into the boot-level configuration area stored in non-volatile flash. See section [Configuration settings, storage, and protection](#) for details about different configuration areas.

*This command briefly halts CPU execution, and may cause connectivity loss for any open connections if this occurs during a precise moment when low-level BLE interrupts require processing. If possible, use this command only while not connected to avoid this potential issue.*

## : Binary header

	Type	Length	Group	ID	Notes
CMD	C0	00	02	04	None
RSP	C0	02	02	04	None

## Text info

Text name	Response length	Category	Notes
/SCFG	0x000B	ACTION	None

## Command arguments

None.

## Response parameters

None.

## Related commands

`system_factory_reset (/RFAC, ID=2/5)`

### 7.2.2.5 `system_factory_reset (/RFAC, ID=2/5)`

Reset all settings to factory defaults and reboot.

This command reverts all configuration settings back to the values stored in the factory default area. After applying these default values, the system reboots immediately.

*If you have configured custom serial communication settings using the `system_set_transport (ST, ID=2/23)` API command, using this command will undo these changes and may prevent a working communication until you reconfigure your host device to the factory default transport settings. See section **Factory default behavior** for details about these settings.*

#### : Binary header

	Type	Length	Group	ID	Notes
CMD	C0	00	02	05	None
RSP	C0	02	02	05	None

#### Text info

Text name	Response length	Category	Notes
/RFAC	0x000B	ACTION	None

#### Command arguments

None.

#### Response parameters

None.

#### Related events

`system_factory_reset_complete (RFAC, ID=2/3)` – Occurs after the settings are reset  
`system_boot (BOOT, ID=2/1)` – Occurs after the system reboots

### 7.2.2.6 `system_query_firmware_version (/QFV, ID=2/6)`

Query EZ-Serial firmware platform for Ezurio Vela IF820 series module firmware version info.

This command provides the same version details that the `system_boot (BOOT, ID=2/1)` event contains.

#### : Binary header

	Type	Length	Group	ID	Notes
CMD	C0	00	02	06	None
RSP	C0	0D	02	06	None

#### Text info

Text name	Response length	Category	Notes
/QFV	0x002C	ACTION	None

### Command arguments

None.

### Response parameters

Data type	Name	Text	Description
uint32	App	E	Application version number (for example, 0x0101021F = 1.1.2 build 31)
uint32	stack	S	BLE stack version number (for example, 0x02020355 = 2.2.3 build 85)
uint16	protocol	P	API protocol version number (for example, 0x0103 = 1.3)
uint8	hardware	H	Hardware identifier: 0xF1-CYW20820

### Related commands

`system_boot` (BOOT, ID=2/1)

#### 7.2.2.7 `system_query_random_number` (/QRND, ID=2/8)

Query random number generator for 8-byte pseudo-random sequence.

This command provides simple access to the random number generator in the Ezurio Vela IF820 series module's chipset. The query always provides exactly eight bytes of random data.

**Note:** This pseudo-random generation mechanism is FIPS PUB 140-2 compliant.

### : Binary header

	Type	Length	Group	ID	Notes
CMD	C0	00	02	08	None
RSP	C0	0B	02	08	None

### Text info

Text name	Response length	Category	Notes
/QRND	0x001C	ACTION	None

### Command arguments

None.

### Response parameters

Data type	Name	Text	Description
uint8a	data	D	Random 8-byte sequence (1 length byte equal to 0x08, followed by 8 data bytes)

*Note:* uint8a data type requires one prefixed "length" byte before binary parameter payload

#### 7.2.2.8 *system\_write\_user\_data (/WUD, ID=2/11)*

Write arbitrary data to the user flash storage area.

EZ-Serial firmware platform for Ezurio Vela IF820 series module provides 256 bytes of non-volatile flash storage for application data. This command allows writing 1-32 bytes to any position within this 256-byte area.

**Note:** You must specify a data offset and length which do not exceed 256 when combined. For example, if you are writing 32 bytes of data, the specified "offset" argument must be 224 (0xE0) or less.

#### : Binary header

	Type	Length	Group	ID	Notes
CMD	C0	04~23	02	0B	Variable-length command payload, minimum of 4 (0x4), maximum of 35 (0x23)
RSP	C0	02	02	0B	None

#### Text info

Text name	Response length	Category	Notes
/WUD	0x000A	ACTION	None.

#### Command arguments

Data type	Name	Text	Description
uint16	offset	O*	Offset (0-65535)

*Note:* See details for difference case.

UInt8	Mode	M	<p>Operation Mode(0~6):</p> <p>0 = Write user data (default)</p> <p>O: Offset from 0-0xFF</p> <p>D: Data to write</p> <p>1 = Write register. Start from: 0x320000+offset</p> <hr/> <p><b>Note:</b> Obsolete, not implemented.</p> <hr/> <p>O: Offset from 0-0xFFFF</p> <p>D: Data to write. Data length is only 4 bytes</p> <p>2 = Write RAM. Start from: 0x00220000+offset</p> <hr/> <p><b>Note:</b> Obsolete, not implemented.</p> <hr/> <p>O: Offset from 0-0xFFFF</p> <p>D: Data to write</p> <p>3 = Add Init command to list</p> <p>O: Offset in history command list.</p> <p>D: N/A</p> <p>4 = Delete Init command to list.</p> <p>O: Init command number,</p> <p>D: N/A</p> <p>5 = Reset Init command list.</p> <p>O: If 0xFFFF, specially disable Init command list, otherwise delete all Init command list content</p> <p>D: N/A</p> <p>6 = Write register2. Start from: 0x330000+offset</p>
-------	------	---	---

Data type	Name	Text	Description
			<b>Note:</b> Obsolete, not implemented.  O: Offset from 0-0xFFFF D: Data to write
uint8a	data	D*	Data to write (1-32 bytes)
			<b>Note:</b> uint8a data type requires one prefixed "length" byte before binary parameter payload  Detail depends on Mode

#### Response parameters

None.

#### Related commands

`system_read_user_data (/RUD, ID=2/12)`

##### 7.2.2.9 `system_read_user_data (/RUD, ID=2/12)`

Read arbitrary data from the user flash storage area.

EZ-Serial firmware platform for Ezurio Vela IF820 series module provides 256 bytes of non-volatile flash storage for application data. This command allows reading 1-32 bytes from any position within this 256-byte area.

**Note:** You must specify a data offset and length which do not exceed 256 when combined. For example, if you are reading 32 bytes of data, the specified "offset" argument must be 224 (0xE0) or less.

#### : Binary header

	Type	Length	Group	ID	Notes
CMD	C0	03	02	0C	None.
RSP	C0	03	02	0C	Variable-length response payload, minimum of 3 (0x3), maximum of 35 (0x23).

#### Text info

Text name	Response length	Category	Notes
/RUD	0x000D–0x004D	ACTION	Variable-length response payload, minimum of 13 (0xD), maximum of 77 (0x4D).

#### Command arguments

Data type	Name	Text	Description
uint16	offset	O*	Offset (0-65535)
UInt8	Mode	M	Operation Mode(0~6): 0 = Read user data (default) O: Offset from 0-0xFF D: Data read 1 = Read register. Start from: 0x320000+offset ( <b>Note:</b> Obsolete, not implemented). O: Offset from 0-0xFFFF D: Data to Read 2 = Read RAM. Start from: 0x00220000+offset

Data type	Name	Text	Description
			<b>Note:</b> Obsolete, not implemented. <hr/> O: Offset from 0-0xFFFF D: Data to Read 3 = Read Init command O: Init command number, D: Command content to Read 4 = Read Init command list info. O: N/A D: Init command list info Byte 0: disable Byte 1: Init command total number Byte 2&3: Next Init command write location 5 = Print current init command list. O: N/A D: N/A 6 = Read register 2. Start from: 0x330000+offset <b>Note:</b> Obsolete, not implemented. O: Offset from 0-0xFFFF D: Data to Read
uint8	length	L*	Number of bytes to read (1-32) Only valid for M=0,2

#### Response parameters

Data type	Name	Text	Description
uint8a	data	D	Data read (1-32 bytes)
			<b>Note:</b> uint8a data type requires one prefixed "length" byte before binary parameter payload <hr/> Detail depend on M

#### Related commands

[system\\_write\\_user\\_data \(/WUD, ID=2/11\)](#)

#### 7.2.2.10 [system\\_set\\_bluetooth\\_address \(SBA, ID=2/13\)](#)

Configure a new Bluetooth® address.

This address is visible to remote scanning or connected devices, if the module is not operating with privacy enabled.

**Note:** When privacy is enabled, remote peer devices see a random address instead of the fixed address. Central or Peripheral privacy is not the same as encryption. See related commands and example usage for detail.

#### : Binary header

	Type	Length	Group	ID	Notes
CMD	C0	06	02	0D	None.
RSP	C0	02	02	0D	None.

## Text info

Text name	Response length	Category	Notes
SBA	0x0009	SET	None.

## Command arguments

Data type	Name	Text	Description
Macaddr	address	A	New Bluetooth® address. Set all six 0x00 bytes to revert to factory-provided address.

## Response parameters

None.

## Related commands

[system\\_get\\_bluetooth\\_address](#) (GBA, ID=2/14)

[smp\\_set\\_privacy\\_mode](#) (SPRV, ID=7/9)

### 7.2.2.11 [system\\_get\\_bluetooth\\_address](#) (GBA, ID=2/14)

Obtain the current Bluetooth® address.

## : Binary header

	Type	Length	Group	ID	Notes
CMD	C0	00	02	0E	None.
RSP	C0	08	02	0E	None.

## Text info

Text name	Response length	Category	Notes
GBA	0x0018	GET	None.

## Command arguments

None.

## Response parameters

Data type	Name	Text	Description
Macaddr	address	A	Current Bluetooth® address

## Related commands

[system\\_get\\_bluetooth\\_address](#) (GBA, ID=2/14)

[smp\\_set\\_privacy\\_mode](#) (SPRV, ID=7/9)

### 7.2.2.12 `system_set_sleep_parameters` (SSLP, ID=2/19)

Configure new system-wide sleep settings.

To maintain the required activity (including BLE communication, PWM output, and UART output), EZ-Serial firmware platform for Ezurio Vela IF820 series module will not automatically enter Deep Sleep mode even if it is configured as normal sleep mode.

: Binary header

	Type	Length	Group	ID	Notes
CMD	C0	03	02	13	None.
RSP	C0	02	02	13	None.

Text info

Text name	Response length	Category	Notes
SSLP	0x000A	SET	None.

Command arguments

Data type	Name	Text	Description
uint8	Level	L	New maximum system-wide sleep level: 0 = Sleep disabled 1 = EPDS when possible (factory default) 2 = Hibernate when possible
uint16	hid_off_sleep_time	T	hid_off_sleep_time in second 0: Not set Other: Set hid off wake up time

Response parameters

None.

Related commands

`system_get_sleep_parameters` (GSLP, ID=2/20)

`gpio_set_pwm_mode` (SPWM, ID=9/11) – Configure PWM output

`p_cyspp_set_parameters` (.CYSPPSP, ID=10/3) – Configure new CYSPP parameters, including CYSPP data mode sleep level

Example usage

Section [Configuring the system-wide sleep level](#)



### 7.2.2.13 *system\_get\_sleep\_parameters* (GSLP, ID=2/20)

Obtain the current system-wide sleep settings.

#### *: Binary header*

	Type	Length	Group	ID	Notes
CMD	C0	00	02	14	None.
RSP	C0	05	02	14	None.

#### Text info

Text name	Response length	Category	Notes
GSLP	0x000F	GET	None.

#### Command arguments

None.

#### Response parameters

Data type	Name	Text	Description
uint8	Level	L	Current maximum system-wide sleep level: 0 = Sleep disabled (factory default) 1 = EPDS when possible 2 = Hibernate when possible
uint16	hid_off_sleep_time	T	hid_off_sleep_time in second 0: Not set Other: Set hid off wake up time

#### Related commands

*system\_set\_sleep\_parameters* (SSLP, ID=2/19)

**7.2.2.14** *system\_set\_tx\_power (STXP, ID=2/21)*

Configure new transmit power for all outgoing radio communications.

This power setting affects all transmissions, including advertising, scan requests and connection requests, and all packets sent during an active connection. Changes take effect as soon as the next transmitted packet begins.

**: Binary header**

	Type	Length	Group	ID	Notes
CMD	C0	02+	02	15	Variable-length command payload, value specified is minimum.
RSP	C0	02	02	15	None.

**Text info**

Text name	Response length	Category	Notes
STXP	0x000A	SET	None.

**: Command arguments**

Data type	Name	Text	Description
uint8	Power	P	Available power value can be set. See section 3.1.4 for details on the TX output power map.  New transmit power 0: set power level array: 1~8: set power level, the default is 7.
uint8a	power_array	D	Array for power value, valid only if P=0.  There is a 3*8 bytes array for power level.  0 - 7 bytes for BR 8 - 15 bytes for EDR 16- 23 bytes for BLE

**Response parameters**

None.

**Related commands**

*system\_get\_tx\_power (GTXP, ID=2/22)*

### 7.2.2.15 `system_get_tx_power` (GTXP, ID=2/22)

Obtain current transmit power for all outgoing radio communications.

#### : Binary header

	Type	Length	Group	ID	Notes
CMD	C0	00	02	16	None.
RSP	C0	04+	02	16	Variable-length response payload, value specified is minimum.

#### Text info

Text name	Response length	Category	Notes
GTXP	0x0012+	GET	Variable-length response payload, value specified is minimum.

#### : Command arguments

None.

#### Response parameters

Data type	Name	Text	Description
uint8	Power	P	Current active power level value, it should be in the range of 1 and 8. See 3.1.4 for details on the TX output power map.
uint8a	power_array	D	Array for power value there is a 3*8 bytes array for power level. 0 - 7 bytes for BR 8 - 15 bytes for EDR 16- 23 bytes for BLE

#### Related commands

`system_get_tx_power` (GTXP, ID=2/22)

### 7.2.2.16 system\_set\_transport (ST, ID=2/23)

Configure new host communication interface.

This command configures the interface used for wired external host communication. If a change is successful, EZ-Serial firmware platform for Ezurio Vela IF820 series module will send the response packet in the *original* configuration, and then switch to the new transport interface.

**Note:** The current EZ-Serial firmware platform for Ezurio Vela IF820 series module release supports only the UART transport interface. No other options are available.

#### : Binary header

	Type	Length	Group	ID	Notes
CMD	C0	0C+	02	17	Variable-length command payload, value specified is minimum.
RSP	C0	02	02	17	None.

#### Text info

Text name	Response length	Category	Notes
ST	0x0008	SET	None.

#### : Command arguments

Data type	Name	Text	Description
uint8	interface	I	New host transport interface: 1 = UART (factory default)
uint8	cmd_channel	N	cmd_channel: Not implemented
uint8	spp_route	S	spp route: Not implemented
uint8	cyspp_route	C	cyspp route: Not implemented
uint8	BT flag	T	To control BT behavior when BLE is connected. The default value is 0x80: Non-discoverable and Not connectable for BT when BLE link is connected Bit 7: control if this flag is valid 0: Not valid 1: Valid Bit 4: Set connectability 0, Not connectable 1, BT Classic connectable Bit 1~0: Set discoverability 0, Non-discoverable 1, Limited BT Classic discoverable 2, General BT Classic discoverable
uint8	BLE flag	L	To control BLE behavior when BT SPP is connected. The default value is 0x80: No ADV broadcast when BT SPP link is connected Bit 7: control if this flag is valid

Data type	Name	Text	Description
			0: Not valid 1: Valid Bit 3~0: advType after BT SPP is connected 0, Stop advertisement 1, Directed advertisement (high duty cycle) 2, Directed advertisement (low duty cycle) 3, Undirected advertisement (high duty cycle) 4, Undirected advertisement (low duty cycle) 5, Non-connectable advertisement (high duty cycle) 6, Non-connectable advertisement (low duty cycle) 7, Discoverable advertisement (high duty cycle) 8, Discoverable advertisement (low duty cycle)
uint8	active_time_due_puart	A	Active kept time after received data from PUART in low power state. Unit is second.  <b>Note:</b> Not implemented
uint8	event_filter	E	event filter 0: Not set 1: Set, command response and event will not appear in command terminal  <b>Note:</b> Factory default = 0 (Not set)
uint8a	event_array	D	response&events which are not filtered: 12 bytes in max, so it can set 6 non-filtered response&events

### Response parameters

None.

### Related commands

[system\\_get\\_transport \(GT, ID=2/24\)](#)  
[system\\_set\\_uart\\_parameters \(STU, ID=2/25\)](#)

### 7.2.2.17 system\_get\_transport (GT, ID=2/24)

Obtain the current host transport setting.

#### Binary header

	Type	Length	Group	ID	Notes
CMD	C0	00	02	18	None.
RSP	C0	0E+	02	18	Variable-length response payload, value specified is minimum.

#### Text info

Text name	Response length	Category	Notes
GT	0x000D	GET	None.

#### : Command arguments

None.

#### Response parameters

Data type	Name	Text	Description
uint8	interface	I	Current host transport interface: 1 = UART (factory default)
uint8	cmd_channel	N	cmd_channel: Not implemented
uint8	spp_route	S	spp_route: Not implemented
uint32	cyspp_route	C	cyspp_route: Not implemented
uint8	BT flag	T	To control BT behavior when BLE is connected.  In default, value is 0x80: No discoverity, no connect for BT when BLE link is connected  Bit 7: control if this flag is valid 0: Not valid 1: Valid  Bit 4: Set connectabilty 0, Not connectable 1, BT Classic connectable  Bit 1~0: Set discoverability 0, Non-discoverable 1, Limited BT Classic discoverable 2, General BT Classic discoverable
uint8	BLE flag	L	To control BLE behavior when BT SPP is connected.  In default, value is 0x80: No ADV broadcast when BT SPP link is connected

Data type	Name	Text	Description
			Bit 7: control if this flag is valid 0: Not valid 1: Valid Bit 3~0: advType after BT SPP is connected 0, Stop advertising 1, Directed advertisement (high duty cycle) 2, Directed advertisement (low duty cycle) 3, Undirected advertisement (high duty cycle) 4, Undirected advertisement (low duty cycle) 5, Non-connectable advertisement (high duty cycle) 6, Non-connectable advertisement (low duty cycle) 7, Discoverable advertisement (high duty cycle) 8, Discoverable advertisement (low duty cycle)
uint8	active_time_due_puart	A	Active kept time after received data from PUART in low power state. Unit is second.
uint8	event_filter	E	Event filter 0: Not set 1: Set, command response and event will not appear in command terminal Note: Factory default = 0 (Not set)
uint8a	event_array	D	response&events which are not filtered: 12 bytes in max, so it can set 6 non-filtered response&events

#### Related commands

[system\\_set\\_transport \(ST, ID=2/23\)](#)  
[system\\_get\\_uart\\_parameters \(GTU, ID=2/26\)](#)

### 7.2.2.18 *system\_set\_uart\_parameters* (STU, ID=2/25)

Configure new UART settings for host communication.

This command configures the UART peripheral behavior used for wired external host communication when the host transport interface is set to "UART" with the *system\_set\_transport* (ST, ID=2/23) API command. If a change is successful, EZ-Serial firmware platform for Ezurio Vela IF820 series module will send the response packet using the *original* configuration, and then apply the new UART settings.

**Note:** This command affects protected settings, which means you cannot immediately apply changes to flash. To store new settings in non-volatile memory, you must send the command once without the flash storage bit/flag, and then re-send the same command again with the flash storage bit/flag set. This prevents accidental permanent communication lockout resulting from flash-stored settings that the connected host cannot use. For detail, see section *Protected configuration settings*.

#### : Binary header

	Type	Length	Group	ID	Notes
CMD	C0	0B	02	19	None.
RSP	C0	02	02	19	None.

#### Text info

Text name	Response length	Category	Notes
STU	0x0009	SET	None.

#### : Command arguments

Data type	Name	Text	Description
uint32	baud	B	UART baud rate: Minimum = 300 baud (0x12C) Factory default = 115,200 baud (0x1C200) Maximum = 3,000,000 baud (0x2DC6C0)
uint8	autobaud	A	Auto-detect UART baud rate at boot: 0 = Disabled (factory default, must always be disabled in current version)
uint8	autocorrect	C	Auto-correct UART clock to compensate for wide temperature variation: 0 = Disabled (factory default, must always be disabled in current version)
uint8	flow	F	UART RTS/CTS flow control: 0 = Disabled (factory default) 1 = Enabled
uint8	databits	D	UART data bits: 8 = 8 data bits (factory default)
uint8	parity	P	UART parity: 0 = Disabled (factory default) 1 = Odd parity 2 = Even parity
uint8	stopbits	S	UART stop bits: 1 = 1 stop bit (factory default) 2 = 2 stop bits



Data type	Name	Text	Description
uint8	uart_type	T	Uart type: 0 = puart (factory default) 1 = HCI UART (Not implemented)

**Response parameters**

None.

**Related commands**

[system\\_set\\_transport](#) (ST, ID=2/23)  
[system\\_get\\_uart\\_parameters](#) (GTU, ID=2/26)

**Example usage**

See section [Changing the serial communication parameters](#).

### 7.2.2.19 *system\_get\_uart\_parameters (GTU, ID=2/26)*

Obtain the current UART settings for host communication.

#### *: Binary header*

	Type	Length	Group	ID	Notes
CMD	C0	01	02	1A	None.
RSP	C0	0C	02	1A	None.

#### Text info

Text name	Response length	Category	Notes
GTU	0x0032	GET	None.

#### *: Command arguments*

Data type	Name	Text	Description
uint8	uart_type	T	Uart type: 0 = uart (factory default) 1 = HCI UART (Not implemented)

#### *: Response parameters*

Data type	Name	Text	Description
uint32	baud	B	UART baud rate: Minimum = 300 baud (0x12C) Factory default = 115,200 baud (0x1C200) Maximum = 3,000,000 baud (0x2DC6C0)
uint8	autobaud	A	Auto-detect UART baud rate at boot: 0 = Disabled (factory default, must always be disabled in current version)
uint8	autocorrect	C	Auto-correct UART clock to compensate for wide temperature variation: 0 = Disabled (factory default, must always be disabled in current version)
uint8	flow	F	UART RTS/CTS flow control: 0 = Disabled (factory default) 1 = Enabled
uint8	databits	D	UART data bits: 8 = 8 data bits (factory default)
uint8	parity	P	UART parity: 0 = Disabled (factory default) 1 = Odd parity 2 = Even parity
uint8	stopbits	S	UART stop bits: 1 = 1 stop bit (factory default) 2 = 2 stop bits

#### Related commands

*system\_get\_transport (GT, ID=2/24)*  
*system\_set\_uart\_parameters (STU, ID=2/25)*

### 7.2.3 GAP Group (ID=4)

GAP methods relate to the Generic Access Protocol layer of the Bluetooth® stack, which includes management of scanning and advertising, connection establishment, and connection maintenance.

Commands within the GAP group are listed below:

```
gap_connect (/C, ID=4/1)
gap_cancel_connection (/CX, ID=4/2)
gap_update_conn_parameters (/UCP, ID=4/3)
gap_disconnect (/DIS, ID=4/5)
gap_add_whitelist_entry (/WLA, ID=4/6)
gap_delete_whitelist_entry (/WLD, ID=4/7)
gap_start_adv (/A, ID=4/8)
gap_stop_adv (/AX, ID=4/9)
gap_start_scan (/S, ID=4/10)
gap_stop_scan (/SX, ID=4/11)
gap_query_peer_address (/QPA, ID=4/12)
gap_query_rssi (/QSS, ID=4/13)
gap_query_whitelist (/QWL, ID=4/14)
gap_set_device_name (SDN, ID=4/15)
gap_get_device_name (GDN, ID=4/16)
gap_set_device_appearance (SDA, ID=4/17)
gap_get_device_appearance (GDA, ID=4/18)
gap_set_adv_data (SAD, ID=4/19)
gap_get_adv_data (GAD, ID=4/20)
gap_set_sr_data (SSRD, ID=4/21)
gap_get_sr_data (GSRD, ID=4/22)
gap_set_adv_parameters (SAP, ID=4/23)
gap_get_adv_parameters (GAP, ID=4/24)
gap_set_scan_parameters (SSP, ID=4/25)
gap_get_scan_parameters (GSP, ID=4/26)
gap_set_conn_parameters (SCP, ID=4/27)
gap_get_conn_parameters (GCP, ID=4/28)
```

Events within this group are documented in section [GAP Group \(ID=4\)](#).

#### 7.2.3.1 *gap\_connect (/C, ID=4/1)*

Initiate a connection to a remote Peripheral device.

For this command to succeed, EZ-Serial firmware platform for Ezurio Vela IF820 series module must not have other ongoing BLE activity. In other words:

- The module must not be advertising. Use [gap\\_stop\\_adv \(/AX, ID=4/9\)](#) to stop, if necessary.
- The module must not be scanning. Use [gap\\_stop\\_scan \(/SX, ID=4/11\)](#) to stop, if necessary.
- The module must not be in a connection with other remote Peripheral device. Use [gap\\_disconnect \(/DIS, ID=4/5\)](#) to disconnect, if necessary.

After starting the connection process, the module will begin scanning for a connectable advertisement packet from the target device. This will continue until it succeeds, or until the connection attempt is canceled with the [gap\\_cancel\\_connection \(/CX, ID=4/2\)](#) API command, or the connection scan timeout period expires (if it has been set).

When sending this command in text mode, all omitted arguments except `address` and `type` will default to the values set using the [gap\\_set\\_conn\\_parameters \(SCP, ID=4/27\)](#) API command.

**Note:** If `scan_timeout` is set to zero, the connection attempt will persist forever until it succeeds or it is cancelled intentionally. The `supervision_timeout` parameter governs link loss detection after a connection is established, and does not affect the connection attempt itself.

#### : Binary header

	Type	Length	Group	ID	Notes
CMD	C0	13	04	01	None.

	Type	Length	Group	ID	Notes
RSP	C0	03	04	01	None.

#### Text info

Text name	Response length	Category	Notes
/C	0x000D	ACTION	None.

#### : Command arguments

Data type	Name	Text	Description
macaddr	address	A	Target connection address:
uint8	type	T	Address type: 0 = Public 1 = Random/private
uint16	interval	I	Not implemented
uint16	slave_latency	L	Not implemented
uint16	supervision_timeout	O	Not implemented
uint16	scan_interval	V	Not implemented
uint16	scan_window	W	Not implemented
uint16	scan_timeout	M	Connection scan timeout (unit is second): 0 to disable

#### : Response parameters

Data type	Name	Text	Description
uint8	conn_handle	C	Handle assigned to new pending connection

#### Related commands:

gap\_connect (/C, ID=4/1)  
gap\_disconnect (/DIS, ID=4/5)

#### Related events:

gap\_connected (C, ID=4/5) – Occurs when an outgoing connection attempt succeeds

#### Example usage:

None.

### 7.2.3.2 `gap_cancel_connection (/CX, ID=4/2)`

Cancel a pending connection attempt.

Use this command to manually end a pending connection attempt to a remote peer device which you previously initiated with the `gap_connect (/C, ID=4/1)` API command. This command takes no parameters because it is not possible to have more than one pending outgoing connection attempt at a time.

**Note:** This command applies only when ending a connection attempt that has not succeeded yet. To close an established connection, use the `gap_disconnect (/DIS, ID=4/5)` API command instead.

#### : Binary header

	Type	Length	Group	ID	Notes
CMD	C0	00	04	02	None.
RSP	C0	02	04	02	None.

#### Text info

Text name	Response length	Category	Notes
/CX	0x0009	ACTION	None.

#### Command arguments:

None.

#### Response parameters:

None.

#### Related commands:

`gap_connect (/C, ID=4/1)`  
`gap_disconnect (/DIS, ID=4/5)`

#### Related events:

`gap_connected (C, ID=4/5)`

#### Example usage:

None.

### 7.2.3.3 `gap_update_conn_parameters (/UCP, ID=4/3)`

Request a connection parameter update for an active connection.

Use this command to change the connection interval, slave latency, and supervision timeout for an active connection. If the parameter update is successful, EZ-Serial firmware platform for Ezurio Vela IF820 series module will generate the `gap_connection_updated (CU, ID=4/8)` API event after applying new parameters. This will only occur if one or more of the parameters changes from its previous value.

The behavior following this command depends on the link-layer role (master or slave) of the device which initiated the request. The master device has final authority over connection parameters. The AIROC™

If used while in the slave role (connection from peer initiated remotely):

New connection parameters must be confirmed by the master  
Local device will generate the `gap_connection_updated (CU, ID=4/8)` event if master accepts parameters

#### : Binary header

	Type	Length	Group	ID	Notes
CMD	C0	07	04	03	None.

	Type	Length	Group	ID	Notes
RSP	C0	02	04	03	None.

#### Text info

Text name	Response length	Category	Notes
/UCP	0x000A	ACTION	None.

#### : Command arguments

Data type	Name	Text	Description
uint8	conn_handle	C	Handle of connection to update (Ignored in current release)
uint16	interval	I*	Connection interval (1.25 ms units): Minimum = 0x0006 (6 * 1.25 ms = 7.5 ms) Maximum = 0x0C80 (3200 * 1.25 ms = 4 seconds)
uint16	slave_latency	L*	Slave latency (connection interval count): Minimum = 0, no intervals skipped Maximum depends on interval and supervision timeout, such that: [interval * slave_latency] < supervision_timeout
uint16	supervision_timeout	O*	Supervision timeout (10 ms units): Minimum = 0x000A (10 * 10 ms = 100 ms) Maximum = 0x0C80 (3200 * 10 ms = 32 seconds)

#### Response parameters

None.

#### Related commands

None.

#### Related events

[gap\\_connection\\_updated \(CU, ID=4/8\)](#)

### 7.2.3.4 *gap\_disconnect (/DIS, ID=4/5)*

Close an open connection to a remote device.

Use this command to cleanly close an established connection with a remote peer device. The connection must first have been fully opened, indicated by the *gap\_connected (C, ID=4/5)* API event.

#### *: Binary header*

	Type	Length	Group	ID	Notes
CMD	C0	01	04	05	None.
RSP	C0	02	04	05	None.

#### Text info

Text name	Response length	Category	Notes
/DIS	0x000A	ACTION	None.

#### *: Command arguments*

Data type	Name	Text	Description
uint8	conn_handle	C	Handle of connection to disconnect

#### Response parameters

None.

#### Related commands

None.

#### Related events

*gap\_disconnected (DIS, ID=4/6)*

### 7.2.3.5 *gap\_add\_whitelist\_entry (/WLA, ID=4/6)*

Add a new Bluetooth® address to the whitelist.

The whitelist is an optional filter for determining which remote peers are allowed to connect, or which the local module may try to connect. When whitelist filtering is active, devices that are not on the whitelist are not allowed to connect with the module. You can control whitelist filter usage during advertising, scanning, or outgoing connect attempts.

The whitelist is an optional filter: it will determine which remote peers are allowed to connect the local module. On the other hand, the local module may try to connect the device in the whitelist. When whitelist filtering is active, devices that are not on the whitelist are not allowed to connect with the module. You can control whitelist filter usage during advertising, scanning, or outgoing connect attempts.

**Note:** You can only use this command while disconnected. Changes to the whitelist are not allowed during a connection.

Each whitelist entry is made up of two parts: the peer's Bluetooth® address and the type of address (public or private). You must specify the correct address type for each peer based on the type of address being used. This information is available in scan results and connection details.

**Note:** The BLE stack in EZ-Serial firmware platform for Ezurio Vela IF820 series module automatically mirrors the bonded device list into the whitelist. This behavior accommodates the most common use case for the whitelist, and you may not need any manual additions or removals from the whitelist.

#### : Binary header

	Type	Length	Group	ID	Notes
CMD	C0	07	04	06	None.
RSP	C0	03	04	06	None.

#### Text info

Text name	Response length	Category	Notes
/WLA	0x000F	ACTION	None.

#### : Command arguments

Data type	Name	Text	Description
macaddr	address	A*	Bluetooth® address
uint8	type	T	Address type: 0 = Public (default) 1 = Random/private

#### : Response parameters

Data type	Name	Text	Description
uint8	count	C	Updated whitelist entry count

#### Command-specific result codes

None.

#### Related commands

[gap\\_delete\\_whitelist\\_entry \(/WLD, ID=4/7\)](#)  
[gap\\_query\\_peer\\_address \(/QPA, ID=4/12\)](#)  
[gap\\_set\\_adv\\_parameters \(SAP, ID=4/23\)](#) – Configure whitelist filter for advertising

#### Related events

None.



### 7.2.3.6 *gap\_delete\_whitelist\_entry* (/WLD, ID=4/7)

Remove a Bluetooth® address from the whitelist.

Use this command to remove a specific device that exists on the whitelist. For details on whitelist behavior, see the documentation for the *gap\_add\_whitelist\_entry* (/WLA, ID=4/6) API command.

#### : Binary header

	Type	Length	Group	ID	Notes
CMD	C0	07	04	07	None.
RSP	C0	03	04	07	None.

#### Text info

Text name	Response length	Category	Notes
/WLD	0x000F	ACTION	None.

#### : Command arguments

Data type	Name	Text	Description
macaddr	address	A	Bluetooth® address
uint8	type	T	Address type: 0 = Public (default) 1 = Random/private

#### : Response parameters

Data type	Name	Text	Description
uint8	count	C	Updated whitelist entry count

#### Related commands

*gap\_add\_whitelist\_entry* (/WLA, ID=4/6)

### 7.2.3.7 *gap\_start\_adv* (/A, ID=4/8)

Start advertising.

This command begins advertising using the specified parameters or using the pre-configured default advertising parameters if in text mode and some arguments are omitted. EZ-Serial firmware platform for Ezurio Vela IF820 series module must not already be advertising for this command to succeed. However, it is possible to advertise and scan simultaneously.

EZ-Serial firmware platform for Ezurio Vela IF820 series module will generate the *gap\_adv\_state\_changed* (ASC, ID=4/2) API event when the advertising state changes.

**Note:** You can start advertising while connected only if you specify "0" (broadcast-only) for the mode argument. The BLE stack does not support being connected and connectable at the same time.

**Note:** When using the "scannable, undirected" type or "non-connectable, undirected" setting for the type argument, the advertisement interval must be 100 ms (0xA0) or greater, per the Bluetooth® specification. Shorter intervals than this will result in an error response.

### : Binary header

	Type	Length	Group	ID	Notes
CMD	C0	13	04	08	None.
RSP	C0	02	04	08	None.

### Text info

Text name	Response length	Category	Notes
/A	0x0008	ACTION	None.

### : Command arguments

Data type	Name	Text	Description
UInt08	Mode	M	Discovery mode: Not implemented.
uint8	type	T	Advertisement type: 0 = Stop advertising 1 = Directed advertisement (high duty cycle) 2 = Directed advertisement (low duty cycle) 3 = Undirected advertisement (high duty cycle) 4 = Undirected advertisement (low duty cycle) 5 = Non-connectable advertisement (high duty cycle) 6 = Non-connectable advertisement (low duty cycle) 7 = discoverable advertisement (high duty cycle) 8 = discoverable advertisement (low duty cycle) <hr/> <b>Note:</b> If you set high duty (T=3,5,7), FW will auto switch to low duty(T=4,6,8) respectively after high duration is elapsed
uint8	channels	C	Advertisement channel selection bitmask (at least one bit must be set): Bit 0 (0x1) = Channel 37 Bit 1 (0x2) = Channel 38 Bit 2 (0x4) = Channel 39
uint16	high interval	H	high_duty_interval: (625 $\mu$ s units): Minimum = 0x0020 (32 * 0.625 ms = 20 ms) Maximum = 0x4000 (16384 * 0.625 ms = 10.24 seconds)
uint16	high duration	D	high duty duration in seconds (0 for infinite)
uint16	low interval	L	low_duty_interval: (625 $\mu$ s): Minimum = 0x0020 (32 * 0.625 ms = 20 ms) Maximum = 0x4000 (16384 * 0.625 ms = 10.24 seconds)
uint16	low duration	O	low duty duration in seconds (0 for infinite)
uint8	flag	F	Advertisement interval Advertisement behavior flags bitmask: Bit 0 (0x1) = Enable automatic advertising mode upon boot/disconnection Bit 1 (0x2) = Use custom advertisement and scan response data

Data type	Name	Text	Description
			<b>Note:</b> Factory default = 0x00 (no bits set)
uint8	directAddr	A	Directed advertisement address
uint8	directAddrType	Y	Directed Address type (if using directed advertisement mode): 0: BLE_ADDR_PUBLIC 1: BLE_ADDR_RANDOM
			<b>Note:</b> Current implementation does not care address type

**Response parameters**

None.

**Related commands**

[gap\\_stop\\_adv \(/AX, ID=4/9\)](#)  
[gap\\_set\\_adv\\_data \(SAD, ID=4/19\)](#)  
[gap\\_set\\_sr\\_data \(SSRD, ID=4/21\)](#)  
[gap\\_set\\_adv\\_parameters \(SAP, ID=4/23\)](#)

**Related events**

[gap\\_adv\\_state\\_changed \(ASC, ID=4/2\)](#)

**Example usage**See [Advertising as peripheral device](#)**7.2.3.8 [gap\\_stop\\_adv \(/AX, ID=4/9\)](#)**

Stop advertising.

This command immediately stops advertising if it is currently active. Note that advertising may have started because of the [gap\\_start\\_adv \(/A, ID=4/8\)](#) API command, or due to specific configuration settings (GAP parameters, CYSPP profile) that automatically begin advertising.

EZ-Serial firmware platform for Ezurio Vela IF820 series module will generate the [gap\\_adv\\_state\\_changed \(ASC, ID=4/2\)](#) API event when the advertising state changes.

**: Binary header**

	Type	Length	Group	ID	Notes
CMD	C0	00	04	09	None.
RSP	C0	02	04	09	None.

**Text info**

Text name	Response length	Category	Notes
/AX	0x0009	ACTION	None.

**Command arguments**

None.

**Response parameters**

None.

**Related commands**

gap\_start\_adv (/A, ID=4/8)

#### Related events

gap\_adv\_state\_changed (ASC, ID=4/2)

#### 7.2.3.9 gap\_start\_scan (/S, ID=4/10)

Start scanning. This command will use the configured default scan parameters, unless specified otherwise using arguments.

#### : Binary header

	Type	Length	Group	ID	Notes
CMD	C0	0A	04	0A	None.
RSP	C0	02	04	0A	None.

#### Text info

Text name	Response length	Notes
/S	0x0008	None.

#### : Command arguments

Data type	Name	Text	Description
uint8	mode	M	Discovery mode: 0 = Not scan 1 = High duty cycle scan 2 = Low duty cycle scan
uint16	interval	I	Scan interval (625 $\mu$ s units): Minimum = 0x0004 (4 * 0.625 ms = 2.5 ms) Maximum = 0x4000 (16384 * 0.625 ms = 10.24 seconds) Factory default = 0x0100 (256 * 0.625 ms = 160 ms)
uint16	window	W	Scan window (625 $\mu$ s units): Minimum = 0x0004 (4 * 0.625 ms = 2.5 ms) Maximum = 0x4000 (16384 * 0.625 ms = 10.24 seconds) Factory default = 0x0100 (256 * 0.625 ms = 160 ms) Cannot be greater than interval
uint8	active	A	Active scanning: 0 = Passive scanning 1 = Active scanning
uint8	filter	F	Whitelist filter policy: Not implemented and always 0
uint8	nodupe	D	Duplicate filter policy: 0 = Disable duplicate result filtering 1 = Enable duplicate result filtering
uint16	timeout	O	Scan timeout (seconds): Maximum = 255 0 to disable

#### Response parameters

None.

#### Related commands:

gap\_stop\_scan (/SX, ID=4/11)

#### Related events:

None.

### 7.2.3.10 *gap\_stop\_scan (/SX, ID=4/11)*

Stop scanning.

#### : Binary header

	Type	Length	Group	ID	Notes
CMD	C0	00	04	0B	None.
RSP	C0	02	04	0B	None.

#### Text info

Text name	Response length	Notes
/SX	0x0009	None.

#### Command arguments:

None.

#### Response parameters:

None.

#### Related commands:

*gap.start\_scan*

#### Related events:

None.

### 7.2.3.11 *gap\_query\_peer\_address (/QPA, ID=4/12)*

Query remote peer Bluetooth® address.

This command returns the Bluetooth® address of the currently connected remote peer device. An active connection is required to use this command successfully.

#### : Binary header

	Type	Length	Group	ID	Notes
CMD	C0	01	04	0C	None.
RSP	C0	09	04	0C	None.

#### Text info

Text name	Response length	Notes
/S	0x0008	None.

#### : Command arguments

Data type	Name	Text	Description
uint8	conn_handle	C	Handle of connection for which to query remote peer address (Ignored in current release)

#### Response parameters

Data type	Name	Text	Description
macaddr	address	A	Peer Bluetooth® address

Data type	Name	Text	Description
uint8	address_type	T	Address type

**Related commands**

`gap_query_rssi (/QSS, ID=4/13)`

### 7.2.3.12 `gap_query_rssi (/QSS, ID=4/13)`

This command returns the remote signal strength indication (RSSI) value detected in the packet received most recently from the currently connected remote peer device. An active connection is required to use this command successfully.

**Note:** RSSI values in real-world environments often fall in the -50 dBm to -70 dBm range. An RSSI value at this level does not necessarily indicate a poor connection.

The RSSI value returned in the response is expressed as a signed 8-bit integer. In text mode, it will appear in two's complement form. Positive numbers in this form fall in the range [0, 127] and are as they appear. Negative numbers fall in the range [128, 255] and should have 256 subtracted from the value to obtain the real value.

Examples:

0x03 = +3 dBm

0xFF = -1 dBm (0xFF = 255 - 256 = -1)

0xF0 = -16 dBm (0xF0 = 240 - 256 = -16)

0xC5 = -59 dBm (0xC5 = 197 - 256 = -59)

**: Binary header**

	Type	Length	Group	ID	Notes
CMD	C0	01	04	0D	None.
RSP	C0	03	04	0D	None.

**Text info**

Text name	Response length	Notes
/QSS	0x000F	None.

**: Command arguments**

Data type	Name	Text	Description
uint8	conn_handle	C	Handle of connection for which to query signal strength

**Response parameters**

Data type	Name	Text	Description
int8	rssi	R	RSSI value in dBm (between -85 and +5), or 0 if used while not connected

**Related commands**

`gap_query_peer_address (/QPA, ID=4/12)`

**7.2.3.13** *gap\_query\_whitelist (/QWL, ID=4/14)*

Request a list of whitelisted devices.

This command provides access to the current whitelist. The response from this command includes the number of devices on the whitelist, and the response is followed by the *gap\_whitelist\_entry* (WL, ID=4/1) API events which provide details for each entry.

**: Binary header**

	Type	Length	Group	ID	Notes
CMD	C0	00	04	0E	None.
RSP	C0	03	04	0E	None.

**Text info**

Text name	Response length	Category	Notes
/QWL	0x000F	ACTION	None.

**: Command arguments**

None.

**Response parameters**

Data type	Name	Text	Description
uint8	Count	C	Whitelist entry count

**Related commands**

*gap\_add\_whitelist\_entry* (/WLA, ID=4/6)

*gap\_delete\_whitelist\_entry* (/WLD, ID=4/7)

**Related events**

*gap\_whitelist\_entry* (WL, ID=4/1)

**7.2.3.14** *gap\_set\_device\_name (SDN, ID=4/15)*

Configure a new device name.

This is typically a UTF-8 string value that is stored in the Device Name characteristic (UUID 0x2A00) in the local GATT structure. This characteristic is part of the GAP service (UUID 0x1800). The GAP service is mandatory for all Bluetooth® Smart devices, and the Device Name characteristic is a mandatory part of the GAP service.

Using this command affects the value in the local GATT Server Device Name characteristic, and the local name field in the automatically managed scan response packed used for advertising.

**: Binary header**

	Type	Length	Group	ID	Notes
CMD	C0	02–42	04	0F	Variable-length command payload, minimum of 2 (0x02), maximum of 66 (0x42)
RSP	C0	02	04	0F	None.

**Text info**

Text name	Response length	Category	Notes
SDN	0x0009	SET	None.

#### : Command arguments

Data type	Name	Text	Description
uint8	Type	T	Device Type: 0 = BLE Device Name 1 = BT classic Device Name
string	name	N	New device name (0-64 bytes, raw ASCII data when in text mode)

#### Response parameters

None.

#### Related commands

[gap\\_get\\_device\\_name \(GDN, ID=4/16\)](#)

#### Example usage

See section [Changing device name and appearance](#)

#### 7.2.3.15 [gap\\_get\\_device\\_name \(GDN, ID=4/16\)](#)

Obtain the current device name.

#### : Binary header

	Type	Length	Group	ID	Notes
CMD	C0	00	04	10	None.
RSP	C0	03-43	04	10	Variable-length response payload, minimum of 3 (0x03), maximum of 67 (0x43)

#### Text info

Text name	Response length	Category	Notes
GDN	0x000C-0x004C	GET	Variable-length response payload, minimum of 12 (0x0C), maximum of 76 (0x4C)

#### : Command arguments

Data type	Name	Text	Description
uint8	Type	T	Device Type: 0 = BLE Device Name 1 = BT classic Device Name

#### : Response parameters

Data type	Name	Text	Description
string	name	N	Current device name (0-64 bytes, raw ASCII data when in text mode)

#### Related Commands

[gap\\_set\\_device\\_name \(SDN, ID=4/15\)](#)

#### 7.2.3.16 [gap\\_set\\_device\\_appearance \(SDA, ID=4/17\)](#)

Configure a new device name.

Define the device appearance value. This is a 16-bit value which is stored in the Appearance characteristic (UUID 0x2A01) in the local GATT structure. This characteristic is part of the GAP service (UUID 0x1800). The GAP service is mandatory for every Bluetooth® Smart device, and the Appearance characteristic is a mandatory part of the GAP service.

Using this command affects the value in the local GATT Server Device Appearance characteristic.



*: Binary header*

	Type	Length	Group	ID	Notes
CMD	C0	02	04	11	None.
RSP	C0	02	04	11	None.

## Text info

Text name	Response length	Category	Notes
SDA	0x0009	SET	None.

*: Command arguments*

Data type	Name	Text	Description
uint16	appearance	A	New device appearance value (factory default is 0x0000)

## Response parameters

None.

## Related commands

*gap\_get\_device\_appearance (GDA, ID=4/18)**7.2.3.17 gap\_get\_device\_appearance (GDA, ID=4/18)*

Obtain the current device appearance value.

*: Binary header*

	Type	Length	Group	ID	Notes
CMD	C0	00	04	12	None.
RSP	C0	04	04	12	None.

## Text info

Text name	Response length	Category	Notes
GDA	0x0010	GET	None.

*: Command arguments*

None.

## Response parameters

Data type	Name	Text	Description
uint16	appearance	A	Current device appearance value

## Related commands

*gap\_set\_device\_appearance (SDA, ID=4/17)**7.2.3.18 gap\_set\_adv\_data (SAD, ID=4/19)*

Configure new custom advertisement packet data.

Define a new byte sequence for the primary advertisement packet data payload. This content is visible to all scanning devices performing a passive or active scan when the Ezurio Vela IF820 series module is in an advertising state.

**Note:** EZ-Serial firmware platform for Ezurio Vela IF820 series module automatically manages advertisement content unless you enable the use of user-defined data with the `gap_set_adv_parameters` (SAP, ID=4/23) API command. If you only set custom data but do not enable user-defined content, the data here remains unused.

*: Binary header*

	Type	Length	Group	ID	Notes
CMD	C0	01–20	04	13	Variable-length command payload, minimum of 1 (0x01), maximum of 32 (0x20)
RSP	C0	02	04	13	None.

**Text info**

Text name	Response length	Category	Notes
SAD	0x0009	SET	None.

*: Command arguments*

Data type	Name	Text	Description
uint8a	data	D	New advertisement payload data (0-31 bytes)

*Note:* uint8a data type requires one prefixed "length" byte before binary parameter payload

**Response parameters**

None.

**Related commands**

`gap_start_adv` (/A, ID=4/8)  
`gap_get_adv_data` (GAD, ID=4/20)  
`gap_set_sr_data` (SSRD, ID=4/21)  
`gap_set_adv_parameters` (SAP, ID=4/23)

**Example usage**

See [section Customizing advertisement and scanning response data](#)

*7.2.3.19 gap\_get\_adv\_data (GAD, ID=4/20)*

Obtain the current custom advertisement packet data.

*: Binary header*

	Type	Length	Group	ID	Notes
CMD	C0	00	04	14	None.
RSP	C0	03–22	04	14	Variable-length response payload, minimum of 3 (0x03), maximum of 34 (0x22)

**Text info**

Text name	Response length	Category	Notes
GAD	0x000D–0x004B	GET	Variable-length response payload, minimum of 13 (0x0D), maximum of 75 (0x4B)

#### : Command arguments

None.

#### Response parameters

Data type	Name	Text	Description
uint8a	data	D	Current advertisement payload data (0-31 bytes)

*Note:* uint8a data type requires one prefixed "length" byte before binary parameter payload

#### Related commands

[gap\\_set\\_adv\\_data](#) (SAD, ID=4/19)

#### 7.2.3.20 [gap\\_set\\_sr\\_data](#) (SSRD, ID=4/21)

Configure new custom scan response packet payload.

This command defines a new byte sequence for the scan response packet. This content is visible to all scanning devices performing an active scan when the Ezurio Vela IF820 series module is in a scannable advertising state.

**Note:** EZ-Serial firmware platform for Ezurio Vela IF820 series module automatically manages scan response content unless you enable the use of user-defined data with the [gap\\_set\\_adv\\_parameters](#) (SAP, ID=4/23) API command. If you only set custom data but do not enable user-defined content, the data in [gap\\_set\\_sr\\_data](#) will remain unused.

#### : Binary header

	Type	Length	Group	ID	Notes
CMD	C0	01-20	04	15	Variable-length command payload, minimum of 1 (0x01), maximum of 32 (0x20)
RSP	C0	02	04	15	None.

#### Text info

Text name	Response length	Category	Notes
SSRD	0x000A	SET	None.

#### : Command arguments

Data type	Name	Text	Description
uint8a	data	D	New scan response payload data (0-31 bytes)

**Note:** uint8a data type requires one prefixed "length" byte before binary parameter payload

#### Response parameters

None.

#### Related commands

[gap\\_start\\_adv](#) (/A, ID=4/8)

[gap\\_set\\_adv\\_data](#) (SAD, ID=4/19)

[gap\\_get\\_sr\\_data](#) (GSRD, ID=4/22)

[gap\\_set\\_adv\\_parameters](#) (SAP, ID=4/23)

#### Example usage

See section [Customizing advertisement and scanning response data](#)

### 7.2.3.21 *gap\_get\_sr\_data* (GSRD, ID=4/22)

Obtain the current custom scan response packet data.

#### : Binary header

	Type	Length	Group	ID	Notes
CMD	C0	00	04	16	None.
RSP	C0	03–22	04	16	Variable-length response payload, minimum of 3 (0x03), maximum of 34 (0x22)

#### Text info

Text name	Response length	Category	Notes
GSRD	0x000D–0x004B	GET	Variable-length response payload, minimum of 13 (0xD), maximum of 75 (0x4B)

#### : Command arguments

None.

#### Response parameters

Data type	Name	Text	Description
uint8a	data	D	Current scan response payload data (0-31 bytes)
<b>Note:</b>			uint8a data type requires one prefixed "length" byte before binary parameter payload

#### Related commands

*gap\_set\_sr\_data* (SSRD, ID=4/21)

### 7.2.3.22 *gap\_set\_adv\_parameters* (SAP, ID=4/23)

Configure new default advertisement parameters.

These parameters are used when sending the *gap\_start\_adv* (IA, ID=4/8) API command in text mode without specifying non-default arguments.

**Note:** Setting Bit 0 (0x01) of the flags value using this command enables automatic advertisement on boot, as described. However, advertisements may automatically start even if this bit is cleared if the enable setting of CYSPP is set to the "enable + autostart" setting. Factory default settings include this value for the CYSPP feature.

#### : Binary header

	Type	Length	Group	ID	Notes
CMD	C0	13	04	17	None.
RSP	C0	02	04	17	None.

#### Text info

Text name	Response length	Category	Notes
SAP	0x0009	SET	None.

### : Command arguments

Data type	Name	Text	Description
uint8	mode	M	Discovery mode: Not implemented
uint8	type	T	Advertisement type: 0 = Stop advertising 1 = Directed advertisement (high duty cycle) 2 = Directed advertisement (low duty cycle) 3 = Undirected advertisement (high duty cycle) 4 = Undirected advertisement (low duty cycle) 5 = Non-connectable advertisement (high duty cycle) 6 = Non-connectable advertisement (low duty cycle) 7 = discoverable advertisement (high duty cycle) 8 = discoverable advertisement (low duty cycle) <hr/> <b>Note:</b> If you set high duty (T=3,5,7), FW will auto switch to low duty (T=4,6,8) respectively after high duration is elapsed
uint8	channels	C	Advertisement channel selection bitmask: Bit 0 (0x1) = Channel 37 Bit 1 (0x2) = Channel 38 Bit 2 (0x4) = Channel 39 <hr/> <b>Note:</b> At least one bit must be set, factory default is all <b>0x07</b> (all bits set)
uint16	high interval	H	high_duty_interval: (625 $\mu$ s units): Minimum = 0x0020 (32 * 0.625 ms = 20 ms) Maximum = 0x4000 (16384 * 0.625 ms = 10.24 seconds)
uint16	high duration	D	High duty duration in seconds (0 for infinite)
uint16	low interval	L	low_duty_interval: (625 $\mu$ s units): Minimum = 0x0020 (32 * 0.625 ms = 20 ms) Maximum = 0x4000 (16384 * 0.625 ms = 10.24 seconds)
uint16	low duration	O	Low duty duration in seconds (0 for infinite)
uint8	flags	F	Advertisement behavior flags bitmask: Bit 0 (0x1) = Enable automatic advertising mode upon boot/disconnection Bit 1 (0x2) = Use custom advertisement and scan response data <hr/> <b>Note:</b> Factory default = 0x00 (no bits set)
macaddr	directAddr	A	Directed advertisement address
uint8	directAddrType	Y	Directed address type (if using directed advertisement mode): 0: BLE_ADDR_PUBLIC 1: BLE_ADDR_RANDOM

### Response parameters

None.

### Related commands

<https://www.ezurio.com/>

gap\_start\_adv (IA, ID=4/8)

gap\_get\_adv\_parameters (GAP, ID=4/24)

### 7.2.3.23 gap\_get\_adv\_parameters (GAP, ID=4/24)

Obtain the current advertisement parameters.

#### : Binary header

	Type	Length	Group	ID	Notes
CMD	C0	00	04	18	None.
RSP	C0	15	04	18	None.

#### Text info

Text name	Response length	Category	Notes
GAP	0x004D	GET	None.

#### : Command arguments

None.

#### Response parameters

Data type	Name	Text	Description
uint8	Mode	M	Discovery mode: Not implemented.
uint8	Type	T	Advertisement type: 0 = Stop advertising 1 = Directed advertisement (high duty cycle) 2 = Directed advertisement (low duty cycle) 3 = Undirected advertisement (high duty cycle) 4 = Undirected advertisement (low duty cycle) 5 = Non-connectable advertisement (high duty cycle) 6 = Non-connectable advertisement (low duty cycle) 7 = discoverable advertisement (high duty cycle) 8 = discoverable advertisement (low duty cycle)
uint8	channels	C	Advertisement channel selection bitmask: Bit 0 (0x1) = Channel 37 Bit 1 (0x2) = Channel 38 Bit 2 (0x4) = Channel 39  <b>Note:</b> At least one bit must be set, factory default is all 0x07 (all bits set)
uint16	high interval	H	high_duty_interval: (625 $\mu$ s units): Minimum = 0x0020 (32 * 0.625 ms = 20 ms) Maximum = 0x4000 (16384 * 0.625 ms = 10.24 seconds)
uint16	high duration	D	high duty duration in seconds (0 for infinite)
uint16	low interval	L	low_duty_interval: (625 $\mu$ s units): Minimum = 0x0020 (32 * 0.625 ms = 20 ms) Maximum = 0x4000 (16384 * 0.625 ms = 10.24 seconds)

Data type	Name	Text	Description
uint16	low duration	O	low duty duration in seconds (0 for infinite)
uint8	flags	F	Advertisement behavior flags bitmask: Bit 0 (0x1) = Enable automatic advertising mode upon boot/disconnection Bit 1 (0x2) = Use custom advertisement and scan response data  <b>Note:</b> Factory default = 0x00 (no bits set)
macaddr	directAddr	A	Directed advertisement address
uint8	directAddrType	Y	Directed address type (if using directed advertisement mode): 0: BLE_ADDR_PUBLIC 1: BLE_ADDR_RANDOM

### Related commands

[gap\\_set\\_adv\\_parameters \(SAP, ID=4/23\)](#)

#### 7.2.3.24 [gap\\_set\\_scan\\_parameters \(SSP, ID=4/25\)](#)

Configure new default scan parameters.

These parameters will be used when sending the [gap\\_start\\_scan \(/S, ID=4/10\)](#) API command in text mode without specifying non-default arguments.

#### : Binary header

	Type	Length	Group	ID	Notes
CMD	C0	0A	04	19	None.
RSP	C0	02	04	19	None.

#### Text info

Text name	Response length	Category	Notes
SSP	0x0009	SET	None.

#### : Command arguments

Data type	Name	Text	Description
uint8	mode	M	Discovery mode: not implemented.
uint16	interval	I	Scan interval (625 $\mu$ s units): Minimum = 0x0004 (4 * 0.625 ms = 2.5 ms) Maximum = 0x4000 (16384 * 0.625 ms = 10.24 seconds) Factory default = 0x0100 (256 * 0.625 ms = 160 ms)
uint16	window	W	Scan window (625 $\mu$ s units): Minimum = 0x0004 (4 * 0.625 ms = 2.5 ms) Maximum = 0x4000 (16384 * 0.625 ms = 10.24 seconds) Factory default = 0x0100 (256 * 0.625 ms = 160 ms) Cannot be greater than interval.
uint8	active	A	Active scanning: 0 = Passive scanning 1 = Active scanning
uint8	filter	F	Whitelist filter policy: not implemented. 0 = Accept all advertising packets 1 = Accept only from whitelisted devices

Data type	Name	Text	Description
			2 = Accept only from devices sending directed advertisements to this device 3 = Accept only from whitelisted devices sending directed advertisements to this device
uint8	nodupe	D	Duplicate filter policy: 0 = Disable duplicate result filtering 1 = Enable duplicate result filtering
uint16	timeout	O	Scan timeout (seconds): Maximum = 255 0 to disable

#### Response parameters:

None.

#### Related commands:

`gap_start_scan` (/S, ID=4/10)

`gap_get_scan_parameters` (GSP, ID=4/26)

#### 7.2.3.25 `gap_get_scan_parameters` (GSP, ID=4/26)

Obtain the current scan parameters.

#### : Binary header

	Type	Length	Group	ID	Notes
CMD	C0	00	04	1A	None.
RSP	C0	0C	04	1A	None.

#### Text info

Text name	Response length	Category	Notes
GSP	0x0032	GET	None.

#### : Command arguments

None.

#### Response parameters

Data type	Name	Text	Description
uint8	mode	M	Discovery mode: not implemented.
uint16	interval	I	Scan interval (625 $\mu$ s units): Minimum = 0x0004 (4 * 0.625 ms = 2.5 ms) Maximum = 0x4000 (16384 * 0.625 ms = 10.24 seconds) Factory default = 0x0100 (256 * 0.625 ms = 160 ms)
uint16	window	W	Scan window (625 $\mu$ s units): Minimum = 0x0004 (4 * 0.625 ms = 2.5 ms) Maximum = 0x4000 (16384 * 0.625 ms = 10.24 seconds) Factory default = 0x0100 (256 * 0.625 ms = 160 ms) Cannot be greater than <code>interval</code>
uint8	active	A	Active scanning: 0 = Passive scanning (factory default) 1 = Active scanning
uint8	filter	F	Whitelist filter policy: 0 = Accept all advertising packets (factory default) 1 = Accept only from whitelisted devices 2 = Accept only from devices sending directed advertisements to this device



Data type	Name	Text	Description
			3 = Accept only from whitelisted devices sending directed advertisements to this device
uint8	nodupe	D	Duplicate filter policy: 0 = Disable duplicate result filtering (factory default) 1 = Enable duplicate result filtering
uint16	timeout	O	Scan timeout (seconds): 0 to disable (factory default)

#### Related commands

`gap_set_scan_parameters` (SSP, ID=4/25)

#### 7.2.3.26 `gap_set_conn_parameters` (SCP, ID=4/27)

Configure new default connection parameters.

These parameters will be used when sending the `gap_connect` (IC, ID=4/1) API command in text mode without specifying non-default arguments.

#### : Binary header

	Type	Length	Group	ID	Notes
CMD	C0	0C	04	1B	None.
RSP	C0	02	04	1B	None.

#### Text info

Text name	Response length	Category	Notes
SCP	0x0009	SET	None.

#### : Command arguments

Data type	Name	Text	Description
uint16	interval	I	Connection interval (1.25 ms units): Minimum = 0x0006 (6 * 1.25 ms = 7.5 ms, factory default) Maximum = 0x0C80 (3200 * 1.25 ms = 4 seconds)
uint16	slave_latency	L	Slave latency (connection interval count): Minimum = 0, no intervals skipped (factory default) Maximum depends on interval and supervision timeout, such that: <code>[interval * slave_latency] &lt; supervision_timeout</code>
uint16	supervision_timeout	O	Supervision timeout (10 ms units): Minimum = 0x000A (10 * 10 ms = 100 ms) Maximum = 0x01F4 (500 * 10 ms = 5 seconds) Factory default = 0x064 (100 * 10 ms = 1 second)
uint16	scan_interval	V	Connection scan interval (625 µs units): Minimum = 0x0004 (4 * 0.625 ms = 2.5 ms) Maximum = 0x4000 (16384 * 0.625 ms = 10.24 seconds) Factory default = 0x0100 (256 * 0.625 ms = 160 ms)
uint16	scan_window	W	Connection scan window (625 µs units): Minimum = 0x0004 (4 * 0.625 ms = 2.5 ms) Maximum = 0x4000 (16384 * 0.625 ms = 10.24 seconds) Factory default = 0x0100 (256 * 0.625 ms = 160 ms) Cannot be greater than <code>scan_interval</code>
uint16	scan_timeout	M	Connection scan timeout (seconds): 0 to disable (factory default)

#### Response parameters:

None.

**Related commands:**

gap\_connect (/C, ID=4/1)  
gap\_update\_conn\_parameters (/UCP, ID=4/3)  
gap\_get\_conn\_parameters (GCP, ID=4/28)

**7.2.3.27 gap\_get\_conn\_parameters (GCP, ID=4/28)**

Get the current default connection parameters.

**: Binary header**

	Type	Length	Group	ID	Notes
CMD	C0	00	04	1C	None.
RSP	C0	0E	04	1C	None.

**Text info**

Text name	Response length	Category	Notes
GCP	0x0033	GET	None.

**: Command arguments**

None.

**: Response parameters**

Data type	Name	Text	Description
uint16	interval	I	Connection interval (1.25 ms units): Minimum = 0x0006 (6 * 1.25 ms = 7.5 ms, factory default) Maximum = 0x0C80 (3200 * 1.25 ms = 4 seconds)
uint16	slave_latency	L	Slave latency (connection interval count): Minimum = 0, no intervals skipped (factory default) Maximum depends on interval and supervision timeout, such that: [interval * slave_latency] < supervision_timeout
uint16	supervision_timeout	O	Supervision timeout (10 ms units): Minimum = 0x000A (10 * 10 ms = 100 ms) Maximum = 0x01F4 (500 * 10 ms = 5 seconds) Factory default = 0x064 (100 * 10 ms = 1 second)
uint16	scan_interval	V	Connection scan interval (625 µs units): Minimum = 0x0004 (4 * 0.625 ms = 2.5 ms) Maximum = 0x4000 (16384 * 0.625 ms = 10.24 seconds) Factory default = 0x0100 (256 * 0.625 ms = 160 ms)
uint16	scan_window	W	Connection scan window (625 µs units): Minimum = 0x0004 (4 * 0.625 ms = 2.5 ms) Maximum = 0x4000 (16384 * 0.625 ms = 10.24 seconds) Factory default = 0x0100 (256 * 0.625 ms = 160 ms) Cannot be greater than scan_interval
uint16	scan_timeout	M	Connection scan timeout (seconds): 0 to disable (factory default)

**Related commands:**

gap\_set\_conn\_parameters (SCP, ID=4/27)

#### 7.2.4 GATT Server Group (ID=5)

GATT Server methods relate to the Server role of the Generic Attribute Protocol layer of the Bluetooth® stack. These methods are used for working with the local GATT structure.

Commands within this group are listed below:

```

gatts_create_attr (/CAC, ID=5/1)
gatts_delete_attr (/CAD, ID=5/2)
gatts_validate_db (/VGDB, ID=5/3)
gatts_store_db (/SGDB, ID=5/4)
gatts_dump_db (/DGDB, ID=5/5)
gatts_discover_services (/DLS, ID=5/6)
gatts_discover_characteristics (/DLC, ID=5/7)
gatts_discover_descriptors (/DLD, ID=5/8)
gatts_read_handle (/RLH, ID=5/9)
gatts_write_handle (/WLH, ID=5/10)
gatts_notify_handle (/NH, ID=5/11)
gatts_indicate_handle (/IH, ID=5/12)
gatts_set_parameters (SGSP, ID=5/14)
gatts_get_parameters (GGSP, ID=5/15)

```

Events within this group are documented in section [GATT Server Group \(ID=5\)](#).

##### 7.2.4.1 *gatts\_create\_attr (/CAC, ID=5/1)*

Add a new custom attribute to the local GATT structure.

The new attribute is given the next available handle. All handles are assigned sequentially. Attributes must be added in order, and are always appended to the next available position in the GATT structure.

New attributes must be entered such that the database always has a valid structure, other than possibly being incomplete while adding other required attributes. EZ-Serial firmware platform for Ezurio Vela IF820 series module rejects new attribute creation attempts that would result in an invalid structure and provide a validity report code from the list in section [EZ-Serial firmware platform for Vela IF820 GATT database validation error codes](#).

See sections [Defining custom local GATT services and characteristics](#) and [Adopted Bluetooth SIG GATT profile structure snippets](#) for detailed instructions and example usage, including important guidelines for permission settings.

---

**Note:** Always configure structural declarations (types 0x2800 and 0x2803) to have unrestricted read permissions (0x01) and no write permissions (0x00) to ensure that clients can properly discover the basic GATT database structure. Special security requirements should only be applied to characteristic value attributes or, in limited cases, related configuration descriptors.

---

Use the [gatts\\_dump\\_db \(/DGDB, ID=5/5\)](#) API command to list the current local GATT database entries in a format similar to what this command requires.

---

**Note:** EZ-Serial firmware platform for Ezurio Vela IF820 series module includes a fixed set of attributes as part of the core functionality, which cannot be deleted or modified. These attributes occupy the handle range from 1 (0x0001) to 21 (0x0015). Therefore, the first custom attribute created in a factory default state receives the handle value 22 (0x0016).

---



---

**Note:** Additions to and removals from the GATT structure are always stored in flash. As long as the "result" value in the response indicates success, the change is effective immediately and persist through power cycles and resets. The internal CPU is occupied for approximately 15 ms during each flash write operation; during this time, no other activity is processed (UART or BLE communication). Any UART data sent during this brief window is lost. Therefore, you should modify the GATT structure only while disconnected, and you should allow a gap of at least 20 ms between the end of one API command and the beginning of a new one. If you have enabled hardware flow control using the [system\\_set\\_uart\\_parameters \(STU, ID=2/25\)](#) API command, EZ-Serial firmware platform for Ezurio Vela IF820 series module blocks incoming data flow during flash writes to prevent serial data corruption or loss.

---

#### : Binary header

	Type	Length	Group	ID	Notes
CMD	C0	06	05	01	Variable-length command payload, value specified is minimum
RSP	C0	06	05	01	None.

#### Text info

Text name	Response length	Category	Notes
/CAC	0x0018	ACTION	None.

#### : Command arguments

Data type	Name	Text	Description
uint8	type	T*	<p>type:</p> <ul style="list-style-type: none"> <li>0 = structure</li> <li>1 = characteristic value</li> </ul> <p>Structural entries require constant data containing the definition. Structural entries optionally allow additional RAM data beyond the constant length for descriptor value information, such a two-byte CCCD values.</p> <p>Characteristic value entries do not require any constant data but may have it if a default boot-time value is desired.</p>
uint8	perm	P*	<p>Permission bits:</p> <ul style="list-style-type: none"> <li>Bit 0 (0x01) = Variable length</li> <li>Bit 1 (0x02) = Readable</li> <li>Bit 2 (0x04) = Write command (unacknowledged)</li> <li>Bit 3 (0x08) = Write request (acknowledged)</li> <li>Bit 4 (0x10) = Authenticated readable</li> <li>Bit 5 (0x20) = Reliable write (includes prepared write)</li> <li>Bit 6 (0x40) = Authenticated writeable</li> </ul>
uint16	length	L*	Indicates how many bytes of RAM are allocated for the definition (structure) or content (characteristic value)
longuint8a	data	D*	<p>Data (UUID or default attribute value where applicable)</p> <p>Data may include UUID, characteristic properties byte and/or value.</p> <p><b>Note:</b> longuint8a data type requires two prefixed "length" bytes before binary parameter payload.</p> <p>Characteristic properties:</p> <ul style="list-style-type: none"> <li>Bit 0 (0x01) = Broadcast</li> <li>Bit 1 (0x02) = Read</li> <li>Bit 2 (0x04) = Write without response</li> <li>Bit 3 (0x08) = Write</li> <li>Bit 4 (0x10) = Notify</li> <li>Bit 5 (0x20) = Indicate</li> <li>Bit 6 (0x40) = Signed write</li> <li>Bit 7 (0x80) = Extended properties (requires 0x2900)</li> </ul> <p>Characteristic declaration stores the UUID of the Characteristic value attribute. So its 'D' will be: 0x2803 (UUID) + Characteristic properties (1 byte) + handle of value attribute (2 byte) + UUID of value attribute.</p>

#### : Response parameters

Data type	Name	Text	Description
uint16	handle	H	New attribute handle (0x0001-0xFFFF)

Data type	Name	Text	Description
uint16	valid	V	GATT database validity status

**Related commands**

[gatts\\_delete\\_attr \(/CAD, ID=5/2\)](#)  
[gatts\\_validate\\_db \(/VGDB, ID=5/3\)](#)  
[gatts\\_dump\\_db \(/DGDB, ID=5/5\)](#)

**Related events**

[gatts\\_db\\_entry\\_blob \(DGATT, ID=5/4\)](#)

**Example usage**

Section [Defining custom local GATT services and characteristics](#)  
 Section [Adopted Bluetooth SIG GATT profile structure snippets](#)

**7.2.4.2 [gatts\\_delete\\_attr \(/CAD, ID=5/2\)](#)**

Remove one or more attributes from the GATT structure.

If you use this command without a handle in text mode or you supply handle value 0 in either text or binary mode, the highest attribute number (most recently added) is removed. If you supply a non-zero handle, the attribute with that handle and all higher handles are removed.

After removing an attribute with this command, the local GATT database may no longer be strictly valid. See section [EZ-Serial firmware platform for Vela IF820 GATT database validation error codes](#) for possible validity states. Use the [gatts\\_dump\\_db \(/DGDB, ID=5/5\)](#) API command to list the current local GATT database entries.

**Note:** EZ-Serial firmware platform for Ezurio Vela IF820 series module includes a fixed set of attributes as part of the core functionality, which cannot be deleted or modified. These attributes occupy the handle range from 1 (0x0001) to 28 (0x001C). Therefore, you cannot delete any attribute with a handle value less than 29 (0x001D).

**Note:** Additions to and removals from the GATT structure are always stored in flash. If the “result” value in the response indicates success, the change is effective immediately and persists through power cycles and resets. The internal CPU is occupied for approximately 15 ms during each flash write operation; during this time, no other activity is processed (UART or BLE communication). Any UART data sent during this brief window is lost. Therefore, you should modify the GATT structure only while disconnected, and you should allow a gap of at least 20 ms between the end of one API command and the beginning of a new one. If you have enabled hardware flow control using the [system\\_set\\_uart\\_parameters \(STU, ID=2/25\)](#) API command, EZ-Serial firmware platform for Ezurio Vela IF820 series module blocks incoming data flow during flash writes to prevent serial data corruption or loss.

**: Binary header**

	Type	Length	Group	ID	Notes
CMD	C0	02	05	02	None.
RSP	C0	08	05	02	None.

**Text info**

Text name	Response length	Category	Notes
/CAC	0x0018	ACTION	None.

#### Command arguments

Data type	Name	Text	Description
uint16	handle	H	Attribute handle to remove (includes all higher attributes)

#### : Response parameters

Data type	Name	Text	Description
uint16	Count	C	Number of attributes deleted from GATT structure
uint16	next_handle	H	Next available attribute handle after removal
uint16	valid	V	GATT database validity status

#### Related commands

gatts\_create\_attr (/CAC, ID=5/1)  
gatts\_validate\_db (/VGDB, ID=5/3)  
gatts\_dump\_db (/DGDB, ID=5/5)

#### 7.2.4.3 gatts\_validate\_db (/VGDB, ID=5/3)

Check to ensure that the custom GATT structure has no malformed or missing elements.

Use this command to check for errors in the custom GATT structure configured in EZ-Serial firmware platform for Ezurio Vela IF820 series module. The dynamic GATT implementation automatically tests for validity issues when making changes to the structure with the **gatts\_create\_attr (/CAC, ID=5/1)** and **gatts\_delete\_attr (/CAD, ID=5/2)** API commands, but this command provides the same test result upon request without making or attempting any modifications. See section **EZ-Serial firmware platform for Vela IF820 GATT database validation error codes** for possible validity states.

EZ-Serial firmware platform for Ezurio Vela IF820 series module allows only one non-valid state, indicated by GATTS\_DB\_VALID\_WARNING\_NOT\_ENOUGH\_ATTRIBUTES code (0x0001). This non-valid state is unavoidable during custom attribute creation because attributes must be added one at a time, and every new service or characteristic requires multiple attributes. All other non-valid states prevent the addition of a custom attribute in the first place. Therefore, running this command should result only in a valid state (0x0000) or the warning state noted here (0x0001).

#### : Binary header

	Type	Length	Group	ID	Notes
CMD	C0	00	05	03	None.
RSP	C0	04	05	03	None.

#### Text info

Text name	Response length	Category	Notes
/VGDB	0x0012	ACTION	None.

#### : Command arguments

None.

#### : Response parameters

Data type	Name	Text	Description
uint16	Valid	V	GATT database validity status

#### Related commands

gatts\_create\_attr (/CAC, ID=5/1)  
gatts\_delete\_attr (/CAD, ID=5/2)  
gatts\_dump\_db (/DGDB, ID=5/5)

#### 7.2.4.4 gatts\_store\_db (/SGDB, ID=5/4)

Store the current custom GATT structure in flash.

**Note:** This command has been deprecated and has no effect when used. As of the latest firmware build, GATT database changes are always written instantly to flash when using either **gatts\_create\_attr (/CAC, ID=5/1)** or **gatts\_delete\_attr (/CAD, ID=5/2)**.

#### : Binary header

	Type	Length	Group	ID	Notes
CMD	C0	00	05	04	None.
RSP	C0	02	05	04	None.

#### Text info

Text name	Response length	Category	Notes
/SGDB	0x000B	ACTION	None.

#### : Command arguments

None.

#### : Response parameters

None.

#### Related commands

gatts\_create\_attr (/CAC, ID=5/1)  
gatts\_delete\_attr (/CAD, ID=5/2)  
gatts\_validate\_db (/VGDB, ID=5/3)  
gatts\_dump\_db (/DGDB, ID=5/5)

#### 7.2.4.5 gatts\_dump\_db (/DGDB, ID=5/5)

List current local GATT database attributes.

This command produces a series of **gatts\_db\_entry\_blob (DGATT, ID=5/4)** API events, one for each attribute in the current local GATT database. The output is similar to that of the **gatts\_discover\_descriptors (/DLD, ID=5/8)** API command, but in a format that more closely matches the input parameters of the **gatts\_create\_attr (/CAC, ID=5/1)** API command.

You can choose to dump only those attributes in the user-definable range (0x001D and above), or include fixed attributes as well (0x0001 and above) for complete reference.

#### : Binary header

	Type	Length	Group	ID	Notes
CMD	C0	01	05	05	None.
RSP	C0	04	05	05	None.

## Text info

Text name	Response length	Notes
/DGDB	0x0012	None.

## : Command arguments

Data type	Name	Text	Description
uint8	include_fixed	F	Include fixed attributes: 0 = Start from handle 0x0015, do not include fixed attributes (default) 1 = Start from handle 0x0001, include fixed attributes

## : Response parameters

Data type	Name	Text	Description
uint16	Count	C	Number of entries to be returned

## Related commands

[gatts\\_create\\_attr \(/CAC, ID=5/1\)](#)  
[gatts\\_delete\\_attr \(/CAD, ID=5/2\)](#)  
[gatts\\_validate\\_db \(/VGDB, ID=5/3\)](#)  
[gatts\\_discover\\_descriptors \(/DLD, ID=5/8\)](#)

## Related events

[gatts\\_db\\_entry\\_blob \(DGATT, ID=5/4\)](#)

7.2.4.6 [gatts\\_discover\\_services \(/DLS, ID=5/6\)](#)

Request a list of all services in the local GATT structure.

This allows convenient discovery of services within the local GATT database. This command does not require an active connection because it concerns only local resources. Normally, you should not need to use this command except during development because the application should already know all relevant details about its own local GATT structure. To find all services in the local database, use "0" for both arguments, or explicitly set 0x0001 and 0xFFFF for the beginning and end handles.

The [gatts\\_discover\\_result \(DL, ID=5/1\)](#) API events resulting from this command will be generated when any local GATT services discovered.

For local GATT database information that more closely matches the input format required for the [gatts\\_create\\_attr \(/CAC, ID=5/1\)](#) API command, use the [gatts\\_dump\\_db \(/DGDB, ID=5/5\)](#) API command instead.

## : Binary header

	Type	Length	Group	ID	Notes
CMD	C0	04	05	06	None.
RSP	C0	04	05	06	None.

## Text info

Text name	Response length	Category	Notes
/DLS	0x0011	ACTION	None.

## : Command arguments

Data type	Name	Text	Description
uint16	begin	B	Handle to begin searching



Data type	Name	Text	Description
uint16	end	E	Handle to end searching (inclusive)

**: Response parameters**

Data type	Name	Text	Description
uint16	Count	C	Number of entries to be returned

**Related commands**

[gatts\\_dump\\_db \(/DGDB, ID=5/5\)](#)  
[gatts\\_discover\\_characteristics \(/DLC, ID=5/7\)](#)  
[gatts\\_discover\\_descriptors \(/DLD, ID=5/8\)](#)

**Related events**

[gatts\\_discover\\_result \(DL, ID=5/1\)](#)

**Example usage**

**Note:** Any attribute that requires authentication (bonding) must also require encryption. If you enable the authentication bit, ensure that you also enable the encryption bit, or the command will be rejected with an error result.

Section [Listing Local GATT Services, Characteristics, and Descriptors](#)**7.2.4.7 [gatts\\_discover\\_characteristics \(/DLC, ID=5/7\)](#)**

Request a list of all characteristics in the local GATT structure.

This allows convenient discovery of characteristics within the local GATT database. This command does not require an active connection because it concerns only local resources. Normally, you should not need to use this command except during development because the application should already know all relevant details about its own local GATT structure. To find all characteristics in the local database, use "0" for both arguments, or explicitly set 0x0001 and 0xFFFF for the beginning and end handles.

The [gatts\\_discover\\_result \(DL, ID=5/1\)](#) API events resulting from this command will be generated when any local GATT characteristics discovered.

For local GATT database information that more closely matches the input format required for the [gatts\\_create\\_attr \(/CAC, ID=5/1\)](#) API command, use the [gatts\\_dump\\_db \(/DGDB, ID=5/5\)](#) API command instead.

**: Binary header**

	Type	Length	Group	ID	Notes
CMD	C0	06	05	07	None.
RSP	C0	04	05	07	None.

**Text info**

Text name	Response length	Category	Notes
/DLC	0x0011	ACTION	None.

**: Command arguments**

Data type	Name	Text	Description
uint16	begin	B	Handle to begin searching
uint16	end	E	Handle to end searching (inclusive)

Data type	Name	Text	Description
uint16	service	S	Service UUID filter (0 for all) – Currently not implemented in firmware, set to 0

**: Response parameters**

Data type	Name	Text	Description
uint16	Count	C	Number of entries to be returned

**Related commands**

[gatts\\_dump\\_db \(/DGDB, ID=5/5\)](#)  
[gatts\\_discover\\_services \(/DLS, ID=5/6\)](#)  
[gatts\\_discover\\_descriptors \(/DLD, ID=5/8\)](#)

**Related events**

[gatts\\_discover\\_result \(DL, ID=5/1\)](#)

**Example usage**

See section [Listing Local GATT Services, Characteristics, and Descriptors](#)

**7.2.4.8 [gatts\\_discover\\_descriptors \(/DLD, ID=5/8\)](#)**

Request a list of all descriptors in the local GATT structure.

This allows convenient discovery of descriptors within the local GATT database. This command does not require an active connection because it concerns only local resources. Normally, you should not need to use this command except during development because the application should already know all relevant details about its own local GATT structure. To find all descriptors in the local database, use "0" for both arguments, or explicitly set 0x0001 and 0xFFFF for the beginning and end handles, respectively.

The [gatts\\_discover\\_result \(DL, ID=5/1\)](#) API events resulting from this command will be generated when any local GATT descriptors discovered.

For local GATT database information that more closely matches the input format required for the [gatts\\_create\\_attr \(/CAC, ID=5/1\)](#) API command, use the [gatts\\_dump\\_db \(/DGDB, ID=5/5\)](#) API command instead.

**: Binary header**

	Type	Length	Group	ID	Notes
CMD	C0	08	05	08	None.
RSP	C0	04	05	08	None.

**Text info**

Text name	Response length	Category	Notes
/DLD	0x0011	ACTION	None.

**: Command arguments**

Data type	Name	Text	Description
uint16	begin	B	Handle to begin searching
uint16	end	E	Handle to end searching (inclusive)
uint16	service	S	Service UUID filter (0 for all) (Ignored in current release, set to 0)
uint16	characteristic	C	Characteristic UUID filter (0 for all) (Ignored in current release, set to 0)

*Response parameters*

Data type	Name	Text	Description
uint16	Count	C	Number of entries to be returned

**Related commands**

[gatts\\_dump\\_db \(/DGDB, ID=5/5\)](#)  
[gatts\\_discover\\_services \(/DLS, ID=5/6\)](#)  
[gatts\\_discover\\_characteristics \(/DLC, ID=5/7\)](#)

**Related events**

[gatts\\_discover\\_result \(DL, ID=5/1\)](#)

**Example usage**

See section [Listing Local GATT Services, Characteristics, and Descriptors](#).

*7.2.4.9 gatts\_read\_handle (/RLH, ID=5/9)*

Read the value of an attribute in the local GATT Server.

This command does not require an active connection because it concerns only local resources.

*: Binary header*

	Type	Length	Group	ID	Notes
CMD	C0	02	05	09	None.
RSP	C0	04+	05	09	Variable-length response payload, value specified is minimum.

**Text info**

Text name	Response length	Category	Notes
/RLH	0x000D+	ACTION	Variable-length response payload, value specified is minimum.

*: Command arguments*

Data type	Name	Text	Description
uint16	attr_handle	H*	Handle of attribute to read value from

*: Response parameters*

Data type	Name	Text	Description
longuint8a	data	D	Data read from attribute

*Note:* longuint8a data type requires two prefixed "length" bytes before binary parameter payload

**Related commands**

[gatts\\_write\\_handle \(/WLH, ID=5/10\)](#)

*7.2.4.10 gatts\_write\_handle (/WLH, ID=5/10)*

Write a new value to an attribute in the local GATT Server.

This command does not require an active connection because it concerns only local resources.

**Note:** Writing data to a local characteristic value attribute does not automatically trigger a notification or indication of that data to a connected Client, even if the Client has subscribed to notifications or indications for the characteristic. This command affects only the value stored locally in RAM if the Client performs a GATT read operation later. To push data to a Client that subscribed to notifications or indications, use the `gatts_notify_handle (/NH, ID=5/11)` or `gatts_indicate_handle (/IH, ID=5/12)` API command.

#### : Binary header

	Type	Length	Group	ID	Notes
CMD	C0	04	05	0A	Variable-length command payload, value specified is minimum.
RSP	C0	02	05	0A	None.

#### Text info

Text name	Response length	Category	Notes
/WLH	0x000A	ACTION	None.

#### : Command arguments

Data type	Name	Text	Description
uint16	attr_handle	H*	Handle of attribute to read value from
longuint8a	data	D*	Data read from attribute

*Note:* longuint8a data type requires two prefixed "length" bytes before binary parameter payload

#### Response parameters

None.

#### Related commands

`gatts_read_handle (/RLH, ID=5/9)`  
`gatts_notify_handle (/NH, ID=5/11)`  
`gatts_indicate_handle (/IH, ID=5/12)`

#### 7.2.4.11 `gatts_notify_handle (/NH, ID=5/11)`

Notify a new attribute value to a remote GATT Client.

*Note:* This command does not change any locally stored values for the notified attribute. To modify the data stored locally in RAM for the attribute in question, use the `gatts_write_handle (/WLH, ID=5/10)` API command.

#### : Binary header

	Type	Length	Group	ID	Notes
CMD	C0	06	05	0B	Variable-length command payload, value specified is minimum.
RSP	C0	02	05	0B	None.

#### Text info

Text name	Response length	Category	Notes
/NH	0x0009	ACTION	None.

**: Command arguments**

Data type	Name	Text	Description
uint8	conn_handle	C	Connection handle to use for notification (Ignored in current release)
uint16	attr_handle	H*	Handle of attribute to notify
uint8a	data	D*	Data to push to remote Client via notification

*Note:* uint8a data type requires one prefixed "length" byte before binary parameter payload

**Response parameters**

None.

**Related commands**

[gatts\\_write\\_handle \(/WLH, ID=5/10\)](#)  
[gatts\\_indicate\\_handle \(/IH, ID=5/12\)](#)

#### 7.2.4.12 *gatts\_indicate\_handle* (/IH, ID=5/12)

Indicate a new attribute value to a remote GATT Client.

If successful, pushing an indicated value to a remote client results in the *gatts\_indication\_confirmed* (IC, ID=5/3) API event occurring after the client acknowledges the transfer.

This method requires client acknowledgement, so you cannot attempt another GATT operation until this confirmation event arrives. A single acknowledged transfer requires two connection intervals: one for the actual data transfer and one for the acknowledgement. Using this type of transfer has effects on potential throughput; see section *Maximizing throughput to a remote peer* for details on alternative design choices.

**Note:** This command does not change any locally stored values for the indicated attribute. To modify the data stored locally in RAM for the attribute in question, use the *gatts\_write\_handle* (/WLH, ID=5/10) API command.

##### : Binary header

	Type	Length	Group	ID	Notes
CMD	C0	06	05	0C	Variable-length command payload, value specified is minimum.
RSP	C0	02	05	0C	None.

##### Text info

Text name	Response length	Category	Notes
/IH	0x0009	ACTION	None.

##### : Command arguments

Data type	Name	Text	Description
uint8	conn_handle	C	Connection handle to use for indication (Ignored in current release)
uint16	attr_handle	H*	Handle of attribute to indicate
uint8a	data	D*	Data to indicate

**Note:** uint8a data type requires one prefixed "length" byte before binary parameter payload

##### Response parameters

None.

##### Related commands

*gatts\_read\_handle* (/RLH, ID=5/9)  
*gatts\_write\_handle* (/WLH, ID=5/10)  
*gatts\_notify\_handle* (/NH, ID=5/11)

##### Related events

*gatts\_indication\_confirmed* (IC, ID=5/3) - Occurs on the Server after the remote Client confirms receipt of indicated data

#### 7.2.4.13 *gatts\_send\_writereq\_response* (/WRR, ID=5/13)

Respond to a GATT client's acknowledged write request.

Use this command after receiving a *gatts\_data\_written* (W, ID=5/2) API event an acknowledged request to write data to a local GATT server attribute (the event's **type** parameter will be 0x80). Sending a response value of zero indicates success, while any non-zero value indicates an error. Values 0x01 through 0x7F are errors defined in the Bluetooth® specification, while values 0x80 through 0xFF are user-defined errors.

EZ-Serial firmware platform for Ezurio Vela IF820 series module will automatically respond to write requests unless **Bit 0** of the GATT server behavior flags is cleared using the **flags** field in the

**gatts\_set\_parameters** (SGSP, ID=5/14) API command, or if the characteristic being written has **Bit 24** set for user data management in the GATT database structure entry created with the **gatts\_create\_attr** (/CAC, ID=5/1) API command.

#### : Binary header

	Type	Length	Group	ID	Notes
CMD	C0	02	05	0D	None.
RSP	C0	02	05	0D	None.

#### Text info

Text name	Response length	Category	Notes
/WRR	0x000A	ACTION	None.

#### : Command arguments

Data type	Name	Text	Description
uint8	conn_handle	C	Connection handle to use for response (Ignored in current release due to internal BLE stack functionality, set to 0)
uint8	response	R*	GATT result code for response: 0 = Success 0x01-0x7F = Error from Bluetooth® specification 0x80-0xFF = Error from application (user-defined)

#### Response parameters

None.

#### Related commands:

**gattc\_write\_handle** (/WRH, ID=6/5)

#### Related events:

**gatts\_data\_written** (W, ID=5/2)

#### 7.2.4.14 gatts\_set\_parameters (SGSP, ID=5/14)

Configure new GATT Server parameters.

#### : Binary header

	Type	Length	Group	ID	Notes
CMD	C0	01	05	0E	None.
RSP	C0	02	05	0E	None.

#### Text info

Text name	Response length	Category	Notes
SGSP	0x000A	SET	None.

*: Command arguments*

Data type	Name	Text	Description
uint8	flags	F	GATT Server behavior flags bitmask: Bit 0 (0x01) = Enable automatic response to acknowledged writes
<b>Note:</b>			Factory default is 0x01 (all bits set)

**Response parameters**

None.

**Related commands**

*gatts\_get\_parameters (GGSP, ID=5/15)*

*7.2.4.15 gatts\_get\_parameters (GGSP, ID=5/15)*

Obtain current GATT Server parameters.

*: Binary header*

	Type	Length	Group	ID	Notes
CMD	C0	00	05	0F	None.
RSP	C0	03	05	0F	None.

**Text info**

Text name	Response length	Category	Notes
GGSP	0x000F	GET	None.

*: Command arguments*

None.

**Response parameters**

Data type	Name	Text	Description
uint8	flags	F	GATT Server behavior flags bitmask: Bit 0 (0x01) = Enable automatic response to acknowledged writes
<b>Note:</b>			Factory default is 0x01 (all bits set)

**Related commands**

*gatts\_set\_parameters (SGSP, ID=5/14)*



### 7.2.5 GATT Client Group (ID=6)

GATT Client methods relate to the client role of the GATT layer of the Bluetooth® stack. These methods are used for working with the GATT structures on remote devices, and can only be used while a device is connected.

Commands within this group are listed below:

`gattc_discover_services` (/DRS, ID=6/1)  
`gattc_discover_characteristics` (/DRC, ID=6/2)  
`gattc_discover_descriptors` (/DRD, ID=6/3)  
`gattc_read_handle` (/RRH, ID=6/4)  
`gattc_write_handle` (/WRH, ID=6/5)  
`gattc_confirm_indication` (/CI, ID=6/6)  
`gattc_set_parameters` (SGCP, ID=6/7)  
`gattc_get_parameters` (GGCP, ID=6/8)

Events within this group are documented in section [GATT Client Group \(ID=6\)](#).

#### 7.2.5.1 `gattc_discover_services` (/DRS, ID=6/1)

Request a list of GATT services from a connected remote GATT Server.

This command performs a GATT Client operation, and requires a connection to a remote peer. To discover the local GATT structure instead, use the `gatts_discover_services` (/DLS, ID=5/6) API command.

**Note:** Because this command works with remote data, it cannot determine the number of records to be returned in advance. Only local GATT Server discovery operations can do this. Therefore, you must wait for the `gattc_remote_procedure_complete` (RPC, ID=6/2) API event to indicate that the discovery procedure is finished.

#### : Binary header

	Type	Length	Group	ID	Notes
CMD	C0	05	06	01	None.
RSP	C0	02	06	01	None.

#### Text info

Text name	Response length	Category	Notes
/DRS	0x000A	ACTION	None.

#### : Command arguments

Data type	Name	Text	Description
uint8	conn_handle	C	Connection handle to use for discovery (Ignored in current release)
uint16	begin	B	Handle to begin searching
uint16	end	E	Handle to end searching (inclusive)

#### Response parameters

None.

#### Related commands

`gatts_discover_services` (/DLS, ID=5/6)  
`gattc_discover_characteristics` (/DRC, ID=6/2)  
`gattc_discover_descriptors` (/DRD, ID=6/3)

#### Related events:

`gattc_discover_result` (DR, ID=6/1)  
`gattc_remote_procedure_complete` (RPC, ID=6/2)

**Example usage:**

See section [How to discover a remote server's GATT structure](#).

**7.2.5.2 `gattc_discover_characteristics` (/DRC, ID=6/2)**

Request a list of GATT characteristics from a connected remote GATT Server.

This command performs a GATT Client operation, and requires a connection to a remote peer. To discover the local GATT structure instead, use the `gatts_discover_characteristics` (/DLC, ID=5/7) API command.

**Note:** Because this command works with remote data, it cannot determine the number of records to be returned in advance. Only local GATT Server discovery operations can do this. Therefore, you must wait for the `gattc_remote_procedure_complete` (RPC, ID=6/2) API event to indicate that the discovery procedure is finished.

**: Binary header**

	Type	Length	Group	ID	Notes
CMD	C0	07	06	02	None.
RSP	C0	02	06	02	None.

**Text info**

Text name	Response length	Notes
/DRC	0x000A	None.

**: Command arguments**

Data type	Name	Text	Description
uint8	conn_handle	C	Connection handle to use for discovery (Ignored in current release)
uint16	begin	B	Handle to begin searching
uint16	end	E	Handle to end searching (inclusive)
uint16	service	S	Service UUID filter (0 for all) (Ignored in current release, set to 0)

**Response parameters**

None.

**Related commands**

`gatts_discover_characteristics` (/DLC, ID=5/7)  
`gattc_discover_services` (/DRS, ID=6/1)  
`gattc_discover_descriptors` (/DRD, ID=6/3)

**Related events:**

`gattc_discover_result` (DR, ID=6/1)  
`gattc_remote_procedure_complete` (RPC, ID=6/2)

**Example usage:**

See section [How to discover a remote server's GATT structure](#).

### 7.2.5.3 *gattc\_discover\_descriptors (/DRD, ID=6/3)*

Request a list of GATT attribute descriptors from a connected remote GATT Server.

This command performs a GATT Client operation, and requires a connection to a remote peer. To discover the local GATT structure instead, use the *gatts\_discover\_descriptors (/DLD, ID=5/8)* API command.

**Note:** Because this command works with remote data, it cannot determine the number of records to be returned in advance. Only local GATT Server discovery operations can do this. Therefore, you must wait for the *gattc\_remote\_procedure\_complete (RPC, ID=6/2)* API event to indicate that the discovery procedure is finished.

#### : Binary header

	Type	Length	Group	ID	Notes
CMD	C0	09	06	03	None.
RSP	C0	02	06	03	None.

#### Text info

Text name	Response length	Category	Notes
/DRD	0x000A	ACTION	None.

#### : Command arguments

Data type	Name	Text	Description
uint8	conn_handle	C	Connection handle to use for discovery (Ignored in current release)
uint16	begin	B	Handle to begin searching
uint16	end	E	Handle to end searching (inclusive)
uint16	service	S	Service UUID filter (0 for all) (Ignored in current release, set to 0)
uint16	characteristic	T	Characteristic UUID filter (0 for all) (Ignored in current release, set to 0)

#### Response parameters

None.

#### Related commands

*gatts\_discover\_descriptors (/DLD, ID=5/8)*  
*gattc\_discover\_services (/DRS, ID=6/1)*  
*gattc\_discover\_characteristics (/DRC, ID=6/2)*

#### Related events:

*gattc\_discover\_result (DR, ID=6/1)*  
*gattc\_remote\_procedure\_complete (RPC, ID=6/2)*

#### Example usage:

See section [How to discover a remote server's GATT structure](#).

### 7.2.5.4 *gattc\_read\_handle (/RRH, ID=6/4)*

Read the value of an attribute on a remote GATT Server.

This command performs a GATT Client operation, and requires a connection to a remote peer. To read a value from the local GATT structure instead, use the `gatts_read_handle (/RLH, ID=5/9)` API command.

#### : Binary header

	Type	Length	Group	ID	Notes
CMD	C0	03	06	04	None.
RSP	C0	02	06	04	None.

#### Text info

Text name	Response length	Category	Notes
/RRH	0x000A	ACTION	None.

#### : Command arguments

Data type	Name	Text	Description
uint8	conn_handle	C	Connection handle to use for the read operation (Ignored in current release)
uint16	attr_handle	H*	Handle of remote attribute to read

#### Response parameters

None.

#### Related commands:

`gattc_write_handle (/WRH, ID=6/5)`

#### Related events:

`gattc_remote_procedure_complete (RPC, ID=6/2)` – Occurs if the Client Read operation fails (parameters include error code)

`gattc_data_received (D, ID=6/3)` – Occurs if the Client Read operation succeeds

#### 7.2.5.5 `gattc_write_handle (/WRH, ID=6/5)`

Write a new value to an attribute on a remote GATT Server.

This command performs a GATT Client operation, and requires a connection to a remote peer. To write a value to the local GATT structure instead, use the `gatts_write_handle (/WLH, ID=5/10)` API command.

#### : Binary header

	Type	Length	Group	ID	Notes
CMD	C0	06	06	05	Variable-length command payload, value specified is minimum.
RSP	C0	02	06	05	None.

#### Text info

Text name	Response length	Category	Notes
/WRH	0x000A	ACTION	None.

### : Command arguments

Data type	Name	Text	Description
uint8	conn_handle	C	Connection handle to use for write operation (Ignored in current release)
uint16	attr_handle	H*	Handle of the remote attribute to write
uint8	type	T	Type of write to perform: 0 = Simple write – acknowledged (default) 1 = Write without response – unacknowledged
longuint8a	data	D*	New data to write

**Note:** The longuint8a data type requires two prefixed “length” bytes before binary the parameter payload. In the current implementation, the length is 255 in MAX due to resource limitation.

### Response parameters

None.

### Related commands:

[gattc\\_read\\_handle \(/RRH, ID=6/4\)](#)

### Related events:

[gatts\\_data\\_written \(W, ID=5/2\)](#) – Occurs on the remote server after using this command on the local client

[gattc\\_remote\\_procedure\\_complete \(RPC, ID=6/2\)](#) – Occurs once the write is acknowledged, if using acknowledged write type

#### 7.2.5.6 [gattc\\_confirm\\_indication \(/CI, ID=6/6\)](#)

Confirm an indication from a remote GATT Server.

This command confirms the receipt of indicated data from a remote server. Indicated data is pushed from a server to a client after the client has subscribed to indications for a desired characteristic and that characteristic’s value has changed. Indicated data will arrive via the [gattc\\_data\\_received \(D, ID=6/3\)](#) API event; you must use this command to manually confirm the indication if the **source** parameter of that event shows indication with manual confirmation needed. See the event documentation for details.

EZ-Serial firmware platform for Ezurio Vela IF820 series module will automatically confirm indications unless **Bit 0** of the GATT Client behavior flags is cleared using the **flags** field in the [gattc\\_set\\_parameters \(SGCP, ID=6/7\)](#) API command.

**Note:** If the indicated data arrives and requires manual confirmation, you must use this command to confirm it before performing any other GATT operations.

### : Binary header

	Type	Length	Group	ID	Notes
CMD	C0	03	06	06	None.
RSP	C0	02	06	06	None.

### Text info

Text name	Response length	Category	Notes
/CI	0x0009	ACTION	None.

**: Command arguments**

Data type	Name	Text	Description
uint8	conn_handle	C	Connection handle to use for confirmation (Ignored in current release due)
uint16	attr_handle	H*	Attribute handle to confirm

**Response parameters:**

None.

**Related commands:****gatts\_indicate\_handle** (/IH, ID=5/12) – Used on a remote GATT Server to indicate data to a client**gattc\_set\_parameters** (SGCP, ID=6/7) – Configure local GATT Client parameters, including auto-confirm behavior**Related events:****gatts\_indication\_confirmed** (IC, ID=5/3) – Occurs on a remote GATT Server after confirming indication on the client**gattc\_data\_received** (D, ID=6/3) – Occurs on the local GATT Client when a remote server indicates data**7.2.5.7 gattc\_set\_parameters (SGCP, ID=6/7)**

Configure new GATT Client parameters.

**: Binary header**

	Type	Length	Group	ID	Notes
CMD	C0	01	06	07	None.
RSP	C0	02	06	07	None.

**Text info**

Text name	Response length	Category	Notes
SGCP	0x000A	SET	None.

**: Command arguments**

Data type	Name	Text	Description
uint8	flags	F	GATT Client behavior flags bitmask: Bit 0 (0x01) = Enable automatic confirmation of remote GATT Server indications
<b>Note:</b>			Factory default is 0x01 (all bits set)

**Response parameters:**

None.

**Related commands:****gattc\_confirm\_indication** (/CI, ID=6/6) – Necessary to use for indicated data if flags Bit 0 is clear**gattc\_get\_parameters** (GGCP, ID=6/8)**7.2.5.8 gattc\_get\_parameters (GGCP, ID=6/8)**

Get current GATT Client parameters.

*: Binary header*

	Type	Length	Group	ID	Notes
CMD	C0	00	06	08	None.
RSP	C0	03	06	08	None.

**Text info**

Text name	Response length	Category	Notes
GGCP	0x000F	GET	None.

*: Command arguments*

None.

*: Response parameters:*

Data type	Name	Text	Description
uint8	Flags	F	GATT Client behavior flags bitmask: Bit 0 (0x01) = Enable automatic confirmation of remote GATT Server indications

*Note:* Factory default is 0x01 (all bits set)

**Related commands:**

[gattc\\_set\\_parameters](#) (SGCP, ID=6/7)

### 7.2.6 SMP Group (ID=7)

SMP methods relate to the Security Manager Protocol layer of the Bluetooth® stack. These methods are used for working with privacy, encryption, pairing, and bonding between two devices.

Commands within this group are listed below:

```
smp_query_bonds (/QB, ID=7/1)
smp_delete_bond (/BD, ID=7/2)
smp_pair (/P, ID=7/3)
smp_set_privacy_mode (SPRV, ID=7/9)
smp_get_privacy_mode (GPRV, ID=7/10)
smp_set_security_parameters (SSBP, ID=7/11)
smp_get_security_parameters (GGBP, ID=7/12)
smp_set_fixed_passkey (SFPK, ID=7/13)
smp_get_fixed_passkey (GFPK, ID=7/14)
```

Events within this group are documented in section [SMP Group \(ID=7\)](#).

#### 7.2.6.1 smp\_query\_bonds (/QB, ID=7/1)

Request a list of bonded devices.

This command accesses the current bonded device list. Bonded devices are those which have previously paired (exchanged encryption data) and bonded (stored the exchanged encryption data).

The response from this command includes the number of bonded devices, and the response are followed by the [smp\\_bond\\_entry \(B, ID=7/1\)](#) API events that provide details for each device.

**Note:** EZ-Serial firmware platform for Ezurio Vela IF820 series module currently supports **a maximum of four bonded devices at the same time**. To bond with additional devices after all four bond slots are full, you must delete one of the existing bonds with the [smp\\_delete\\_bond \(/BD, ID=7/2\)](#) API command.

#### : Binary header

	Type	Length	Group	ID	Notes
CMD	C0	00	07	01	None.
RSP	C0	03	07	01	None.

#### Text info

Text name	Response length	Category	Notes
/QB	0x000E	ACTION	None.

#### : Command arguments

None.

#### : Response parameters:

Data type	Name	Text	Description
uint8	Count	C	Bond entry count

#### Related commands:

[smp\\_pair \(/P, ID=7/3\)](#) – Creates a new bond entry if pairing process succeeds with bonding enabled

#### Related events

[smp\\_bond\\_entry \(B, ID=7/1\)](#) – Occurs once for each bonded device after requesting bond list



### 7.2.6.2 smp\_delete\_bond (/BD, ID=7/2)

Remove a bonded device.

This command removes the stored encryption key data for a device that has previously paired (exchanged encryption data) and bonded (stored the exchanged encryption data).

#### : Binary header

	Type	Length	Group	ID	Notes
CMD	C0	07	07	02	None.
RSP	C0	03	07	02	None.

#### Text info

Text name	Response length	Category	Notes
/BD	0x000E	ACTION	None.

#### : Command arguments

Data type	Name	Text	Description
Macaddr	address	A*	Bluetooth® address
uint8	type	T	Address type: 0 = Public (default) 1 = Random/private Address type does not effect delete operation in current implementation.

#### : Response parameters:

Data type	Name	Text	Description
uint8	count	C	Updated bond entry count

#### Related commands

smp\_query\_bonds (/QB, ID=7/1)

smp\_pair (/P, ID=7/3) – Creates a new bond entry if pairing process succeeds with bonding enabled

### 7.2.6.3 smp\_pair (/P, ID=7/3)

Initiate pairing process with a connected device.

**Note:** EZ-Serial firmware platform for Ezurio Vela IF820 series module **currently supports a maximum of 8 bonded devices** at the same time. To bond with additional devices after all 8 bond slots are full, you must delete one of the existing bonds with the **smp\_delete\_bond (/BD, ID=7/2)** API command.

#### : Binary header

	Type	Length	Group	ID	Notes
CMD	C0	05	07	03	None.
RSP	C0	02	07	03	None.

#### Text info

Text name	Response length	Category	Notes
/P	0x0008	ACTION	None.

#### : Command arguments

Data type	Name	Text	Description
• uint8	• conn_handle	• C	• Connection handle to use for pairing

Data type	Name	Text	Description
uint8	mode	M	Security level setting reported to peer: always 0
uint8	bonding	B	Bond during pairing process: not implemented, always zero
uint8	keysize	K	Encryption key size (7-16), value ignored if pairing initiated by slave device Factory default is 16 bytes (0x10)
uint8	pairprop	P	Pairing properties: always 0

#### : Response parameters:

None.

#### Related commands

**smp\_set\_security\_parameters** (SSBP, ID=7/11) – Use to configure default security settings

#### Related events

**smp\_pairing\_requested** (P, ID=7/2) – Occurs when remote device initiates pairing

**smp\_pairing\_result** (PR, ID=7/3) – Occurs when pairing process completes (success or failure)

**smp\_encryption\_status** (ENC, ID=7/4) – Occurs when encryption status changes during a pairing process

#### 7.2.6.4 smp\_set\_privacy\_mode (SPRV, ID=7/9)

Configure new privacy settings.

Use this command to enable or disable Peripheral or Central privacy. Enabling privacy in each mode causes the Bluetooth® connection address used in related states to be random (private) instead of fixed (public). This can make passive profiling by a remote observer more difficult.

Peripheral privacy affects the Bluetooth® connection address broadcast during advertisements, which the remote Central device may log or use for a scan request or connection request. Central privacy affects the Bluetooth® connection address used for scan requests or connection requests when scanning for or communicating with a remote device.

#### : Binary header

	Type	Length	Group	ID	Notes
CMD	C0	03	07	09	None.
RSP	C0	02	07	09	None.

#### Text info

Text name	Response length	Category	Notes
SPRV	0x000A	SET	None.

#### : Command arguments

Data type	Name	Text	Description
uint8	mode	M	Privacy mode bitmask: Bit 0 (0x01) = Enable Peripheral privacy Bit 1 (0x02) = Enable Central privacy Bit 2 (0x04) = Enable Random address  Factory default is 0x04 (Enable Random address) Current FW does not differ Peripheral privacy and Central privacy.
uint16	interval	I	Randomization interval (seconds): Not Available

#### : Response parameters:

None.

## Related commands

`smp_get_privacy_mode` (GPRV, ID=7/10)

### 7.2.6.5 `smp_get_privacy_mode` (GPRV, ID=7/10)

Obtain current privacy settings.

#### : Binary header

	Type	Length	Group	ID	Notes
CMD	C0	00	07	0A	None.
RSP	C0	05	07	0A	None.

#### Text info

Text name	Response length	Category	Notes
GPRV	0x0016	GET	None.

#### : Command arguments

None.

#### : Response parameters:

Data type	Name	Text	Description
uint8	Mode	M	Privacy mode bitmask: Bit 0 (0x01) = Enable Peripheral privacy Bit 1 (0x02) = Enable Central privacy Bit 2 (0x04) = Enable Random address Factory default is 0x04 (Enable Random address)
uint16	interval	I	Randomization interval (seconds): Not Available

## Related commands

`smp_set_privacy_mode` (SPRV, ID=7/9)

### 7.2.6.6 `smp_set_security_parameters` (SSBP, ID=7/11)

Configure new security and bonding parameters.

These parameters are used when the `smp_pair` (IP, ID=7/3) API command is used without specifying non-default arguments. These values are reported to the remote device as part of the pairing process and affect the type of key generation and exchange that takes place during pairing and bonding.

**Note:** Changing the I/O capabilities affects the command/event flow necessary to complete a pairing and bonding process. See the related commands and events for details concerning each one's use. Also, MITM protection requires I/O capabilities other than "No Input + No Output" to function correctly.

#### : Binary header

	Type	Length	Group	ID	Notes
CMD	C0	06	07	0B	None.
RSP	C0	02	07	0B	None.

#### Text info

Text name	Response length	Category	Notes
SSBP	0x000A	SET	None.

### : Command arguments

uint8	mode	M	<p>High four bits are for BT classic:</p> <p>0 = MITM Protection Not Required - Single Profile/non-bonding. Numeric comparison with automatic accept allowed.</p> <p>1 = MITM Protection Required - Single Profile/non-bonding. Use IO Capabilities to determine authentication procedure.</p> <p>2 = MITM Protection Not Required - All Profiles/dedicated bonding. Numeric comparison with automatic accept allowed.</p> <p>3 = MITM Protection Required - All Profiles/dedicated bonding. Use IO Capabilities to determine authentication procedure.</p> <p>4 = MITM Protection Not Required - Single Profiles/general bonding. Numeric comparison with automatic accept allowed.</p> <p>5 = MITM Protection Required - Single Profiles/general bonding. Use IO Capabilities to determine authentication procedures.</p> <p>Low four bits are for BLE:</p> <p>0x00 = Not required - No Bond</p> <p>0x01 = Required - General Bond</p> <p>0x04 = MITM required - Auth Y/N</p> <p>0x08 = LE Secure Connection, no MITM, no Bonding</p> <p>0x08I0x01 = LE Secure Connection, no MITM, Bonding</p> <p>0x08I0x04 = LE Secure Connection, MITM, no Bonding</p> <p>0x08I0x04I0x01= LE Secure Connection, MITM, Bonding</p>
uint8	bonding	B	Bond during pairing process: not implemented, always zero
uint8	keysize	K	Encryption key size (7-16), value ignored if pairing initiated by slave device Factory default is 16 bytes (0x10)
uint8	pairprop	P	Pairing properties: Don't care and always 0
uint8	io	I	<p>I/O capabilities:</p> <p>0 = Display Only – ability to convey a 6-digit number to user</p> <p>1 = Display + Yes/No – display and the ability to have user indicate “yes” or “no”</p> <p>2 = Keyboard Only – ability for the user to enter ‘0’ through ‘9’ and “yes” or “no”</p> <p>3 = No Input + No Output – no ability to display or input anything (factory default)</p> <p>4 = Keyboard + Display – ability to provide full numeric input and display</p>
uint8	flags	F	<p>Security behavior flags bitmask:</p> <p>Bit 0 (0x01) = Enable auto-accept for incoming pairing requests (Always be 1)</p> <p>Bit 1 (0x02) = Enable use of fixed passkey during pairing</p> <p>Bit 2 (0x04) = Enable use of legacy PIN code during paring for BT classic device.</p> <p>Factory default is 0x01</p>

### : Response parameters:

None.

### Related commands

[smp\\_pair \(/P, ID=7/3\)](#)

[smp\\_get\\_security\\_parameters \(GSBP, ID=7/12\)](#)

[smp\\_set\\_fixed\\_passkey \(SFPK, ID=7/13\)](#)

smp\_set\_pin\_code (SBTPIN, ID=7/15)

smp\_get\_pin\_code (GBTPIN, ID=7/16)

#### Related events

smp\_pairing\_requested (P, ID=7/2)

smp\_pairing\_result (PR, ID=7/3)

smp\_encryption\_status (ENC, ID=7/4)

#### 7.2.6.7 smp\_get\_security\_parameters (GSBP, ID=7/12)

Obtain current security and bonding parameters.

#### : Binary header

	Type	Length	Group	ID	Notes
CMD	C0	06	07	0B	None.
RSP	C0	02	07	0B	None.

#### Text info

Text name	Response length	Category	Notes
SSBP	0x000A	SET	None.

#### : Command arguments:

None.

#### : Response parameters:

Data type	Name	Text	Description
uint8	Mode	M	<p>Security level setting reported to peer:</p> <p>High four bits are for BT classic:</p> <ul style="list-style-type: none"> <li>0 = MITM Protection Not Required - Single Profile/non-bonding. Numeric comparison with automatic accept allowed.</li> <li>1 = MITM Protection Required - Single Profile/non-bonding. Use IO Capabilities to determine authentication procedure.</li> <li>2 = MITM Protection Not Required - All Profiles/dedicated bonding. Numeric comparison with automatic accept allowed.</li> <li>3 = MITM Protection Required - All Profiles/dedicated bonding. Use IO Capabilities to determine authentication procedure.</li> <li>4 = MITM Protection Not Required - Single Profiles/general bonding. Numeric comparison with automatic accept allowed.</li> <li>5 = MITM Protection Required - Single Profiles/general bonding. Use IO Capabilities to determine authentication procedures.</li> </ul> <p>Low four bits are for BLE:</p> <ul style="list-style-type: none"> <li>0x00 = Not required - No Bond</li> <li>0x01 = Required - General Bond</li> <li>0x04 = MITM required - Auth Y/N</li> <li>0x08 = LE Secure Connection, no MITM, no Bonding</li> <li>0x08I0x01 = LE Secure Connection, no MITM, Bonding</li> <li>0x08I0x04 = LE Secure Connection, MITM, no Bonding</li> <li>0x08I0x04I0x01 = LE Secure Connection, MITM, Bonding</li> </ul>
uint8	bonding	B	<p>Bond during pairing process:</p> <ul style="list-style-type: none"> <li>0 = Do not bond (exchange keys and encrypt only)</li> <li>1 = Bond (permanently store exchanged encryption data)</li> </ul>
uint8	keysize	K	<p>Encryption key size (7-16), value ignored if pairing initiated by slave device</p> <p>Factory default is 16 bytes (0x10)</p>

Data type	Name	Text	Description
uint8	pairprop	P	Pairing properties: Don't care and always 0
uint8	lo	I	I/O capabilities: <ul style="list-style-type: none"> <li>0 = Display Only – ability to convey a 6-digit number to user</li> <li>1 = Display + Yes/No – display and the ability to have user indicate “yes” or “no”</li> <li>2 = Keyboard Only – ability for the user to enter '0' through '9' and “yes” or “no”</li> <li>3 = No Input + No Output – no ability to display or input anything (factory default)</li> <li>4 = Keyboard + Display – ability to provide full numeric input and display</li> </ul>
uint8	Flags	F	Security behavior flags bitmask: <ul style="list-style-type: none"> <li>Bit 0 (0x01) = Enable auto-accept for incoming pairing requests</li> <li>Bit 1 (0x02) = Enable use of fixed passkey during pairing</li> <li>Bit 2 (0x04) = Enable use of legacy PIN code during pairing for BT classic device.</li> </ul> Factory default is 0x01

### Related commands

[smp\\_set\\_security\\_parameters \(SSBP, ID=7/11\)](#)

[smp\\_set\\_pin\\_code \(SBTPIN, ID=7/15\)](#)

[smp\\_get\\_pin\\_code \(GBTPIN, ID=7/16\)](#)

#### 7.2.6.8 [smp\\_set\\_fixed\\_passkey \(SFPK, ID=7/13\)](#)

Configure new fixed passkey value.

While the Bluetooth® specification describes that the passkey should be randomized during pairing, you can configure a fixed (non-random) 6-digit passkey between 000000 and 999999 using this command and configuring the local I/O capabilities to the “Display Only” value.

**Note:** The fixed passkey defined here takes effect only if you enable fixed passkey use by setting Bit 1 (0x02) of the security flags parameter and set the “Display Only” I/O capabilities value (0x00) using the [smp\\_set\\_security\\_parameters \(SSBP, ID=7/11\)](#) API command. If both conditions are not met, then the stack will revert to the default behavior of using a random passkey.

**Note:** This feature is obsolete since it is not Bluetooth® spec suggest.

### : Binary header

	Type	Length	Group	ID	Notes
CMD	C0	04	07	0D	None.
RSP	C0	02	07	0D	None.

### Text info

Text name	Response length	Category	Notes
SFPK	0x000A	SET	None.

### : Command arguments

Data type	Name	Text	Description
uint32	passkey	P	Fixed passkey value Minimum = 0 ('000000' decimal entry during pairing) Maximum = 0xF423F ('999999' decimal entry during pairing) Factory default is 0

### : Response parameters:

None.

### Related commands

`smp_pair` (/P, ID=7/3)

`smp_get_fixed_passkey` (GFPK, ID=7/14)

`smp_set_security_parameters` (SSBP, ID=7/11)

#### Related events

`smp_pairing_requested` (P, ID=7/2)

`smp_pairing_result` (PR, ID=7/3)

`smp_encryption_status` (ENC, ID=7/4)

#### Example Usage

See section [Pairing with a fixed passkey](#)

#### 7.2.6.9 `smp_get_fixed_passkey` (GFPK, ID=7/14)

Obtain current fixed passkey value.

##### : Binary header

	Type	Length	Group	ID	Notes
CMD	C 0	00	07	0E	None.
RSP	C 0	08	07	0E	None.

##### Text info

Text name	Response length	Category	Notes
GFPK	0x0015	GET	None.

##### : Command arguments

None.

##### : Response parameters:

Data type	Name	Text	Description
uint32	passkey	P	Fixed passkey value: Minimum = 0 ('000000' decimal entry during pairing) Maximum = 0xF423F ('999999' decimal entry during pairing) <b>Note:</b> Factory default is 0

#### Related commands

`smp_set_fixed_passkey` (SFPK, ID=7/13)

#### 7.2.6.10 `smp_set_pin_code` (SBTPIN, ID=7/15)

Configure new PIN code value for BT classic device.

##### : Binary header

	Type	Length	Group	ID	Notes
CMD	C 0	01	07	0F	Variable-length command payload, value specified is minimum
RSP	C 0	02	07	0F	None.

##### Text info

Text name	Response length	Category	Notes
SBTPIN	0x000C	SET	None.

**: Command arguments**

Data type	Name	Text	Description
uint8a	PIN code	P	PIN code data (1-16 bytes)
<b>Note:</b> uint8a data type requires one prefixed "length" byte before binary parameter payload.			
<b>Note:</b> Factory default is "0000" (0x30,0x30,0x30,0x30), length 0x04.			

**Response parameters**

None.

**Related commands**[smp\\_get\\_pin\\_code \(GBTPIN, ID=7/16\)](#)[smp\\_set\\_security\\_parameters \(SSBP, ID=7/11\)](#)**7.2.6.11 [smp\\_get\\_pin\\_code \(GBTPIN, ID=7/16\)](#)**

Obtain current PIN code value.

**: Binary header**

	Type	Length	Group	ID	Notes
CMD	C0	00	07	10	None.
RSP	C0	03	07	10	None.

**Text info**

Text name	Response length	Category	Notes
GBTPIN	0x0010F	GET	Variable-length command payload, value specified is minimum

**: Command arguments**

None.

**: Response parameters**

Data type	Name	Text	Description
uint8a16	PIN code	P	PIN code data (1-16 bytes)
uint8a data type requires one prefixed "length" byte before binary parameter payload			

**Related commands**[smp\\_set\\_pin\\_code \(SBTPIN, ID=7/15\)](#)



#### 7.2.6.12 *smp\_send\_pinreq\_response (/BTPIN, ID=7/17)*

Sends the PIN code value back to a remote device waiting for it.

##### *: Binary header*

	Type	Length	Group	ID	Notes
CMD	C0	05	07	11	None.
RSP	C0	02	07	11	None.

##### **Text info**

Text name	Response length	Notes
/BTPIN	0x000C	None.

##### *: Command arguments*

Data type	Name	Text	Description
uint8	conn_handle	C	Connection handle
uint32	pin_code	P*	PIN code value

##### *: Response parameters*

None.

##### **Related commands:**

*smp\_pair (/P, ID=7/3)*

##### **Related events:**

*smp\_pin\_entry\_requested (BTPIN, ID=7/7)*

### 7.2.7 GPIO Group (ID=9)

GPIO methods relate to the physical pins on the module.

Commands within this group are listed below:

```
gpio_query_adc (/QADC, ID=9/2)
gpio_set_drive (SIOD, ID=9/5)
gpio_get_drive (GIOD, ID=9/6)
gpio_set_logic (SIOL, ID=9/7)
gpio_get_logic (GIOL, ID=9/8)
gpio_set_pwm_mode (SPWM, ID=9/11)
gpio_get_pwm_mode (GPWM, ID=9/12)
```

Events within this group are documented in section [GPIO Group \(ID=9\)](#).

#### 7.2.7.1 *gpio\_query\_adc (/QADC, ID=9/2)*

Read the immediate analog voltage level on the selected channel.

EZ-Serial firmware platform for Ezurio Vela IF820 series module provides a single dedicated ADC input pin (ADC0) for reading analog voltages. The ADC supports an input voltage range of 0 V minimum to VDD (usually 3.3 V) maximum. Use this command to perform a single ADC conversion. Once the conversion completes, the module transmits the result back in response parameters.

See section [GPIO pin map](#) for a pin map table showing ADC pin assignment.

ADC channel is internal defined, see the following definition which copied from WICED SDK, user can check the GPIO availability in the models and find the right channel index:

```
ADC_INPUT_P17      = 0x0,    // ADC CHANNEL #1 on GPIO P17
ADC_INPUT_P16      = 0x1,    // ADC CHANNEL #2 on GPIO P16
ADC_INPUT_P15      = 0x2,    // ADC CHANNEL #3 on GPIO P15
ADC_INPUT_P14      = 0x3,    // ADC CHANNEL #4 on GPIO P14
ADC_INPUT_P13      = 0x4,    // ADC CHANNEL #5 on GPIO P13
ADC_INPUT_P12      = 0x5,    // ADC CHANNEL #6 on GPIO P12
ADC_INPUT_P11      = 0x6,    // ADC CHANNEL #7 on GPIO P11
ADC_INPUT_P10      = 0x7,    // ADC CHANNEL #8 on GPIO P10
ADC_INPUT_P9       = 0x8,    // ADC CHANNEL #9 on GPIO P9
ADC_INPUT_P8       = 0x9,    // ADC CHANNEL #10 on GPIO P8
ADC_INPUT_P1       = 0xA,    // ADC CHANNEL #11 on GPIO P1
ADC_INPUT_P0       = 0xB,    // ADC CHANNEL #12 on GPIO P0
ADC_INPUT_VDDIO    = 0xC,    // ADC_INPUT_VBAT_VDDIO on Channel 13
ADC_INPUT_VDD_CORE = 0xD,    // ADC_INPUT_VDDC on Channel 14
ADC_INPUT_ADC_BGREF = 0xE,    // ADC BANDGAP REF on Channel 15
ADC_INPUT_ADC_REFGND = 0xF,  // ADC REF GND on Channel 16
ADC_INPUT_P38      = 0x10,   // ADC CHANNEL #17 on GPIO P38
ADC_INPUT_P37      = 0x11,   // ADC CHANNEL #18 on GPIO P37
ADC_INPUT_P36      = 0x12,   // ADC CHANNEL #19 on GPIO P36
ADC_INPUT_P35      = 0x13,   // ADC CHANNEL #20 on GPIO P35
ADC_INPUT_P34      = 0x14,   // ADC CHANNEL #21 on GPIO P34
ADC_INPUT_P33      = 0x15,   // ADC CHANNEL #22 on GPIO P33
```

```

ADC_INPUT_P32      = 0x16,    // ADC CHANNEL #23 on GPIO P32
ADC_INPUT_P31      = 0x17,    // ADC CHANNEL #24 on GPIO P31
ADC_INPUT_P30      = 0x18,    // ADC CHANNEL #25 on GPIO P30
ADC_INPUT_P29      = 0x19,    // ADC CHANNEL #26 on GPIO P29
ADC_INPUT_P28      = 0x1A,    // ADC CHANNEL #27 on GPIO P28
ADC_INPUT_P23      = 0x1B,    // ADC CHANNEL #28 on GPIO P23
ADC_INPUT_P22      = 0x1C,    // ADC CHANNEL #29 on GPIO P22
ADC_INPUT_P21      = 0x1D,    // ADC CHANNEL #30 on GPIO P21
ADC_INPUT_P19      = 0x1E,    // ADC CHANNEL #31 on GPIO P19
ADC_INPUT_P18      = 0x1F,    // ADC CHANNEL #32 on GPIO P18
ADC_INPUT_CHANNEL_MASK = 0x1f,

```

#### : Binary header

	Type	Length	Group	ID	Notes
CMD	C0	01	09	02	None.
RSP	C0	02	09	02	None.

#### Text info

Text name	Response length	Category	Notes
/QADC	0x000B	ACTION	None.

#### : Command arguments

Data type	Name	Text	Description
uint8	channel	N*	ADC channel (0 ~31)
uint8	reference	R	Voltage reference for conversion (Ignored in current release, set to 0 and VDD will be used)

#### : Response parameters

Data type	Name	Text	Description
uint16	Value	A	Raw ADC conversion value, 0 – 2047 (0x0 – 0x7FF)
uint32	Uvolts	U	Scaled ADC result in microvolts, 0 – VDD (0x0 – 0x325AA0 if VDD is 3.3V)

#### 7.2.7.2 *gpio\_set\_drive (SIOD, ID=9/5)*

Configure a new drive mode for the selected pin.

#### : Binary header

	Type	Length	Group	ID	Notes
CMD	C0	05	09	05	None.
RSP	C0	02	09	05	None.

#### Text info

Text name	Response length	Category	Notes
SIOD	0x000A	SET	None.

### : Command arguments

Data type	Name	Text	Description
uint8	pin	P*	Pin number (0-47)
Uint16	pin_config	C*	Pin configuration
uint8	pin_out_value	L	Pin out value: 0 - pin will be set to 0 (default) non-zero - pin will be set to 1
uint8	pin_operation	O	Pin operation: 0: immediate or start_up(default) 1: enter low power 2: exit low power 3: register interrupt 4: release this pin from operation list

### : Response parameters

None.

#### Related commands:

[gpio\\_get\\_drive \(GIOD, ID=9/6\)](#)

#### 7.2.7.3 [gpio\\_get\\_drive \(GIOD, ID=9/6\)](#)

Get current new drive mode for the selected pin.

### : Binary header

	Type	Length	Group	ID	Notes
CMD	C0	02	09	06	None.
RSP	C0	06	09	06	None.

### Text info

Text name	Response length	Category	Notes
GIOD	0x0006	GET	None.

### : Command arguments

Data type	Name	Text	Description
uint8	pin	P*	Pin number (0-47)
uint8	pin_operation	O	Pin operation: 0: immediate or start_up(default) 1: enter low power 2: exit low power 3: register interrupt

### : Response parameters

Data type	Name	Text	Description
uint16	Pin_config	C	Pin configuration
uint8	Pin_out_value	L	Pin out value: 0- pin output be set to 0 non-zero - pin output be set to 1

Data type	Name	Text	Description
uint8	Pin_operation	O	Pin operation: 0: immediate or start_up 1: enter low power 2: exit low power 3: register interrupt 4: release this pin from operation list

#### Related Commands:

`gpio_set_drive` (SIOD, ID=9/5)

#### 7.2.7.4 `gpio_set_logic` (SIOL, ID=9/7)

Configure a new output logic for the selected pin.

#### : Binary header

	Type	Length	Group	ID	Notes
CMD	C0	02	09	07	None.
RSP	C0	02	09	07	None.

#### Text info

Text name	Response length	Category	Notes
SIOL	0x000A	SET	None.

#### : Command arguments

Data type	Name	Text	Description
uint8	pin	P*	Pin number (0-47)
uint8	pin_out_value	L	Pin out value: 0- pin output be set to 0 non-zero - pin output be set to 1

#### : Response parameters

None.

#### Related commands:

`gpio_get_logic` (GIOL, ID=9/8)

#### 7.2.7.5 `gpio_get_logic` (GIOL, ID=9/8)

Obtain the current output logic for the selected pin.

#### : Binary header

	Type	Length	Group	ID	Notes
CMD	C0	02	09	08	None.
RSP	C0	0A	09	08	None.

#### Text info

Text name	Response length	Notes
GIOL	0x0020	None.

**: Command arguments**

Data type	Name	Text	Description
uint8	pin	P*	Pin number (0~47 or 0xFF)
uint8	direction	D	Direction for get pint logic if pin is 0 ~47: 0 - get the pin input status (default) 1 - get the pin output status 2 -get the interrupt status Selection for get bit map of pins operation list if pin=0xFF: 0 - pin map (default) 1 - slot map

**: Response parameters:**

Data type	Name	Text	Description
uint32	Pin logic or pin_map_low	L	Pin logic when pin is 0~47 Low 32 bit map when pin is 0xFF
uint32	Pin configure or pin_map_high	H	Pin configuration when pin is 0~47 High 32-bit map when pin is 0xFF

**Related commands:**

[gpio\\_set\\_logic \(SIOL, ID=9/7\)](#)

**7.2.7.6 [gpio\\_set\\_pwm\\_mode \(SPWM, ID=9/11\)](#) (Not implemented)**

Configure new PWM output behavior for selected channel.

EZ-Serial firmware platform for Ezurio Vela IF820 series module provides four dedicated PWM output pins (PWM0/1/2/3). Enabling PWM on the channel means you cannot use that pin for another generic I/O. To return a PWM channel pin to standard functionality, use the [gpio\\_set\\_pwm\\_mode \(SPWM, ID=9/11\)](#) API command to disable PWM output on that pin. See section [GPIO pin map](#) for a pin map table showing pin availability and default assignment.

**Note:** Enabling PWM output automatically prevents the CPU from entering normal sleep under any circumstances. This happens because the high-frequency clock required to generate the PWM signal cannot operate while the CPU is in sleep. To allow normal sleep mode again, you must disable all PWM output. See section [Managing sleep states](#) for further detail.

**: Binary header**

	Type	Length	Group	ID	Notes
CMD	C0	08	09	0B	None.
RSP	C0	02	09	0B	None.

**Text info**

Text name	Response length	Category	Notes
SPWM	0x000A	SET	None.

**: Command arguments**

Data type	Name	Text	Description
uint8	channel	N*	Channel number (0~3)
uint8	enable	E	Enable PWM output (0 to disable, 1 to enable)
uint8	divider	D	Clock divider value (24 MHz input): Minimum = 0 (factory default) Maximum = 255 <b>Note:</b> Divider denominator is divider+1, so "0" is "divide by 1"

Data type	Name	Text	Description
uint8	prescaler	S	PWM prescaler value: 0 = 1x (no prescaling) 1 = 2x 2 = 4x 3 = 8x 4 = 16x 5 = 32x 6 = 64x 7 = 128x <b>Note:</b> Factory default is 0 (1x, no prescaling)
uint16	period	P	Period (0-1023)
uint16	compare	C	Compare (0-1023, must not be greater than period)

**: Response parameters:**

None.

**Related commands**`gpio_get_pwm_mode` (GPWM, ID=9/12)**7.2.7.7 `gpio_get_pwm_mode` (GPWM, ID=9/12) (Not Implemented)**

Obtain current PWM output behavior for selected channel.

See section [GPIO pin map](#) for a pin map table showing pin availability and default assignment.**: Binary header**

	Type	Length	Group	ID	Notes
CMD	C0	01	09	0C	None.
RSP	C0	09	09	0C	None.

**Text info**

Text name	Response length	Category	Notes
GPWM	0x0027	GET	None.

**: Command arguments**

Data type	Name	Text	Description
uint8	channel	N*	Channel number (0~3)

**: Response parameters:**

Data type	Name	Text	Description
uint8	enable	E	Enable PWM output (0 to disable, 1 to enable)
uint8	divider	D	Clock divider value (24 MHz input): Minimum = 0 (factory default) Maximum = 255 Divider denominator is divider+1, so "0" is "divide by 1"
uint8	prescaler	S	PWM prescaler value: 0 = 1x (no prescaling) 1 = 2x

Data type	Name	Text	Description
			2 = 4x
			3 = 8x
			4 = 16x
			5 = 32x
			6 = 64x
			7 = 128x
			Factory default is 0 (1x, no prescaling)
uint16	period	P	Period (0-1023)
uint16	compare	C	Compare (0-1023, must not be greater than period)

#### Related commands

[gpio\\_set\\_pwm\\_mode](#) (SPWM, ID=9/11)



### 7.2.8 CYSPP Group (ID=10)

CYSPP methods relate to the Cypress Serial Port Profile.

Commands within this group are listed below:

```

p_cyspp_start (.CYSPPSTART, ID=10/2)
p_cyspp_set_parameters (.CYSPPSP, ID=10/3)
p_cyspp_get_parameters (.CYSPPGP, ID=10/4)
p_cyspp_set_packetization (.CYSPPSK, ID=10/7)
p_cyspp_get_packetization (.CYSPPGK, ID=10/8)

```

Events within this group are documented in section [CYSPP Group \(ID=10\)](#).

You can find further details and examples concerning CYSPP operation here:

[Section Using CYSPP mode](#)  
[Section Configuring the CYSPP data mode sleep level](#)  
[Section Performing a factory reset](#)

#### 7.2.8.1 *p\_cyspp\_start (.CYSPPSTART, ID=10/2)*

Activate CYSPP operation.

Use this command to start CYSPP via the API protocol, rather than asserting the CYSPP pin or configuring automatic start with the [p\\_cyspp\\_set\\_parameters \(.CYSPPSP, ID=10/3\)](#) API command.

See section [CYSPP state machine](#) for details about how CYSPP moves between different operational states.

#### : Binary header

	Type	Length	Group	ID	Notes
CMD	C0	00	0A	02	None.
RSP	C0	02	0A	02	None.

#### Text info

Text name	Response length	Category	Notes
.CYSPPSTART	0x0011	ACTION	None.

#### : Command arguments

None.

#### Response parameters

None.

#### Related commands

[p\\_cyspp\\_set\\_parameters \(.CYSPPSP, ID=10/3\)](#)

#### Related events

[p\\_cyspp\\_status \(.CYSPP, ID=10/1\)](#)

#### 7.2.8.2 *p\_cyspp\_set\_parameters (.CYSPPSP, ID=10/3)*

Configure new CYSPP behavior settings.

Use this command to control how CYSPP behaves. You can find example usage and practical explanations of how these settings affect behavior in section [Using CYSPP mode](#) and section [Cable replacement examples with CYSPP](#).

**Note:** Disabling CYSPP with this API method causes EZ-Serial firmware platform for Ezurio Vela IF820 series module to hide the relevant GATT database attributes from Client discovery. All other visible attributes remain the same and keep their original handles, but those inside the CYSPP attribute range are hidden and are unusable by connected Clients. This remains in effect until you enable the profile again or assert the CYSPP pin.

### : Binary header

	Type	Length	Group	ID	Notes
CMD	C0	13	0A	03	None.
RSP	C0	02	0A	03	None.

### Text info

Text name	Response length	Category	Notes
.CYSPPSP	0x000E	SET	None.

### : Command arguments

Data type	Name	Text	Description
uint8	enable	E	Enable CYSPP profile: 0 = Disable 1 = Enable 2 = Enable + auto-start (factory default)
uint8	role	G	GAP role to use: 0 = Peripheral/Server (factory default) 1 = Central/Client
uint16	company	C	Company ID value for automatic advertisement payload Manufacturer Data: <b>Note:</b> Factory default is 0x0131 (Cypress Semiconductor)
uint32	local_key	L	Local connection key to present while advertising (peripheral role)
uint32	remote_key	R	Remote connection key to search for while scanning (central role)
uint32	remote_mask	M	Bitmask for bits in remote key which must match for a central-role connection
uint8	sleep_level	P	Maximum sleep level while connected with open CYSPP data pipe: 0 = Sleep disabled 1 = Sleep when possible (factory default) <b>Note:</b> System-wide sleep overrides this if it is set to a lower level
uint8	server_security	S	CYSPP Server security requirement to allow writing CYSPP data from a Client: 0 = Not requires an authenticated link 1 = Requires an authenticated link
uint8	client_flags	F	Client GATT usage flags while operating CYSPP in the central role Bit 0 (0x01) = Use acknowledged data transfers Bit 1 (0x02) = Enable CYSPP RX flow control <b>Note:</b> Factory default is 0x02 (RX flow only)

### : Response parameters:

None.

### Related commands

p\_cyspp\_start (.CYSPPSTART, ID=10/2)  
p\_cyspp\_get\_parameters (.CYSPPGP, ID=10/4)

### Related events

gap\_adv\_state\_changed (ASC, ID=4/2) – May occur if CYSPP is set to start automatically in peripheral role  
p\_cyspp\_status (.CYSPP, ID=10/1)

### Example usage

Section [Using CYSPP mode](#)  
Section [Configuring the CYSPP data mode sleep level](#)

## Section Cable replacement examples with CYSPP

7.2.8.3 *p\_cyspp\_get\_parameters (.CYSPPGP, ID=10/4)*

Obtain current CYSPP behavior settings.

*: Binary header*

	Type	Length	Group	ID	Notes
CMD	C0	01	0A	04	None.
RSP	C0	15	0A	04	None.

**Text info**

Text name	Response length	Category	Notes
.CYSPPGP	0x004F	GET	None.

*: Command arguments*

None.

*: Response parameters*

Data type	Name	Text	Description
uint8	enable	E	Enable CYSPP profile: 0 = Disable 1 = Enable 2 = Enable + auto-start (factory default)
uint8	role	G	GAP role to use: 0 = Peripheral/Server (factory default) 1 = Central/Client
uint16	company	C	Company ID value for automatic advertisement packet payload Manufacturer Data: <b>Note:</b> Factory default is 0x0131 (Cypress Semiconductor)
uint32	local_key	L	Local connection key to present while advertising (peripheral role)
uint32	remote_key	R	Remote connection key to search for while scanning (central role)
uint32	remote_mask	M	Bitmask for bits in remote key which must match for a central-role connection
uint8	sleep_level	P	Maximum sleep level while connected with open CYSPP data pipe: 0 = Sleep disabled 1 = Normal sleep when possible <b>Note:</b> System-wide sleep overrides this if it is set to a lower level
uint8	server_security	S	CYSPP Server security requirement for writing CYSPP data from a Client: 0 = Not requires an authenticated link 1 = Requires an authenticated link
uint8	client_flags	F	Client GATT usage flags while operating CYSPP in the Central role Bit 0 (0x01) = Use acknowledged data transfers Bit 1 (0x02) = Enable CYSPP RX flow control <b>Note:</b> Factory default is 0x02 (RX flow only)

**Related commands**

*p\_cyspp\_set\_parameters (.CYSPPSP, ID=10/3)*

#### 7.2.8.4 `p_cyspp_set_packetization (.CYSPPSK, ID=10/7)`

Control how incoming serial data from an external host is packetized for CYSPP transmission.

Use this command to control whether or how incoming serial data is assembled into specific packets for transmission to the remote peer over a CYSPP connection. Packetization does not affect the content or ordering of serial data in any way, but only affects certain buffering and transmission timing.

---

**Note:** CYSPP packetization does not affect any outgoing UART serial data (module-to-host), nor does it affect incoming serial data while in command mode (that is, the CYSPP data pipe is not open). It impacts only the incoming serial data while CYSPP data mode is active.

---

At 115200 baud, a single byte takes about 80 microseconds to transfer. EZ-Serial firmware platform for Ezurio Vela IF820 series module checks for new bytes at least every 20 microseconds and processes the available bytes. Due to this, a continuous serial byte stream from an external host may be delivered to a remote CYSPP peer with multiple GATT transfers even if all data could fit in a single packet (for instance, two bytes sent as two single-byte transfers). Although the data is always delivered completely and in the correct order, this results in potentially unnecessary complexity on the receiving end, which must buffer and combine incoming data if it does not handle it as a continuous data stream.

To address this behavior, EZ-Serial firmware platform for Ezurio Vela IF820 series module provides this API command to control incoming data packetization. There are five different modes:

##### Mode 0: Immediate

This mode reads and transmits data quickly, always sending as much data as is available when the BLE stack allows a new transmission. In this mode, the first byte or two bytes of a new transmission are usually sent in a single packet even if more data is arriving at the same time.

The [wait] and [length] settings are irrelevant in this mode.

##### Mode 1: Anticipate (factory default with 5 ms wait and 20-byte length)

This mode waits up to [wait] milliseconds in anticipation for at least [length] bytes to arrive from the external host. If the target byte count is reached before the wait time expires, all available bytes are transmitted immediately. If the configured wait time expires before reaching the target byte count, all available bytes are transmitted at that time. Anticipate mode is suitable for most general operations and does not negatively impact the throughput if the incoming serial data arrives fast enough to keep the UART receive buffer full.

The [wait] setting must be between 1 and 255. The [length] setting must be between 1 and 128, which is the internal UART RX software buffer size.

##### Mode 2: Fixed

This mode waits indefinitely until at least [length] bytes have been read, then transmits exactly that many bytes. Fixed mode is best used in cases where the host sends chunks of data which are always of the same size. Setting a [length] value that is greater than the GATT MTU payload size results in multiple transmissions once all data has been buffered. For example, a fixed packet length of 32 bytes with the default GATT MTU size of 23 bytes (usable payload size of 20 bytes) results in one 20-byte packet followed by one 12-byte packet. The MTU depends on the value negotiated by the Client after connection.

The [length] setting must be between 1 and 128, which is the internal UART RX software buffer size. The [wait] setting is irrelevant in this mode.

##### Mode 3: Variable

This mode requires an additional length value from the host before each packet to indicate how many bytes to expect. EZ-Serial firmware platform for Ezurio Vela IF820 series module consumes this byte (it is not transmitted to the remote peer), and then waits until the exact number of bytes have been read before transmitting them. Variable mode is suitable for applications that require packets of differing lengths and can accommodate an extra transmitted byte from the host indicating each packet's length.

For example, the host can send [ **04** 61 62 63 64 ] to transmit the 4-byte ASCII string "abcd" to the remote peer in a single packet. Or, the host can send [ **05** 61 62 63 64 65 **03** 66 67 68 ] to transmit "abcde" in two packets ("abcde" followed by "def").

The prefixed packet length byte must not be greater than 128. Values greater than this will be capped at 128. The [wait] and [length] settings are irrelevant in this mode.

##### Mode 4: End-of-packet

This mode buffers the data until the configured end-of-packet (EOP) byte is encountered in the data stream, or until either the MTU payload size or UART RX buffer has filled. EOP mode allows variable-length packets without knowing in advance the length of the packet

The EOP byte defaults to 0x0D (the carriage return byte, often expressed as '\r' in code). However, you can change it to any value between 0x00 and 0xFF. When the EOP byte occurs in the data stream, all buffered data up to that point including the EOP byte itself will be transmitted to the remote side.

In this mode, EZ-Serial firmware platform for Ezurio Vela IF820 series module will also transmit buffered data under two other conditions:

- If the GATT MTU payload size is less than the UART RX buffer size (128 bytes) and enough data is buffered to fill a single GATT packet, one packet's worth of data is transmitted. The default GATT MTU is 23 bytes with a usable payload size of 20 bytes.
- If the GATT MTU payload size is greater than the UART RX buffer size (128 bytes) and the RX buffer is full, 128 bytes of data are transmitted. This can only occur in cases where the connected client has negotiated a GATT MTU greater than 131 bytes (actual transmit payload is MTU - 3 bytes).

For the "Anticipate" mode (1), you must consider the UART baud rate when choosing the [wait] and [length] values. A 5-ms wait time is suitable for a 20-byte target length at 115200 baud, but this is not enough time to read in 20 bytes at 9600 baud (for example). If you change the baud rate, be sure to choose a [wait] value that allows the target packet length to be filled under normal operating conditions. 0 lists "safe" wait values for 20-byte packets at common baud rates for reference.

#### : Common UART timing for 20-byte packets

Baud Rate	Single Bit Duration	20 Bytes at 8/N/1 (200 Bits)	Safe Wait Value Example
9600	104 us	~21 ms	32 ms (0x20)
38400	26.1 us	~5.2 ms	10 ms (0x0A)
57600	17.4 us	~3.5 ms	5 ms (0x05)
115200	8.68 us	~1.7 ms	5 ms (0x05)
230400	4.34 us	868 us	2 ms (0x02)
460800	2.17 us	434 us	1 ms (0x01)
921600	1.09 us	217 us	1 ms (0x01)

The single-bit duration for any baud rate can be calculated in microseconds using this equation:

$$\text{Bit time} = 1,000,000 \text{ us} / [\text{baud}]$$

Standard UART settings of 8 data bits, no parity, and 1 stop bit yield a total of 10 bits per byte. For a 20-byte packet, this requires allowance for 200 bits.

**Note:** If the packet length used in Anticipate, Fixed, Variable, or End-of-Packet modes exceeds the GATT MTU usable payload size (20 bytes on many platforms), the packets are broken apart to fit within this lower-level constraint. For example, using Fixed mode with [length] set to 32 bytes results in two transmitted packets each time the target length is reached: first a 20-byte packet and then a 12-byte packet.

#### : Binary header

	Type	Length	Group	ID	Notes
CMD	C0	04	0A	07	None.
RSP	C0	02	0A	07	None.

#### Text info

Text name	Response length	Category	Notes
.CYSPPSK	0x000E	SET	None.

#### : Command arguments

Data type	Name	Text	Description
uint8	mode	M	Packetization mode: 0 = Immediate: transmit incoming data as soon as possible 1 = Anticipate: wait a short time to attempt a minimum buffer threshold 2 = Fixed: buffer and send packets of exactly one size 3 = Variable: specify the size of every packet with a prefixed length byte 4 = End-of-packet: transmit data when specific byte occurs in stream

Data type	Name	Text	Description
uint8	wait	W	<p>Anticipation delay (milliseconds), used only in “Anticipate” mode: Minimum = 0x01 (1 millisecond) Maximum = 0x80 (128 bytes)</p> <p><b>Note:</b> Factory default is 1 (Anticipate)</p>
uint8	length	L	<p>Fixed/anticipated packet length (bytes), used only in “Anticipate” or “Fixed” mode: Minimum = 0x01 (1 byte) Maximum = 0x80 (128 bytes)</p> <p><b>Note:</b> Factory default is 0x14 (20 bytes, standard GATT MTU)</p>
uint8	eop	E	<p>End-of-packet byte:</p> <p><b>Note:</b> Factory default is 0x0D (‘\r’ carriage return)</p>

#### : Response parameters

None.

#### Related commands

`p_cyspp_get_packetization (.CYSPPGK, ID=10/8)`

#### 7.2.8.5 `p_cyspp_get_packetization (.CYSPPGK, ID=10/8)`

Obtain current CYSPP packetization settings.

#### : Binary header

	Type	Length	Group	ID	Notes
CMD	C0	00	0A	08	None.
RSP	C0	05	0A	08	None.

#### Text info

Text name	Response length	Category	Notes
.CYSPPGK	0x001D	GET	None.

#### : Command arguments

None.

#### : Response parameters

Data type	Name	Text	Description
uint8	mode	M	<p>Packetization mode:</p> <p>0 = Immediate: Transmit incoming data as soon as possible 1 = Anticipate: Wait a short time to attempt a minimum buffer threshold 2 = Fixed: Buffer and send packets of exactly one size 3 = Variable: Specify the size of every packet with a prefixed length byte 4 = End-of-packet: Transmit data when specific byte occurs in stream</p> <p><b>Note:</b> Factory default is 1 (Anticipate)</p>
uint8	wait	W	<p>Anticipation delay (milliseconds), used only in “Anticipate” mode: Minimum = 0x01 (1 millisecond) Maximum = 0x80 (128 bytes)</p> <p><b>Note:</b> Factory default is 0x5 (5 milliseconds)</p>

Data type	Name	Text	Description
uint8	length	L	Fixed/anticipated packet length (bytes), used only in "Anticipate" and "Fixed" mode: Minimum = 0x01 (1 byte) Maximum = 0x80 (128 bytes) <b>Note:</b> Factory default is 0x14 (20 bytes, standard GATT MTU)
uint8	eop	E	End-of-packet byte: <b>Note:</b> Factory default is 0x0D ('\r' carriage return)

#### Related commands

`p_cyspp_set_packetization` (.CYSPPSK, ID=10/7)

### 7.2.9 BT group (ID=14)

BT methods relate to the BT Classic operation.

Commands within this group are listed below:

```
bt_start_inquiry (/BTI, ID=14/1)
bt_cancel_inquiry (/BTIX, ID=14/2)
bt_query_name (/BTQN, ID=14/3)
bt_connect (/BTC, ID=14/4)
bt_cancel_connection (/BTCX, ID=14/5)
bt_disconnect (/BTDIS, ID=14/6)
bt_query_connections (/BTQC, ID=14/7)
bt_query_peer_address (/BTQPA, ID=14/8)
bt_query_rssi (/BTQSS, ID=14/9)
bt_set_device_class (SBTDC, ID=14/12)
bt_get_device_class (GBTDC, ID=14/13)
```

Events within this group are documented in the following sub sections.

You can find further details and examples concerning SPP operation in section [Bluetooth® classic SPP](#).

#### 7.2.9.1 bt\_start\_inquiry (/BTI, ID=14/1)

Begins the discovery process to identify nearby BT Classic devices.

##### : Binary header

	Type	Length	Group	ID	Notes
CMD	C0	02	0E	01	None.
RSP	C0	02	0E	01	None.

##### Text info

Text name	Response length	Notes
/BTI	0x000A	None.

##### : Command arguments

Data type	Name	Text	Description
uint8	duration	D*	Inquiry duration in seconds: 3 – 30 seconds
uint8	flags	F*	Flags 0 – Inquiry all (name and address) 1 – Inquiry name 2 – Inquiry address

##### Response parameters

None.

##### Command-specific result codes

None.

##### Related commands

[bt\\_cancel\\_inquiry \(/BTIX, ID=14/2\)](#)

##### Related events

```
bt_inquiry_result (BTIR, ID=14/1)
bt_name_result (BTINR, ID=14/2)
bt_inquiry_complete (BTIC, ID=14/3)
```



**7.2.9.2** *bt\_cancel\_inquiry (/BTIX, ID=14/2)*

Cancels any ongoing BT Classic inquiry process before it would normally end.

**: Binary header**

	Type	Length	Group	ID	Notes
CMD	C0	00	0E	02	None.
RSP	C0	02	0E	02	None.

**Text info**

Text name	Response length	Notes
/BTIX	0x000B	None.

**Command arguments**

None.

**Response Parameters**

None.

**Command-specific result codes**

None.

**Related commands**

*bt\_start\_inquiry (/BTI, ID=14/1).*

**Related events**

*bt\_inquiry\_complete (BTIC, ID=14/3)*

**7.2.9.3** *bt\_query\_name (/BTQN, ID=14/3)*

Attempt to obtain a friendly name for a remote device.

**: Binary header**

	Type	Length	Group	ID	Notes
CMD	C0	06	0E	03	None.
RSP	C0	02	0E	03	None.

**Text info**

Text name	Response length	Notes
/BTQN	0x000B	None.

**: Command arguments**

Data type	Name	Text	Description
macaddr	address	A*	Bluetooth® address

**Response parameters**

None.

**Command-specific result codes**

None.

**Related commands**

None.

**Related events**

*bt\_name\_result (BTINR, ID=14/2)*

**7.2.9.4** *bt\_connect (/BTC, ID=14/4)*

Opens a connection to a remote BT Classic target device.

**: Binary header**

	Type	Length	Group	ID	Notes
CMD	C0	07	0E	04	None.
RSP	C0	03	0E	04	None.

**Text info**

Text name	Response length	Notes
/BTC	0x000F	None.

**: Command arguments**

Data type	Name	Text	Description
macaddr	address	A*	Bluetooth® address
uint8	type	T*	Type: 1: SPP

**: Command arguments**

Data type	Name	Text	Description
uint8	conn_handle	C	Handle assigned to new pending connection

**Command-specific result codes**

None.

**Related commands**

*bt\_connected (BTCON, ID=14/4)*  
*bt\_connection\_failed (BTCF, ID=14/6).*  
*bt\_cancel\_connection (/BTCX, ID=14/5)*  
*bt\_cancel\_connection (/BTCX, ID=14/5)*  
*bt\_disconnect (/BTDIS, ID=14/6)*

**Related events**

*bt\_connected (BTCON, ID=14/4)*  
*bt\_connection\_failed (BTCF, ID=14/6).*

#### 7.2.9.5 *bt\_cancel\_connection (/BTCX, ID=14/5) (Not implemented)*

Cancels a pending connection attempt to a remote BT Classic peer device, previously initiated with the 'connect' command.

**Note:** This command should be used only to terminate a pending connection attempt, not to close an open connection. To close an existing connection that has already been established, use the 'disconnect' command instead.

#### : Binary header

	Type	Length	Group	ID	Notes
CMD	C0	00	0E	05	None.
RSP	C0	02	0E	05	None.

#### Text info

Text name	Response length	Notes
/BTCX	0x000B	None.

#### Command arguments

None.

#### Response parameters

None.

#### Command-specific result codes

None.

#### Related commands

*bt\_connect (/BTC, ID=14/4)*  
*bt\_disconnect (/BTDIS, ID=14/6)*

#### Related events

*bt\_connected (BTCON, ID=14/4)*

**7.2.9.6** *bt\_disconnect (/BTDIS, ID=14/6)*

Closes an open BT Classic connection to a remote device, previously initiated with the 'connect' command. If optional connection handle argument is omitted, all open connections will be closed.

**Note:** This command should be used only to close an open connection, not to terminate a pending connection attempt. To cancel a pending connection attempt that has not yet succeeded, use the 'cancel connection' command instead.

**: Binary header**

	Type	Length	Group	ID	Notes
CMD	C0	01	0E	06	None.
RSP	C0	02	0E	06	None.

**Text info**

Text name	Response length	Notes
/BTDIS	0x000C	None.

**: Command arguments**

Data type	Name	Text	Description
uint8	conn_handle	C*	Handle of connection to disconnect

**Response parameters**

None.

**Command-specific result codes**

None.

**Related commands**

*bt\_connect (/BTC, ID=14/4)*

**Related events**

*bt\_disconnected (BTDIS, ID=14/7)*

**7.2.9.7** *bt\_query\_connections (/BTQC, ID=14/7)*

Used to query the current list of active connections.

**: Binary header**

	Type	Length	Group	ID	Notes
CMD	C0	00	0E	07	None.
RSP	C0	03	0E	07	None.

**Text info**

Text name	Response length	Notes
/BTQC	0x0010	None.

**: Command arguments**

None.

**: Response parameters**

Data type	Name	Text	Description
uint8	count	C	Count of all active connections

**Command-specific result codes**

None.

**Related commands**

None.

**Related events:**

*bt\_connection\_status* (BTCS, ID=14/5)

**7.2.9.8** *bt\_query\_peer\_address (/BTQPA, ID=14/8)*

Used to query the Bluetooth® address of a currently connected BT Classic remote peer. This command will generate an error response if it is used without an active connection.

**: Binary header**

	Type	Length	Group	ID	Notes
CMD	C0	01	0E	08	None.
RSP	C0	09	0E	08	None.

**Text info**

Text name	Response length	Notes
/BTQPA	0x0020	None.

**: Command arguments**

Data type	Name	Text	Description
uint8	conn_handle	C	Handle of connection for which to query remote peer address

**: Response parameters**

Data type	Name	Text	Description
macaddr	address	A	Peer Bluetooth® address
uint8	address_type	T	Address type (Always 0 in current implementation)

#### Command-specific result codes

None.

#### Related commands

`bt_connect (/BTC, ID=14/4)`

#### Related events

`bt_connected (BTCON, ID=14/4)`

#### 7.2.9.9 `bt_query_rssi (/BTQSS, ID=14/9)`

Used to query the remote signal strength indication (RSSI) value detected in the packet received most recently from the currently connected remote BT Classic peer. This command will generate an error response if it is used without an active connection. The RSSI value returned in the response is expressed as a signed 8-bit integer. In text mode, it will appear in two's complement form. Positive numbers in this form fall in the range [0, 127] and are as they appear. Negative numbers fall in the range [128, 255] and should have 256 subtracted from them to obtain the real value.

Examples:

0x03 = +3 dBm  
0xFF = -1 dBm (0xFF = 255 - 256 = -1)  
0xF0 = -16 dBm (0xF0 = 240 - 256 = -16)  
0xC5 = -59 dBm (0xC5 = 197 - 256 = -59)

#### : Binary header

	Type	Length	Group	ID	Notes
CMD	C0	01	0E	09	None.
RSP	C0	03	0E	09	None.

#### Text info

Text name	Response length	Notes
/BTQSS	0x0011	None.

#### : Command arguments

Data type	Name	Text	Description
macaddr	address	A	The mac address for which to query signal strength

#### : Response parameters

Data type	Name	Text	Description
int8	Rssi	R	RSSI value

#### Command-specific result codes

None.

#### Related commands:

None.

#### Related events:

None.

### 7.2.9.10 *bt\_set\_parameters* (SBTP, ID=14/10)

Sets BT Classic device behavior.

#### : Binary header

	Type	Length	Group	ID	Notes
CMD	C0	0A	0E	0A	None.
RSP	C0	02	0E	0A	None.

#### Text info

Text name	Response length	Notes
SBTP	0x000A	None.

#### : Command arguments

Data type	Name	Text	Description
uint16	link_super_time_out	T	BT Classic link super time out, unit is 0.625 ms Factory default is 0x7D00 (20 second)
uint8	discoverable	D	BT Classic discoverable mode: 0 Not discoverable 1 Limited BT Classic discoverable 2 General BT Classic discoverable
uint8	connectable	C	BT Classic connectable mode
uint8	flags	F	BT Classic behavior flags (always set to 0 in the current release)
uint8	scn	S	Service Channel Number for SPP server Factory default is 2
uint16	active_bt_discoverability	V	Active time for BT classic discoverable, unit is second. Factory default is 0, means always active
uint16	active_bt_connectability	N	Active time for BT classic connectable, unit is second. Factory default is 0, means always active

#### Response parameters

None.

#### Command-specific result codes

None.

#### Related commands

None.

#### Related events

None.

### 7.2.9.11 *bt\_get\_parameters* (GBTP, ID=14/11)

Used to get the current BT Classic configuration.

#### : Binary header

	Type	Length	Group	ID	Notes
CMD	C0	00	0E	0B	None.
RSP	C0	0C	0E	0B	None.

#### Text info

Text name	Response length	Notes
GBTP	0x0033	None.

#### : Command arguments

None.

#### Response parameters

Data type	Name	Text	Description
uint16	link_super_time_out	T	BT Classic link super time out, unit is 0.625 ms Factory default is 0x7D00 (20 second)
uint8	discoverable	D	BT Classic discoverable mode: 0 Not discoverable 1 Limited BT Classic discoverable 2 General BT Classic discoverable
uint8	connectable	C	BT Classic connectable mode: 0 Not connectable 1 BT Classic connectable
uint8	flags	F	BT Classic behavior flags (always set to 0 in the current release)
uint8	scn	S	Service Channel Number for SPP server <b>Note:</b> Factory default is 2
uint16	active_bt_discoverability	V	Active time for BT classic discoverable, unit is second. <b>Note:</b> Factory default is 0, means always active
uint16	active_bt_connectability	N	Active time for BT classic connectable, unit is second. <b>Note:</b> Factory default is 0, means always active

#### Command-specific result codes

None.

#### Related commands

None.

#### Related events

None.

#### 7.2.9.12 bt\_set\_device\_class (SBTDC, ID=14/12)

Defines the device class value. This is a 24-bit integer value with flag bits defined by the Bluetooth® SIG, reported to remote peers during an inquiry process.

#### : Binary header

	Type	Length	Group	ID	Notes
CMD	C0	04	0E	0C	None.
RSP	C0	02	0E	0C	None.

#### Text info

Text name	Response length	Notes
SBTDC	0x000B	None.



**: Command arguments**

Data type	Name	Text	Description
uint32	cod	C	New device appearance value

**Response parameters**

None.

**Command-specific result codes**

None.

**Related commands**

[bt\\_get\\_device\\_class \(GBTDC, ID=14/13\)](#)

**Related events**

None.

**7.2.9.13 [bt\\_get\\_device\\_class \(GBTDC, ID=14/13\)](#)**

Used to get the current device class value.

**: Binary header**

	Type	Length	Group	ID	Notes
CMD	C0	00	0E	0D	None.
RSP	C0	06	0E	0D	None.

**Text info**

Text name	Response length	Notes
GBTDC	0x0016	None.

**: Command arguments**

None.

**: Response parameters**

Data type	Name	Text	Description
uint32	cod	C	Current device class value

**Command-specific result codes**

None.

**Related commands**

[bt\\_set\\_device\\_class \(SBTDC, ID=14/12\)](#)

**Related events**

None.

**7.2.10 [Spp group \(ID=19\)](#)**

### 7.2.10.1 *spp\_send\_command (.SPPS, ID=19/1) (not implemented)*

Sends data via an SPP connection.

#### *: Binary header*

	Type	Length	Group	ID	Notes
CMD	C0	03+	13	01	Variable-length command payload, value specified is minimum.
RSP	C0	02	13	01	None.

#### Text info

Text name	Response length	Notes
.SPPS	0x000B	None.

#### *: Command arguments*

Data type	Name	Text	Description
uint8	conn_handle	C*	Connection handle
longuint8a	data	D*	New data to send  <b>Note:</b> The longuint8a data type requires two prefixed "length" bytes before binary the parameter payload. In the current implementation, the length is 512 in MAX due to resource limitation.

#### Response parameters

None.

#### Command-specific result codes

None.

#### Related commands

None.

#### Related events

*SPP\_data\_received (SPPD, ID=19/1)*

**7.2.10.2 spp\_set\_config (.SPPSC, ID=19/2) (Not implemented)**

Set SPP connection configuration.

**: Binary header**

	Type	Length	Group	ID	Notes
CMD	C0	01	13	02	None.
RSP	C0	02	13	02	None.

**Text info**

Text name	Response length	Notes
.SPPSC	0x000C	None.

**: Command arguments**

Data type	Name	Text	Description
uint8	connections	N	Support number of connection (1~7) <b>Note:</b> HID device FW only support 1 due to ram limitation

**Response parameters**

None.

**Command-specific result codes**

None.

**Related commands**

**spp\_get\_config (.SPPGC, ID=19/3)**

**Related events**

None.

**7.2.10.3 spp\_get\_config (.SPPGC, ID=19/3)**

Get SPP connection configuration.

**: Binary header**

	Type	Length	Group	ID	Notes
CMD	C0	00	13	03	None.
RSP	C0	03	13	03	None.

**Text info**

Text name	Response length	Notes
.SPPGC	0x0011	None.

**: Command arguments**

None.

**: Response parameters**

Data type	Name	Text	Description
uint8	connections	N	Current SPP support connection number

**Command-specific result codes**

None.

**Related commands**

**spp\_set\_config (.SPPSC, ID=19/2)**

**Related events**

None.

## 7.3 API events

All events implemented in the API protocol are described in detail below. API commands and responses are documented separately in section [API commands and responses](#).

A master list of all possible error codes appearing in certain events can be found in section [Error codes](#).

Commands and responses are broken down into the following groups:

- System Group (ID=2)
- GAP Group (ID=4)
- GATT Server Group (ID=5)
- GATT Client Group (ID=6)
- SMP Group (ID=7)
- GPIO Group (ID=9)
- Bluetooth® Classic Group (ID=14)

### 7.3.1 System Group (ID=2)

System methods relate to the core device, describing things like boot and device address info, and resetting to an initial state.

Events within this group are listed below:

- system\_boot (BOOT, ID=2/1)
- system\_error (ERR, ID=2/2)
- system\_factory\_reset\_complete (RFAC, ID=2/3)
- system\_dump\_blob (DBLOB, ID=2/5)

Commands within this group are documented in section [System group \(ID=2\)](#).

#### 7.3.1.1 system\_boot (BOOT, ID=2/1)

EZ-Serial firmware platform for Ezurio Vela IF820 series module module has booted and is ready to process commands.

##### : Binary header

Type	Length	Group	ID	Notes
80	13	02	01	None.

##### Text info

Text name	Event length	Notes
BOOT	0x003E+	None.

##### : Event parameters

Data type	Name	Text	Description
uint32	app	E	Application version number
uint32	stack	S	BLE stack version number
uint16	protocol	P	API protocol version number
uint8	hardware	H	F1 The hardware platform is CYW20820
uint8	cause	C	Cause of boot event: always 0
macaddr	address	A	Bluetooth® address
uint8a	FW	F	FW description: To release FW image, having model info, FW building data and time

##### Related commands

- system\_reboot (/RBT, ID=2/2)
- system\_factory\_reset (/RFAC, ID=2/5)

### 7.3.1.2 *system\_error (ERR, ID=2/2)*

System error has occurred.

This may be triggered by a malformed command, an operation that failed or could start due to an invalid operational state, or a low-level hardware failure. See section [Error codes](#) for a list of all possible errors.

#### : Binary header

Type	Length	Group	ID	Notes
80	02	02	02	None.

#### Text info

Text name	Event length	Notes
ERR	0x000B	None.

#### : Event parameters

Data type	Name	Text	Description
uint16	error	E	Error code describing what went wrong

### 7.3.1.3 *system\_factory\_reset\_complete (RFAC, ID=2/3)*

Factory reset is complete.

This event will occur after sending the [system\\_factory\\_reset \(/RFAC, ID=2/5\)](#) API command, or asserting (LOW) the FACTORY\_TR and CYSPP pins at boot time. EZ-Serial firmware platform for Ezurio Vela IF820 series module transmits this event using the originally configured host interface settings (if different from the default). After generating this event, the module reboots immediately and the default settings take effect.

**Note:** If you triggered a factory reset using the GPIO method at boot time, the final reboot back into an operational state occurs only after you de-assert one or both the pins. This safeguard prevents an endless loop of factory resets if both pins remain asserted.

#### : Binary header

Type	Length	Group	ID	Notes
80	00	02	03	None.

#### Text info

Text name	Event length	Notes
RFAC	0x0005	None.

#### Event parameters

None.

#### Related commands

[system\\_factory\\_reset \(/RFAC, ID=2/5\)](#)

### 7.3.1.4 *system\_dump\_blob (DBLOB, ID=2/5)*

Single data blob of requested configuration type or system state.

#### : Binary header

Type	Length	Group	ID	Notes
80	04-14	02	05	Variable-length event payload, minimum of 4 (0x04), maximum of 20 (0x14).

#### Text info

Text name	Event length	Notes
DBLOB	0x0015-0x0035	Variable-length event payload, minimum of 21 (0x15), maximum of 53 (0x35)

### : Event parameters

Data type	Name	Text	Description
uint8	type	T	Type of information being dumped: 0 = Runtime configuration data 1 = Boot-level configuration data 2 = Factory-level configuration data 3 = System state data
uint16	offset	O	Blob start offset
uint8a	data	D	Dumped blob of data <b>Note:</b> uint8a data type requires one prefixed "length" byte before binary parameter payload

### Related commands

`system_dump (/DUMP, ID=2/3)`

### 7.3.2 GAP Group (ID=4)

GAP methods relate to the Generic Access Protocol layer of the Bluetooth® stack, which includes management of scanning, advertising, connection establishment, and connection maintenance.

Events within this group are listed below:

- `gap_whitelist_entry` (WL, ID=4/1)
- `gap_adv_state_changed` (ASC, ID=4/2)
- `gap_scan_state_changed` (SSC, ID=4/3)
- `gap_connected` (C, ID=4/5)
- `gap_disconnected` (DIS, ID=4/6)
- `gap_connection_updated` (CU, ID=4/8)

Commands within this group are documented in section [GAP Group \(ID=4\)](#).

#### 7.3.2.1 `gap_whitelist_entry` (WL, ID=4/1)

Details about a single entry in the whitelist table.

##### : Binary header

Type	Length	Group	ID	Notes
80	07	04	01	None.

##### Text info

Text name	Event length	Notes
WL	0x0017	None.

##### : Event parameters

Data type	Name	Text	Description
macaddr	address	A	Bluetooth® address
uint8	type	T	Address type: 0 = Public 1 = Random/private

##### Related commands

`gap_add_whitelist_entry` (/WLA, ID=4/6)  
`gap_query_whitelist` (/QWL, ID=4/14)

#### 7.3.2.2 `gap_adv_state_changed` (ASC, ID=4/2)

Indicates that the module has started or stopped advertising, due to a scheduled timeout, automated process, or intentional action.

##### : Binary header

Type	Length	Group	ID	Notes
80	02	04	02	None.

##### Text info

Text name	Event length	Notes
ASC	0x000E	None.

#### : Event parameters

Data type	Name	Text	Description
uint8	state	S	Advertising state: 0 = Stop advertising 1 = Directed advertisement (high duty cycle) 2 = Directed advertisement (low duty cycle) 3 = Undirected advertisement (high duty cycle) 4 = Undirected advertisement (low duty cycle) 5 = Non-connectable advertisement (high duty cycle) 6 = Non-connectable advertisement (low duty cycle) 7 = discoverable advertisement (high duty cycle)
uint8	reason	R	Reason for state change: 0 = User command 1 = GAP automatic advertisement enabled 2 = Configured timeout expired 3 = CYSPP operation state change 6 = Disconnection

#### Related commands

gap\_start\_adv (/A, ID=4/8)  
gap\_stop\_adv (/AX, ID=4/9)  
gap\_set\_adv\_parameters (SAP, ID=4/23)  
p\_cyspp\_start (.CYSPPSTART, ID=10/2)  
p\_cyspp\_set\_parameters (.CYSPPSP, ID=10/3)

#### 7.3.2.3 gap\_scan\_state\_changed (SSC, ID=4/3)

Indicates that the module has started or stopped scanning, due to a scheduled timeout or intentional action.

#### : Binary header

Type	Length	Group	ID	Notes
80	02	04	03	None.

#### Text info

Text name	Event length	Notes
SSC	0x00E	None.

#### : Event parameters

Data type	Name	Text	Description
uint8	state	S	Scanning state 0 = Stopped 1 = High Duty Scan 2 = Low Duty scan
Unit8	Reason	R	Reason for state change 0 = User command 1 = NOT USED 2 = Configured timeout expired 3 = CYSPP operation state change



#### Related Commands:

gap\_start\_scan (/S, ID=4/10)  
gap\_stop\_scan (/SX, ID=4/11)

#### Related Events:

None.

#### 7.3.2.4 gap\_scan\_result (S, ID=4/4)

Details of an advertisement or scan response packet.

This event occurs while scanning for remote devices. If you have enabled active scanning, most peripherals will provide two separate packets delivered via this API: one advertisement packet and one scan response packet. Passive scanning will result in only the first of those two. Scan response packets typically contain less critical data, such as the friendly name of the device, or its transmit power.

#### : Binary header

Type	Length	Group	ID	Notes
80	0B-2A	04	04	Variable-length event payload, minimum of 11 (0x0B), maximum of 42 (0x2A)

#### Text info

Text name	Event length	Notes
S	0x0028-0x0047	Variable-length event payload, minimum of 40 (0x28), maximum of 71 (0x47)

#### : Event parameters

Data type	Name	Text	Description
uint8	result_type	R	Scan result type: 0 = Connectable undirected advertisement packet 1 = Connectable directed advertisement packet 2 = Scannable undirected advertisement packet 3 = Non-connectable undirected advertisement packet 4 = Scan response packet
macaddr	address	A	Bluetooth® address
uint8	address_type	T	Address type: 0 = Public 1 = Random/private
int8	rss	S	RSSI
uint8	bond	B	Bond entry (0 for no bond)
uint8a	data	D	Advertisement payload data (0-31 bytes) <b>Note:</b> uint8a data type requires one prefixed "length" byte before binary parameter payload

#### Related commands

gap\_connect (/C, ID=4/1)  
gap\_start\_scan (/S, ID=4/10)  
gap\_stop\_scan (/SX, ID=4/11)  
gap\_set\_scan\_parameters (SSP, ID=4/25)

#### Example usage

See section [How to scan](#)

**7.3.2.5** *gap\_connected (C, ID=4/5)*

Connection established with a remote device.

**: Binary header**

Type	Length	Group	ID	Notes
80	0F	04	05	None.

**Text info**

Text name	Event length	Notes
C	0x0035	None.

**: Event parameters**

Data type	Name	Text	Description
uint8	conn_handle	C	Connection handle
macaddr	address	A	Bluetooth® address
uint8	type	T	Address type: 0 = Public 1 = Random/private
uint16	interval	I	Connection interval
uint16	slave_latency	L	Slave latency
uint16	supervision_timeout	O	Supervision timeout
uint8	bond	B	Bond entry (0 for no bond)

**Related commands**

*gap\_disconnect (/DIS, ID=4/5)*

**Related events**

*gap\_disconnected (DIS, ID=4/6)*

**7.3.2.6** *gap\_disconnected (DIS, ID=4/6)*

Connection to a remote device has closed.

For a list of possible disconnection reasons, see the 0x900 range of codes in section [EZ-Serial firmware platform for Vela IF820 system error codes](#). These are the most common reasons:

- 0x0908 – Page timeout (unexpected loss of connectivity, no response within supervision timeout)
- 0x0913 – Remote user terminated connection (cleanly closed remotely)
- 0x0916 – Connection terminated by local host (cleanly closed locally)
- 0x093E – Connection failed to be established (connection initiated locally, but peer did not respond to request)

**: Binary header**

Type	Length	Group	ID	Notes
80	03	04	06	None.

**Text info**

Text name	Event length	Notes
DIS	0x0010	None.

**: Event parameters**

Data type	Name	Text	Description
uint8	conn_handle	C	Connection handle
uint16	reason	R	Reason for disconnection

**Related commands**

*gap\_disconnect (/DIS, ID=4/5)*

### 7.3.2.7 *gap\_connection\_updated* (CU, ID=4/8)

Active connection has negotiated and applied new parameters.

This event occurs on the slave side after a master requests new parameters or accepts the new parameters requested by the slave. It also occurs on the master side after a slave requests new parameters and the master accepts the request.

**Note:** A rejected connection update request sent from a slave does not result in any events indicating the rejection. The slave must assume the original parameters are in effect until after it receives this API event.

#### *: Binary header*

Type	Length	Group	ID	Notes
80	07	04	08	None.

#### **Text info**

Text name	Event length	Notes
CU	0x001D	None.

#### *: Event parameters*

Data type	Name	Text	Description
uint8	conn_handle	C	Connection handle
uint16	interval	I	Connection interval
uint16	slave_latency	L	Slave latency
uint16	supervision_timeout	O	Supervision timeout

#### **Related commands**

*gap\_update\_conn\_parameters* (/UCP, ID=4/3)

### 7.3.3 GATT Server Group (ID=5)

GATT Server methods relate to the server role of the Generic Attribute Protocol layer of the Bluetooth® stack. These methods are used for working with the local GATT structure.

Events within this group are listed below:

gatts\_discover\_result (DL, ID=5/1)  
gatts\_data\_written (W, ID=5/2)  
gatts\_indication\_confirmed (IC, ID=5/3)  
gatts\_db\_entry\_blob (DGATT, ID=5/4)

Commands within this group are documented in section GATT Server Group (ID=5).

#### 7.3.3.1 gatts\_discover\_result (DL, ID=5/1)

Details about a single entry in the local GATT database.

This event occurs while discovering local services, characteristics, or descriptors.

##### : Binary header

Type	Length	Group	ID	Notes
80	08+	05	01	Variable-length event payload, value specified is minimum.

##### Text info

Text name	Event length	Notes
DL	0x0020+	Variable-length event payload, value specified is minimum.

##### : Event parameters

Data type	Name	Text	Description
uint16	attr_handle	H	Attribute handle
uint16	attr_handle_rel	R	Related attribute handle: If discovering services, the end handle for the service group If discovering characteristics, the value handle that holds the application data If discovering descriptors, always 0 (not applicable)
uint16	type	T	Attribute type: 0x2800 = Primary Service Declaration 0x2801 = Secondary Service Declaration 0x2802 = Include Declaration 0x2803 = Characteristic Declaration 0x2900 = Characteristic Extended Properties Descriptor 0x2901 = Characteristic User Description Descriptor 0x2902 = Client Characteristic Configuration Descriptor 0x2903 = Server Characteristic Configuration Descriptor 0x2904 = Characteristic Format Descriptor 0x2905 = Characteristic Aggregate Format Descriptor 0x0000 = Characteristic value attribute or user-defined structure (see UUID)
uint8	properties	P	Characteristic properties bitmask, only non-zero during characteristic discovery: Bit 0 (0x01) = Broadcast Bit 1 (0x02) = Read Bit 2 (0x04) = Write without response Bit 3 (0x08) = Write Bit 4 (0x10) = Notify

Data type	Name	Text	Description
			Bit 5 (0x20) = Indicate Bit 6 (0x40) = Signed write Bit 7 (0x80) = Extended properties (will have 0x2900 descriptor)
uint8a	uuid	U	UUID  <b>Note:</b> uint8a data type requires one prefixed "length" byte before binary parameter payload.

#### Related commands

[gatts\\_discover\\_services \(/DLS, ID=5/6\)](#)  
[gatts\\_discover\\_characteristics \(/DLC, ID=5/7\)](#)  
[gatts\\_discover\\_descriptors \(/DLD, ID=5/8\)](#)

#### 7.3.3.2 *gatts\_data\_written (W, ID=5/2)*

Remote GATT Client has written data to a local attribute.

A connected remote client can write data to a local attribute using either acknowledged unacknowledged write operations Acknowledged writes require two full connection intervals to complete: one for the data transfer from client to server, and one for the acknowledgement back from server to client. Unacknowledged writes may occur multiple times within the same connection interval, and therefore provide greater throughput potential.

EZ-Serial firmware platform for Ezurio Vela IF820 series module automatically responds to acknowledged writes except in two cases:

You have disabled automatic responses using the [gatts\\_set\\_parameters \(SGSP, ID=5/14\)](#) API command.

The attribute written to has the "User data management" bit set in its properties value, set during creation with the [gatts\\_create\\_attr \(/CAC, ID=5/1\)](#) API command.

#### : Binary header

Type	Length	Group	ID	Notes
80	06	05	02	Variable-length event payload, value specified is minimum.

#### Text info

Text name	Event length	Notes
W	0x0016+	Variable-length event payload, value specified is minimum.

#### : Event parameters

Data type	Name	Text	Description
uint8	conn_handle	C	Handle of connection from which write came
uint16	attr_handle	H	Attribute handle
uint8	type	T	Write type: 0x00 = Simple write – acknowledged 0x01 = Write without response – unacknowledged 0x80 = Simple write requiring manual response via API command
longuint8a	data	D	Written data  <b>Note:</b> longuint8a data type requires two prefixed "length" bytes before binary parameter payload.

#### Related commands

None.

**7.3.3.3** *gatts\_indication\_confirmed (IC, ID=5/3)*

Remote GATT Client has confirmed receipt of indicated data.

This event occurs after a client receives and confirms data pushed using the *gatts\_indicate\_handle* (/IH, ID=5/12) API command.

**: Binary header**

Type	Length	Group	ID	Notes
80	03	05	03	None.

**Text info**

Text name	Event length	Notes
IC	0x000F	None.

**: Event parameters**

Data type	Name	Text	Description
uint8	conn_handle	C	Handle of connection from which confirmation came
uint16	attr_handle	H	Attribute handle use for indication

**Related commands**

*gatts\_indicate\_handle* (/IH, ID=5/12)

**Related events**

None.

**7.3.3.4** *gatts\_db\_entry\_blob (DGATT, ID=5/4)*

Single entry from the GATT structure definition.

This event presents local dynamic GATT attribute definition in a format which simplifies reentry using the *gatts\_create\_attr* (/CAC, ID=5/1) API command. For details about the data provided in this event, see section **Defining custom local GATT services and characteristics**.

**Note:** This event includes the attribute handle and the absolute group end value, neither of which is part of the data entered when creating a new custom attribute. Be sure to remove the handle and absolute group end if you are directly copying the content from these output lines into new commands manually.

**: Binary header**

Type	Length	Group	ID	Notes
80	10-20	05	04	Variable-length event payload, minimum of 16 (0x10), maximum of 32 (0x20)

**Text info**

Text name	Event length	Notes
DGATT	0x0037-0x0057	Variable-length event payload, minimum of 55 (0x37), maximum of 87 (0x57)

**: Event parameters**

Data type	Name	Text	Description
uint16	handle	H	Attribute handle (0x0001 – 0xFFFF)
uint16	type	T*	0 = structure 1 = characteristic value
uint8	perm	R*	Permission bits: Bit 0 (0x01) = Variable length Bit 1 (0x02) = Readable Bit 2 (0x04) = Write command (unacknowledged)

Data type	Name	Text	Description
			Bit 3 (0x08) = Write request (acknowledged) Bit 4 (0x10) = Perm auth readable Bit 5 (0x20) = Reliable write (includes prepared write) Bit 6 (0x40) = Authenticated writable Bit 7 (0x80) = UUID is 128 bits
uint16	length	L	Indicates how many bytes of RAM are allocated for the definition (structure) or content (characteristic value)
longuint8a	data	U	Data (UUID or default attribute value where applicable) longuint8a data type requires two prefixed “length” bytes before binary parameter payload.

#### Related commands

[gatts\\_dump\\_db \(/DGDB, ID=5/5\)](#)

### 7.3.4 GATT Client Group (ID=6)

GATT Client methods relate to the client role of the GATT layer of the Bluetooth® stack. These methods are used for working with the GATT structures on remote devices, and can only be used while a device is connected.

Events within this group are listed below:

gattc\_discover\_result (DR, ID=6/1)  
 gattc\_remote\_procedure\_complete (RPC, ID=6/2)  
 gattc\_data\_received (D, ID=6/3)  
 gattc\_write\_response (WRR, ID=6/4)

Commands within this group are documented in section [GATT Client Group \(ID=6\)](#).

#### 7.3.4.1 gattc\_discover\_result (DR, ID=6/1)

Details of a single entry in the remote GATT database. This event occurs while you are discovering remote services, characteristics, or descriptors.

##### : Binary header

Type	Length	Group	ID	Notes
80	09-19	06	01	Variable-length event payload, minimum of 9 (0x09), maximum of 25 (0x19)

##### Text info

Text name	Event length	Notes
DR	0x0025-0x0044	Variable-length event payload, minimum of 37 (0x25), maximum of 69 (0x45)

##### : Event parameters

Data type	Name	Text	Description
uint8	conn_handle	C	Connection handle
uint16	attr_handle	H	Attribute handle
uint16	attr_handle_rel	R	Related attribute handle: If discovering services, it is the end handle for the service group If discovering included services, it is the start handle of the include service. The value of the properties field is reused as attr_count, which contains the number of attributes that are counted within the included service. (Not implemented) If discovering characteristics, it is the value handle of the characteristic value attribute handle If discovering descriptors, always 0 (not applicable)
uint16	type	T	GATT discover type: 1 = Discovery all service 2 = Discovery service by UUID 3 = Discovery an included service within a service 4 = Discovery characteristics of a service with/without type requirement 5 = Discovery characteristics descriptors of a characteristic.
uint8	properties	P	Characteristic properties bitmask, only non-zero during characteristic discovery: Bit 0 (0x01) = Broadcast Bit 1 (0x02) = Read Bit 2 (0x04) = Write without response Bit 3 (0x08) = Write Bit 4 (0x10) = Notify Bit 5 (0x20) = Indicate Bit 6 (0x40) = Signed write Bit 7 (0x80) = Extended properties (will have 0x2900 descriptor)



Data type	Name	Text	Description
uint8a	uuid	U	UUID (16-bit, 32-bit, or 128-bit)  <b>Note:</b> uint8a data type requires one prefixed "length" byte before binary parameter payload

**Related commands**

`gattc_discover_services (/DRS, ID=6/1)`  
`gattc_discover_characteristics (/DRC, ID=6/2)`  
`gattc_discover_descriptors (/DRD, ID=6/3)`

**Related events**

`gattc_remote_procedure_complete (RPC, ID=6/2)`

**Example usage**

See section [How to discover a remote server's GATT structure](#).

### 7.3.4.2 *gattc\_remote\_procedure\_complete* (RPC, ID=6/2)

Remote GATT Client operation has completed.

This event occurs after requesting a GATT Cclient operation that may require an unknown length of time or quantity of returned results before it is finished, such as a remote GATT descriptor discovery. Because you cannot perform multiple GATT Client operations simultaneously, your application logic must wait for this event and continue with additional client operations only after the event occurs.

See the Related Commands list below for specific commands which trigger this event.

#### : Binary header

Type	Length	Group	ID	Notes
80	03	06	02	None.

#### Text info

Text name	Event length	Notes
RPC	0x000D	None.

#### : Event parameters

Data type	Name	Text	Description
uint8	conn_handle	C	Connection handle
uint16	result	R	GATT result code for procedure: 0 = Success 0x01-0x7F = Error from Bluetooth® specification 0x80-0xFF = Error from application (user-defined)

#### Related commands

*gattc\_discover\_services* (/DRS, ID=6/1) – Always triggers this event upon completion

*gattc\_discover\_characteristics* (/DRC, ID=6/2) – Always triggers this event upon completion

*gattc\_discover\_descriptors* (/DRD, ID=6/3) – Always triggers this event upon completion

*gattc\_read\_handle* (/RRH, ID=6/4) – Triggers this event if read fails, otherwise triggers *gattc\_data\_received* (D, ID=6/3)

#### Related events

*gattc\_discover\_result* (DR, ID=6/1) – Occurs during a remote GATT discovery prior to this event

#### Example usage

See section [How to discover a remote server's GATT structure](#).

### 7.3.4.3 *gattc\_data\_received* (D, ID=6/3)

The remote GATT Server has returned or pushed a value from one of its attributes.

This event occurs after sending a read request with the *gattc\_read\_handle* (/RRH, ID=6/4) API command, or when a remote GATT Server pushes a data update using a notification or indication after the client subscribes to either of these transfer types on supported characteristics. The **source** parameter describes which operation triggered the event.

If the data received came from a remote GATT Server indication and you have disabled automatic confirmations by clearing the **auto-confirm** bit of the **flags** argument in the *gattc\_set\_parameters* (SGCP, ID=6/7) API command, you must manually confirm the indication before performing any other operations. If the **source** parameter of this event has the high bit (0x80) set, use the *gattc\_confirm\_indication* (/CI, ID=6/6) API command.

#### : Binary header

Type	Length	Group	ID	Notes
80	05-19	06	03	Variable-length event payload, minimum of 5 (0x05), maximum of 25 (0x19)

#### Text info

Text name	Event length	Notes
D	0x0016-0x003E	Variable-length event payload, minimum of 22 (0x16), maximum of 62 (0x3E)

**: Event parameters**

Data type	Name	Text	Description
uint8	conn_handle	C	Connection handle
uint16	handle	H	Attribute handle
uint8	source	S	Transfer source: 0x00 = GATT Client read request 0x01 = GATT Server notification 0x02 = GATT Server indication 0x82 = GATT Server indication requiring manual confirmation
longuint8a	data	D	Received value (0-20 bytes) longuint8a data type requires two prefixed "length" bytes before binary parameter payload

**Related commands**

[gatts\\_notify\\_handle \(/NH, ID=5/11\)](#)  
[gatts\\_indicate\\_handle \(/IH, ID=5/12\)](#)  
[gattc\\_read\\_handle \(/RRH, ID=6/4\)](#)  
[gattc\\_confirm\\_indication \(/CI, ID=6/6\)](#)

**7.3.4.4 gattc\_write\_response (WRR, ID=6/4)**

The remote GATT Server acknowledged the GATT Client write operation.

This event occurs after attempting an acknowledged write operation with the [gattc\\_write\\_handle \(/WRH, ID=6/5\)](#) API command. If the write is accepted by the remote server, the `result` value will be 0. Any non-zero `result` value indicates an error.

**: Binary header**

Type	Length	Group	ID	Notes
80	05	06	04	None.

**Text info**

Text name	Event length	Notes
WRR	0x0014	None.

**: Event parameters**

Data type	Name	Text	Description
uint8	conn_handle	C	Connection handle
uint16	attr_handle	H	Attribute handle
uint16	result	R	GATT result code: 0 = Success 0x601-0x067F = Error from Bluetooth® specification 0x680-0x06FF = Error from remote server application (user-defined)

**Related commands**

[gattc\\_write\\_handle \(/WRH, ID=6/5\)](#)

### 7.3.5 SMP Group (ID=7)

SMP methods relate to the Security Manager Protocol layer of the Bluetooth® stack. These methods are used for working with encryption, pairing, and bonding between two peers.

Events within this group are listed below:

smp\_bond\_entry (B, ID=7/1)  
 smp\_pairing\_requested (P, ID=7/2)  
 smp\_pairing\_result (PR, ID=7/3)  
 smp\_encryption\_status (ENC, ID=7/4)  
 smp\_passkey\_display\_requested (PKD, ID=7/5)

Commands within this group are documented in section [SMP Group \(ID=7\)](#).

#### 7.3.5.1 smp\_bond\_entry (B, ID=7/1)

Details about a single entry in the bonding table.

This event occurs once after a new bond is created as a result of the pairing process, or multiple times (based on bond list count) after requesting the bond list with the [smp\\_query\\_bonds \(/QB, ID=7/1\)](#) API command.

##### : Binary header

Type	Length	Group	ID	Notes
80	07	07	01	None.

##### Text info

Text name	Event length	Notes
B	0x001B	None.

##### : Event parameters

Data type	Name	Text	Description
uint8	handle	B	Bonded device handle (1-4)
macaddr	address	A	Bluetooth® address
uint8	type	T	Address type: 0 = Public 1 = Random/private

##### Related commands

[smp\\_query\\_bonds \(/QB, ID=7/1\)](#)  
[smp\\_pair \(/P, ID=7/3\)](#)

#### 7.3.5.2 smp\_pairing\_requested (P, ID=7/2)

Remote device has requested pairing.

##### : Binary header

Type	Length	Group	ID	Notes
80	05	07	02	None.

##### Text info

Text name	Event length	Notes
P	0x0016	None.

**: Event parameters**

Data type	Name	Text	Description
uint8	conn_handle	C	Connection handle
uint8	mode	M	Security level setting reported to peer: 0x10 = Mode 1, Level 1 – No security 0x11 = Mode 1, Level 2 – Unauthenticated pairing with encryption (no MITM) 0x12 = Mode 1, Level 3 – Authenticated pairing with encryption (with MITM) 0x21 = Mode 2, Level 2 – Unauthenticated pairing with data signing (no MITM) 0x22 = Mode 2, Level 3 – Authenticated pairing with data signing (with MITM)
uint8	bonding	B	Bond during pairing process: 0 = Do not bond (exchange keys and encrypt only) 1 = Bond (permanently store exchanged encryption data)
uint8	keysize	K	Encryption key size (7-16), value ignored if pairing initiated by slave device
uint8	pairprop	P	Pairing properties: Bit 0 (0x01): MITM enabled for Secure Connections (SC)

**Related commands**

*smp\_pair* (/P, ID=7/3)  
*smp\_set\_security\_parameters* (SSBP, ID=7/11)

**Related events**

*smp\_pairing\_result* (PR, ID=7/3)

**7.3.5.3 *smp\_pairing\_result* (PR, ID=7/3)**

Pairing process has ended.

This event indicates that the pairing process is finished, successfully or otherwise. If the *result* parameter is 0, then pairing has completed successfully, and the *smp\_bond\_entry* (B, ID=7/1) API event follows if bonding is enabled. Any non-zero *result* value indicates failure.

**: Binary header**

Type	Length	Group	ID	Notes
80	03	07	03	None.

**Text info**

Text name	Event length	Notes
PR	0x000C	None.

**: Event parameters**

Data type	Name	Text	Description
uint8	conn_handle	C	Connection handle
uint16	result	R	Result

**Related commands**

*smp\_pair* (/P, ID=7/3)

**Related events**

*smp\_encryption\_status* (ENC, ID=7/4)  
*smp\_bond\_entry* (B, ID=7/1)

### 7.3.5.4 *smp\_encryption\_status* (ENC, ID=7/4)

Encryption status has changed.

This event confirms that a link has transitioned between plaintext and encrypted status during the pairing process.

#### : Binary header

Type	Length	Group	ID	Notes
80	02	07	04	None.

#### Text info

Text name	Event length	Notes
ENC	0x000E	None.

#### : Event parameters

Data type	Name	Text	Description
uint8	conn_handle	C	Connection handle
uint8	status	S	Encryption status: 0 = success other = error code

#### Related commands

*smp\_pair* (IP, ID=7/3)

#### Related events

*smp\_pairing\_result* (PR, ID=7/3)

### 7.3.5.5 *smp\_passkey\_display\_requested* (PKD, ID=7/5)

Remote peer requires passkey display for entry or comparison during pairing.

This event provides the local device with the passkey generated as part of the pairing process, so that the local device may display or otherwise make it available to the user for entry or comparison on the remote device. This type of passkey generation and display will be used if the local I/O capabilities are set to "Display Only" or "Display + Yes/No" using the *smp\_set\_security\_parameters* (SSBP, ID=7/11) API command.

#### : Binary header

Type	Length	Group	ID	Notes
80	05	07	05	None.

#### Text info

Text name	Event length	Notes
PKD	0x0014	None.

#### : Event parameters

Data type	Name	Text	Description
uint8	conn_handle	C	Connection handle
uint32	passkey	P	Passkey to display (should be displayed to user in decimal format)

#### Related commands

None.

#### Related events

*smp\_pairing\_requested* (P, ID=7/2)  
*smp\_pairing\_result* (PR, ID=7/3)

#### 7.3.5.6 *smp\_pin\_entry\_requested (BTPIN, ID=7/7)*

A remote device has generated and displayed a passkey which must be entered locally and sent back for comparison.

##### *: Binary header*

Type	Length	Group	ID	Notes
80	06	07	07	None.

##### **Text info**

Text name	Event length	Notes
BTPIN	0x0015	None.

##### *: Event parameters*

Data type	Name	Text	Description
macaddr	address	A	macaddr address

##### **Related commands**

*smp\_send\_pinreq\_response (/BTPIN, ID=7/17)*

##### **Related events**

None.

### 7.3.6 GPIO Group (ID=9)

GPIO methods relate to the physical pins on the module.

Events within this group are listed in [gpio\\_interrupt \(INT, ID=9/1\)](#)

Commands within this group are documented in section [GPIO Group \(ID=9\)](#).

#### 7.3.6.1 [gpio\\_interrupt \(INT, ID=9/1\)](#)

A configured GPIO interrupt has occurred.

This event is generated for GPIO edge changes that have enabled interrupts via the [gpio\\_set\\_drive \(SIOD, ID=9/5\)](#) API command.

#### : Binary header

Type	Length	Group	ID	Notes
80	09	09	01	None.

#### Text info

Text name	Event length	Notes
INT	0x00A	None.

#### : Event parameters

Data type	Name	Text	Description
uint8	pin	P	Pin number
uint8	logic	L	pin logic state (set bit indicates HIGH)
uint32	runtime	R	Number of seconds since boot
uint16	fraction	F	Fraction of a second (units are 1/32768)

#### Related commands:

[gpio\\_set\\_drive \(SIOD, ID=9/5\)](#)



### 7.3.7 CYSPP Group (ID=10)

CYSPP methods are related to the Cypress Serial Port Profile.

Events within this group are listed in [p\\_cyspp\\_status \(.CYSPP, ID=10/1\)](#)

Commands within this group are documented in section [CYSPP Group \(ID=10\)](#).

#### 7.3.7.1 [p\\_cyspp\\_status \(.CYSPP, ID=10/1\)](#)

CYSPP operational status has changed.

**Note:** If this event occurs within EZ-Serial firmware platform for Ezurio Vela IF820 series module and data mode is active (either Bit 0 or Bit 1 set and the CYSPP GPIO pin is not externally de-asserted), the wired serial interface is logically disconnected from the API protocol parser and routed to CYSPP data pipe instead. For this reason, this event is never transmitted out the serial interface with Bit 5 set (0x20), because outgoing API events are suppressed while operating in CYSPP data mode.

#### : Binary header

Type	Length	Group	ID	Notes
80	01	0A	01	None.

#### Text info

Text name	Event length	Notes
.CYSPP	0x000C	None.

#### : Event parameters

Data type	Name	Text	Description
uint8	status	S	CYSPP status bitmask: Bit 0 (0x01) = Unacknowledged data subscribed Bit 1 (0x02) = Acknowledged data subscribed Bit 2 (0x04) = RX flow subscribed Bit 3 (0x08) = RX flow blocked by remote Server Bit 4 (0x10) = CYSPP peer support verified Bit 5 (0x20) = Data mode active (used internally)

#### Related commands

[p\\_cyspp\\_start \(.CYSPPSTART, ID=10/2\)](#)

[p\\_cyspp\\_set\\_parameters \(.CYSPPSP, ID=10/3\)](#)

#### Example usage

See section [Cable Replacement Examples with CYSPP](#).

### 7.3.8 Bluetooth® Classic Group (ID=14)

BT methods relate to the Bluetooth® Classic of the Bluetooth® stack. These methods are used for working with Inquiry, connection, and disconnection.

Events within this group are listed below:

bt\_inquiry\_result (BTIR, ID=14/1)  
 bt\_name\_result (BTINR, ID=14/2)  
 bt\_inquiry\_complete (BTIC, ID=14/3)  
 bt\_connected (BTCON, ID=14/4)  
 bt\_connection\_status (BTCES, ID=14/5)  
 bt\_connection\_failed (BTCF, ID=14/6)  
 bt\_disconnected (BTDIS, ID=14/7)

Commands within this group are documented in section [BT group \(ID=14\)](#).

#### 7.3.8.1 bt\_inquiry\_result (BTIR, ID=14/1)

An ongoing inquiry process has returned a result.

##### : Binary header

Type	Length	Group	ID	Notes
80	0B	0E	01	None.

##### Text info

Text name	Event length	Notes
BTIR	0x0024	None.

##### : Event parameters

Data type	Name	Text	Description
macaddr	address	A	Bluetooth® address
uint8	bond	B	Bond entry
uint32	cod	C	Class of device

##### Related commands

bt\_start\_inquiry (/BTI, ID=14/1]  
 bt\_cancel\_inquiry (/BTIX, ID=14/2)

##### Related events

bt\_name\_result (BTINR, ID=14/2)  
 bt\_inquiry\_complete (BTIC, ID=14/3)

**7.3.8.2** *bt\_name\_result (BTINR, ID=14/2)*

An ongoing inquiry process has returned a name result.

**: Binary header**

Type	Length	Group	ID	Notes
80	08	0E	02	Variable-length event payload, value specified is minimum.

**Text info**

Text name	Event length	Notes
BTINR	0x001D+	Variable-length response payload, value specified is minimum.

**: Event parameters**

Data type	Name	Text	Description
macaddr	address	A	Bluetooth® address
uint8	bond	B	Bond entry
uint8a	name	N	Device name

**Related commands**

*bt\_start\_inquiry (/BTI, ID=14/1)*  
*bt\_cancel\_inquiry (/BTIX, ID=14/2)*  
*bt\_query\_name (/BTQN, ID=14/3)*

**Related events:**

*bt\_inquiry\_result (BTIR, ID=14/1)*  
*bt\_inquiry\_complete (BTIC, ID=14/3)*

**7.3.8.3** *bt\_inquiry\_complete (BTIC, ID=14/3)*

An ongoing inquiry process is complete (finished or canceled).

**: Binary header**

Type	Length	Group	ID	Notes
80	00	0E	03	None.

**Text info**

Text name	Event length	Notes
BTIC	0x0005	None.

**Event parameters**

None.

**Related commands**

*bt\_start\_inquiry (/BTI, ID=14/1)*  
*bt\_cancel\_inquiry (/BTIX, ID=14/2)*  
*bt\_query\_name (/BTQN, ID=14/3)*

**Related events**

*bt\_inquiry\_result (BTIR, ID=14/1)*  
*bt\_name\_result (BTINR, ID=14/2)*

#### 7.3.8.4 *bt\_connected (BTCON, ID=14/4)*

A connection has been established to a remote device, and may now be used for data transfers.

##### : Binary header

Type	Length	Group	ID	Notes
80	09	0E	04	None.

##### Text info

Text name	Event length	Notes
BTCON	0x0024	None.

##### : Event parameters

Data type	Name	Text	Description
uint8	conn_handle	C	Connection handle
macaddr	address	A	Bluetooth® address
uint8	type	T	Connection type 1: SPP, connection
uint8	bond	B	Bond entry (0 for no bond)

##### Related commands

*bt\_connect (/BTC, ID=14/4)*  
*bt\_disconnect (/BTDIS, ID=14/6)*

##### Related events

*bt\_connection\_failed (BTCTF, ID=14/6)*  
*bt\_disconnected (BTDIS, ID=14/7)*

#### 7.3.8.5 *bt\_connection\_status (BTCS, ID=14/5)*

A connection has been established to a remote device, and may now be used for data transfers.

##### : Binary header

Type	Length	Group	ID	Notes
80	0B	0E	05	None.

##### Text info

Text name	Event length	Notes
BTCS	0x002D	None.

##### : Event parameters

Data type	Name	Text	Description
uint8	conn_handle	C	Connection handle
macaddr	address	A	Bluetooth® address
uint8	type	T	Connection type
uint8	bond	B	Bond entry
uint8	role	R	Connection role
uint8	sniff	S	Sniff mode

##### Related commands

*bt\_connect (/BTC, ID=14/4)*

*bt\_disconnect (/BTDIS, ID=14/6)*

#### Related events

*bt\_connected (BTCON, ID=14/4)*

*bt\_connection\_failed (BTCTF, ID=14/6)*

*bt\_disconnected (BTDIS, ID=14/7)*

#### 7.3.8.6 *bt\_connection\_failed (BTCTF, ID=14/6)*

A pending outgoing connection attempt has failed.

#### : Binary header

Type	Length	Group	ID	Notes
80	03	0E	06	None.

#### Text info

Text name	Event length	Notes
BTCTF	0x0011	None.

#### : Event parameters

Data type	Name	Text	Description
uint8	conn_handle	C	Connection handle
uint16	reason	R	Reason for connection failure: 1: Unknown reason (Usually it is due to wrong address) 2: No SPP service

#### Related commands

*bt\_connect (/BTC, ID=14/4)*

#### Related events

*bt\_connected (BTCON, ID=14/4)*

*bt\_disconnected (BTDIS, ID=14/7)*

#### 7.3.8.7 *bt\_disconnected (BTDIS, ID=14/7)*

A previously open connection to a remote device has been closed.

#### : Binary header

Type	Length	Group	ID	Notes
80	03	0E	07	None.

#### Text info

Text name	Event length	Notes
BTDIS	0x0012	None.

#### : Event parameters

Data type	Name	Text	Description
uint8	conn_handle	C	Connection handle
uint16	reason	R	Reason for disconnection (Always 0 current)

#### Related commands

*bt\_connect (/BTC, ID=14/4)*

*bt\_disconnect (/BTDIS, ID=14/6)*

#### Related events

bt\_connected (BTCON, ID=14/4)

bt\_connection\_failed (BTCF, ID=14/6)

### 7.3.9 Spp group (ID=19)

#### 7.3.9.1 SPP\_data\_received (SPPD, ID=19/1)

Remote SPP Client has written data.

A connected SPP remote client can write data to a local server.

#### : Binary header

Type	Length	Group	ID	Notes
80	04	19	01	Variable-length event payload, value specified is minimum.

#### Text info

Text name	Event length	Notes
SPPD	0x0016+	Variable-length event payload, value specified is minimum.

#### : Event parameters

Data type	Name	Text	Description
uint16	conn_handle	C	Handle of connection
longuint8a	data	D	Received data longuint8a data type requires two prefixed "length" bytes before binary parameter payload.

#### Related commands

spp\_send\_command (.SPPS, ID=19/1)  
spp\_set\_config (.SPPSC, ID=19/2)  
spp\_get\_config (.SPPGC, ID=19/3)

#### Related events

None

## 7.4 Error codes

### 7.4.1 EZ-Serial firmware platform for Vela IF820 system error codes

The complete list of all result/error codes generated by EZ-Serial firmware platform for Ezurio Vela IF820 series module is listed in 0. See the command and event reference in section [API commands and responses](#) and section API events for specific details about each result within the context of the responses and events where they are triggered.

#### : EZ-Serial firmware platform for Ezurio Vela IF820 series module system error codes

Code (Hex)		
0000	EZS_ERR_SUCCESS	Operation successful, no error
0100	EZS_ERR_CORE	Core system error category
0101	EZS_ERR_CORE_NULL_POINTER	Null pointer encountered (internal error)
0102	EZS_ERR_CORE_MALLOC_FAILED	Memory allocation failed (internal error)
0103	EZS_ERR_CORE_BUFFER_OVERFLOW	Buffer overflow (internal error)
0104	EZS_ERR_CORE_FEATURE_NOT_IMPLEMENTED	Unsupported feature (internal error)
0105	EZS_ERR_CORE_TASK_SCHEDULE_OVERFLOW	Task scheduling attempted but schedule is full
0106	EZS_ERR_CORE_TASK_QUEUE_OVERFLOW	Task queue attempted but queue is full
0107	EZS_ERR_CORE_INVALID_STATE	Invalid state for requested operation
0108	EZS_ERR_CORE_OPERATION_NOT_PERMITTED	Operation not permitted
0109	EZS_ERR_CORE_INSUFFICIENT_RESOURCES	Insufficient resources for requested action
010A	EZS_ERR_CORE_FLASH_WRITE_NOT_PERMITTED	Unable to perform flash write at this time
010B	EZS_ERR_CORE_FLASH_WRITE_FAILED	Flash write operation failed during write
010C	EZS_ERR_CORE_HARDWARE_FAILURE	Internal chipset hardware failure
010D	EZS_ERR_CORE_BLE_INITIALIZATION_FAILED	Could not initialize BLE stack
010E	EZS_ERR_CORE_REPEATED_ATTEMPTS	Repeated attempts to initialize BLE stack
010F	EZS_ERR_CORE_TX_POWER_READ	Could not read radio TX power
0110	EZS_ERR_CORE_DB_VERIFICATION_FAILED	Verification prevented custom attribute addition
0200	EZS_ERR_PROTOCOL	Protocol error category
0201	EZS_ERR_PROTOCOL_UNRECOGNIZED_PACKET_TYPE	Unsupported packet type for text parsing (internal error)
0202	EZS_ERR_PROTOCOL_UNRECOGNIZED_ARGUMENT_TYPE	Unsupported argument type for text parsing (internal error)
0203	EZS_ERR_PROTOCOL_UNRECOGNIZED_COMMAND	Command group/method not valid or unrecognized
0204	EZS_ERR_PROTOCOL_UNRECOGNIZED_RESPONSE	Response group/method invalid or unrecognized (internal error)
0205	EZS_ERR_PROTOCOL_UNRECOGNIZED_EVENT	Event group/method invalid or unrecognized (internal error)
0206	EZS_ERR_PROTOCOL_SYNTAX_ERROR	Syntax error while parsing text command
0207	EZS_ERR_PROTOCOL_COMMAND_TIMEOUT	Binary command packet transmission not completed in required time



Code (Hex)		
0208	EZS_ERR_PROTOCOL_RESPONSE_PENDING	Command already sent but response still pending
0209	EZS_ERR_PROTOCOL_INVALID_CHECKSUM	Binary command packet has invalid checksum
020A	EZS_ERR_PROTOCOL_INVALID_COMMAND_LENGTH	Command length is greater than maximum
020B	EZS_ERR_PROTOCOL_INVALID_PARAMETER_COUNT	Incorrect number of parameters provided
020C	EZS_ERR_PROTOCOL_INVALID_PARAMETER_VALUE	Command parameter outside of acceptable range
020D	EZS_ERR_PROTOCOL_MISSING_REQUIRED_ARGUMENT	Text-mode command missing required arguments
020E	EZS_ERR_PROTOCOL_INVALID_HEXADECIMAL_DATA	Invalid hexadecimal data provided (not 0-9, A-F)
020F	EZS_ERR_PROTOCOL_INVALID_ESCAPE_SEQUENCE	Invalid escape sequence
0210	EZS_ERR_PROTOCOL_INVALID_MACRO_SEQUENCE	Invalid macro sequence
0211	EZS_ERR_PROTOCOL_FLASH_SETTINGS_PROTECTED	Attempted direct flash write of protected setting
<b>0300</b>	<b>EZS_ERR_GPIO</b>	<b>GPIO error category</b>
0301	EZS_ERR_GPIO_PORT_NOT_SUPPORTED	Selected port in GPIO command not supported
<b>0400</b>	<b>EZS_ERR_LL</b>	<b>Link layer error category</b>
0401	EZS_ERR_LL_CONTROLLER_BUSY	Link layer controller busy
0402	EZS_ERR_LL_NO_DEVICE_ENTITY	Device entity not available
0403	EZS_ERR_LL_NOT_IN_BOND_LIST	Device not found in bond list
0404	EZS_ERR_LL_DEVICE_ALREADY_EXISTS	Device already exists
<b>0500</b>	<b>EZS_ERR_GAP</b>	<b>GAP error category</b>
0501	EZS_ERR_GAP_INVALID_CONNECTION_HANDLE	Invalid connection handle specified
0502	EZS_ERR_GAP_CONNECTION_REQUIRED	Connection required, but none is available
0503	EZS_ERR_GAP_ROLE	Incorrect GAP role for this operation
0504	EZS_ERR_GAP_ADV_QUEUE_OVERFLOW	Advertisement queue attempted but queue is full
<b>0600</b>	<b>EZS_ERR_GATT</b>	<b>GATT error category</b>
0601	EZS_ERR_GATT_INVALID_ATTRIBUTE_HANDLE	Invalid attribute handle for GATT operation
0602	EZS_ERR_GATT_READ_NOT_PERMITTED	Read not permitted on this attribute
0603	EZS_ERR_GATT_WRITE_NOT_PERMITTED	Write not permitted on this attribute
0604	EZS_ERR_GATT_INVALID_PDU	Invalid PDU for requested operation
0605	EZS_ERR_GATT_INSUFFICIENT_AUTHENTICATION	Insufficient authentication for requested operation
0606	EZS_ERR_GATT_REQUEST_NOT_SUPPORTED	Request not supported
0607	EZS_ERR_GATT_INVALID_OFFSET	Invalid offset specified for requested operation
0608	EZS_ERR_GATT_INSUFFICIENT_AUTHORIZATION	Insufficient authorization for requested operation
0609	EZS_ERR_GATT_PREPARE_WRITE_QUEUE_FULL	Prepare write queue full, cannot prepare new write
060A	EZS_ERR_GATT_ATTRIBUTE_NOT_FOUND	Attribute not found in database

Code (Hex)		
060B	EZS_ERR_GATT_ATTRIBUTE_NOT_LONG	Attribute not long when long operation requested
060C	EZS_ERR_GATT_INSUFFICIENT_ENC_KEY_SIZE	Insufficient encryption key size
060D	EZS_ERR_GATT_INVALID_ATTRIBUTE_LENGTH	Invalid attribute length
060E	EZS_ERR_GATT_UNLIKELY_ERROR	Unlikely error occurred, unknown cause
060F	EZS_ERR_GATT_INSUFFICIENT_ENCRYPTION	Insufficient encryption for requested operation
0610	EZS_ERR_GATT_UNSUPPORTED_GROUP_TYPE	Unsupported group type specified in Read By Group Type operation
0611	EZS_ERR_GATT_INSUFFICIENT_RESOURCES	Insufficient resources to perform operation
0680	EZS_ERR_GATT_CLIENT_NOT_SUBSCRIBED	Client has not subscribed to updates on characteristic (local error code when sending notifications or indications)
<b>0800</b>	<b>EZS_ERR_SMP</b>	<b>SMP error category</b>
0801	EZS_ERR_SMP_OOB_NOT_AVAILABLE	Out-of-band pairing data not available
0802	EZS_ERR_SMP_SECURITY_OPERATION_FAILED	Security operation failed
0803	EZS_ERR_SMP_MIC_AUTH_FAILED	Message integrity check authentication failed
<b>0900</b>	<b>EZS_ERR_SPEC</b>	<b>Bluetooth® Core Specification error category</b>
0901	EZS_ERR_SPEC_UNKNOWN_HCI_COMMAND	Unknown HCI command
0902	EZS_ERR_SPEC_UNKNOWN_CONNECTION_IDENTIFIER	Unknown connection identifier
0903	EZS_ERR_SPEC_HARDWARE_FAILURE	Hardware failure
0904	EZS_ERR_SPEC_PAGE_TIMEOUT	Page timeout
0905	EZS_ERR_SPEC_AUTHENTICATION_FAILURE	Authentication Failure
0906	EZS_ERR_SPEC_PIN_OR_KEY_MISSING	PIN or Key Missing
0907	EZS_ERR_SPEC_MEMORY_CAPACITY_EXCEEDED	Memory capacity exceeded
0908	EZS_ERR_SPEC_CONNECTION_TIMEOUT	Connection Timeout
0909	EZS_ERR_SPEC_CONNECTION_LIMIT_EXCEEDED	Connection limit exceeded
090A	EZS_ERR_SPEC_SYNCHRONOUS_CONN_LIMIT_DEVICE_EXCEEDED	Synchronous connection limit to a device exceeded
090B	EZS_ERR_SPEC_ACL_CONNECTION_ALREADY_EXISTS	ACL connection already exists
090C	EZS_ERR_SPEC_COMMAND_DISALLOWED	Command disallowed
090D	EZS_ERR_SPEC_CONNECTION_REJECTED_LIMITED_RESOURCES	Connection rejected due to limited resources
090E	EZS_ERR_SPEC_CONNECTION_REJECTED_SECURITY_REASONS	Connection rejected due to security reasons
090F	EZS_ERR_SPEC_CONNECTION_REJECTED_UNACCEPTABLE_BDADDR	Connection rejected due to unacceptable BD_ADDR
0910	EZS_ERR_SPEC_CONNECTION_ACCEPT_TIMEOUT_EXCEEDED	Connection Accept Timeout exceeded
0911	EZS_ERR_SPEC_UNSUPPORTED_FEATURE_OR_PARAMETER_VALUE	Unsupported feature or parameter value
0912	EZS_ERR_SPEC_INVALID_HCI_COMMAND_PARAMETERS	Invalid HCI command parameters

Code (Hex)		
0913	EZS_ERR_SPEC_REMOTE_USER_TERMINATED _CONNECTION	Remote User Terminated Connection
0914	EZS_ERR_SPEC_REMOTE_DEVICE_TERMINATED _LOW_RESOURCES	Remote device terminated connection due to low resources
0915	EZS_ERR_SPEC_REMOTE_DEVICE_TERMINATED _POWER_OFF	Remote device terminated connection due to power off
0916	EZS_ERR_SPEC_CONNECTION_TERMINATED _BY_LOCAL_HOST	Connection Terminated by Local Host
0917	EZS_ERR_SPEC_REPEATED_ATTEMPTS	Repeated attempts
0918	EZS_ERR_SPEC_PAIRING_NOT_ALLOWED	Pairing Not Allowed
0919	EZS_ERR_SPEC_UNKNOWN_LMP_PDU	Unknown LMP PDU
091A	EZS_ERR_SPEC_UNSUPPORTED_REMOTE _LMP_FEATURE	Unsupported remote feature / unsupported LMP feature
091B	EZS_ERR_SPEC_SCO_OFFSET_REJECTED	SCO offset rejected
091C	EZS_ERR_SPEC_SCO_INTERVAL_REJECTED	SCO interval rejected
091D	EZS_ERR_SPEC_SCO_AIR_MODE_REJECTED	SCO air mode rejected
091E	EZS_ERR_SPEC_INVALID_LMP_LL_PARAMETERS	Invalid LMP parameters / invalid LL parameters
091F	EZS_ERR_SPEC_UNSPECIFIED_ERROR	Unspecified error
0920	EZS_ERR_SPEC_UNSUPPORTED_LMP_LL PARAMETER_VALUE	Unsupported LMP parameter value / Unsupported LL parameter value
0921	EZS_ERR_SPEC_ROLE_CHANGE_NOT_ALLOWED	Role change not allowed
0922	EZS_ERR_SPEC_LMP_LL_RESPONSE_TIMEOUT	LMP Response Timeout / LL Response Timeout
0923	EZS_ERR_SPEC_LMP_ERROR_TRANSACTION_COLLISION	LMP error transaction collision
0924	EZS_ERR_SPEC_LMP_PDU_NOT_ALLOWED	LMP PDU not allowed
0925	EZS_ERR_SPEC_ENCRYPTION_MODE_NOT_ACCEPTABLE	Encryption mode not acceptable
0926	EZS_ERR_SPEC_LINK_KEY_CANNOT_BE_CHANGED	Link key cannot be changed
0927	EZS_ERR_SPEC_REQUESTED_QOS_NOT_SUPPORTED	Requested QoS not supported
0928	EZS_ERR_SPEC_INSTANT_PASSED	Instant passed
0929	EZS_ERR_SPEC_PAIRING_WITH_UNIT_KEY _NOT_SUPPORTED	Pairing with unit key not supported
092A	EZS_ERR_SPEC_DIFFERENT_TRANSACTION_COLLISION	Different transaction collision
092B	/* 0x2B reserved */	Reserved
092C	EZS_ERR_SPEC_QOS_UNACCEPTABLE_PARAMETER = 0x092C	QoS unacceptable parameter
092D	EZS_ERR_SPEC_QOS_REJECTED	QoS rejected
092E	EZS_ERR_SPEC_CHANNEL_CLASSIFICATION NOT_SUPPORTED	Channel classification not supported
092F	EZS_ERR_SPEC_INSUFFICIENT_SECURITY	Insufficient security
0930	EZS_ERR_SPEC_PARAMETER_OUT_OF MANDATORY_RANGE	Parameter out of mandatory range
0931	/* 0x31 reserved */	Reserved
0932	EZS_ERR_SPEC_ROLE_SWITCH_PENDING = 0x0932	Role switch pending
0933	/* 0x33 reserved */	Reserved

Code (Hex)		
0934	EZS_ERR_SPEC_RESERVED_SLOT_VIOLATION = 0x0934	Reserved slot violation
0935	EZS_ERR_SPEC_ROLE_SWITCH_FAILED	Role switch failed
0936	EZS_ERR_SPEC_EXTENDED_INQUIRY_RSP_TOO_LARGE	Extended inquiry response too large
0937	EZS_ERR_SPEC_SSP_NOT_SUPPORTED_BY_HOST	Secure simple pairing not supported by host
0938	EZS_ERR_SPEC_HOST_BUSY_PAIRING	Host busy - pairing
0939	EZS_ERR_SPEC_CONNECTION_REJECTED _NO_SUITABLE_CHANNEL	Connection rejected due to no suitable channel found
093A	EZS_ERR_SPEC_CONTROLLER_BUSY	Controller busy
093B	EZS_ERR_SPEC_UNACCEPTABLE _CONNECTION_PARAMETERS	Unacceptable connection parameters
093C	EZS_ERR_SPEC_DIRECTED_ADVERTISING_TIMEOUT	Directed advertising timeout
093D	EZS_ERR_SPEC_CONNECTION_TERMINATED _MIC_FAILURE	Connection terminated due to MIC failure
093E	EZS_ERR_SPEC_CONNECTION_FAILED _TO_BE_ESTABLISHED	Connection Failed to be Established
093F	EZS_ERR_SPEC_MAC_CONNECTION_FAILED	MAC connection failed
0940	EZS_ERR_SPEC_COARSE_CLOCK_ADJ_REJECTED	Coarse clock adjustment rejected but will try to adjust using clock dragging
EEEE	EZS_ERR_UNKNOWN	Unknown problem (internal error)

#### 7.4.2 EZ-Serial firmware platform for Vela IF820 GATT database validation error codes

The complete list of result/error codes generated by EZ-Serial firmware platform for Ezurio Vela IF820 series module during dynamic GATT database validation is listed in 0 . See section [Defining custom local GATT services and characteristics](#) and the documentation for the related [GATT Server Group \(ID=5\)](#) API command methods for detail.

: EZ-Serial firmware platform for Ezurio Vela IF820 series module GATT validation error codes

Code (Hex)	Name	Description
0000	GATTS_DB_VALID_OK	Validation passed with no warnings or errors
0001	GATTS_DB_VALID_WARNING_NOT_ENOUGH_ATTRIBUTES	Structure is valid, but more attributes are required
0002	GATTS_DB_VALID_ERROR_ATTRIBUTE_LIMIT_EXCEEDED	Attribute count limit exceeded
0003	GATTS_DB_VALID_ERROR_ATTRIBUTE_DATA_EXCEEDED	Runtime attribute value data byte limit exceeded
0004	GATTS_DB_VALID_ERROR_CONSTANT_DATA_EXCEEDED	Constant default data byte limit exceeded
0005	GATTS_DB_VALID_ERROR_CCCD_LIMIT_EXCEEDED	CCCD attribute limit exceeded
0006	GATTS_DB_VALID_ERROR_SVC_DECL_REQUIRED	Service declaration required
0007	GATTS_DB_VALID_ERROR_UNEXPECTED_SVC_DECL	Unexpected service declaration
0008	GATTS_DB_VALID_ERROR_CHAR_DECL_REQUIRED	Characteristic declaration required
0009	GATTS_DB_VALID_ERROR_UNEXPECTED_CHAR_DECL	Unexpected characteristic declaration
000A	GATTS_DB_VALID_ERROR_CHAR_VALUE_REQUIRED	Characteristic value attribute required
000B	GATTS_DB_VALID_ERROR_UNEXPECTED_DESCRIPTOR	Specified descriptor not allowed at this position
000C	GATTS_DB_VALID_ERROR_INVALID_ATT_PROPERTIES	Attribute properties not compatible with type

Code (Hex)	Name	Description
000D	GATTS_DB_VALID_ERROR_INVALID_ATT_LENGTH	Invalid attribute length
000E	GATTS_DB_VALID_ERROR_INVALID_ATT_DATA	Attribute data not compatible with type

## 7.5 Macro definitions

Macros in EZ-Serial firmware platform for Ezurio Vela IF820 series module are simple codes that result in text substitution within the parser. Macros may be used in either text mode or binary mode. Macros always begin with the '%' character and are followed by one or more alphanumeric characters (A-Z, 0-9). Macros are not case-sensitive.

: *Macro code table*

Code	Description	Example Input	Example Output	Notes
%M1	Byte #1 of local public MAC address	MyDevice %M1	MyDevice 00	Examples assume that the local device has a public MAC address of 00:A0:50:E3:83:5F.
%M2	Byte #2 of local public MAC address	MyDevice %M2	MyDevice A0	
%M3	Byte #3 of local public MAC address	MyDevice %M3	MyDevice 50	
%M4	Byte #4 of local public MAC address	MyDevice %M4	MyDevice E3	
%M5	Byte #5 of local public MAC address	MyDevice %M5	MyDevice 83	
%M6	Byte #6 of local public MAC address	MyDevice %M6	MyDevice 5F	

Macros may be used in series with or without special separators, if the entire macro code (including the '%' byte) remains intact. For example, to use the last three bytes of the MAC address in the same string, separated by the ':' byte, use the following:

```
MyDevice %M4:%M5:%M6
```

This string is particularly useful for setting a module-specific device name using the `gap_set_device_name` (SDN, ID=4/15) API command without needing to query or track the MAC address separately by hand.

## 8 GPIO Reference

This section describes the various GPIO connections provided by the EZ-Serial firmware platform for Ezurio Vela IF820 series module firmware on supported modules. It also provides details on the default boot state and the expected behavior in different operational modes.

### 8.1 GPIO pin map for Vela IF820 DVK

Refer to the [Vela IF820 DVK User Guide](#).

---

**Note:** User need check and do not use pins which have pre-configured by Ezurio Vela IF820 series module.

---

## 9 GATT profile

### 9.1 CYSPP Profile

The Cypress Serial Port Profile (CYSPP) provides bidirectional serial data transfer between two remote devices, each of which passes data in through a single local hardware serial interface. It supports both acknowledged transfers and unacknowledged transfers and provides a mechanism for virtual flow control in both the RX and TX direction.

The profile contains a single service ("CYSPP"), which contains three characteristics for data transfer and flow control ("Acknowledged Data", "Unacknowledged Data", and "RX Flow"). The structural outline of this profile is as follows:

CYSPP Service: UUID 65333333-A115-11E2-9E9A-0800200CA100

**Acknowledged Data Characteristic:** UUID 65333333-A115-11E2-9E9A-0800200CA101  
(Write, Indicate)

The Acknowledged Data Characteristic is used to send and receive data in an acknowledged fashion. The EZ-Serial firmware platform for Ezurio Vela IF820 series module can fully track every transfer in both directions. This characteristic has a variable length, supporting transfers in each direction of up to 20 bytes per packet.

Configuration Descriptor: UUID 0x2902

**Unacknowledged Data Characteristic:** UUID 65333333-A115-11E2-9E9A-0800200CA102  
(Write without response, Notify)

The Unacknowledged Data Characteristic is used to send and receive data in an unacknowledged fashion. The EZ-Serial firmware platform for Ezurio Vela IF820 series module cannot track transfers using this mode once they have been accepted by the BLE stack. This provides less control, but the lack of acknowledgements also allows for greater maximum throughput. This characteristic has a variable length, supporting transfers in each direction of up to 20 bytes per packet.

Configuration Descriptor: UUID 0x2902

**RX Flow Characteristic:** UUID 65333333-A115-11E2-9E9A-0800200CA103  
(Indicate)

The RX Flow Characteristic is used to indicate to the client that the server can no longer safely receive new data. If the client subscribes to indications from this characteristic, the server will assume that the client obeys flow control signals. This characteristic is one byte in length. An indicated value of "0" means that it is safe for the client to send data, while a value of "1" means that the client must refrain from sending data.

Configuration Descriptor: UUID 0x2902

## 10 Configuration example reference

The configuration examples provided in this section are each designed to work independently, assuming in each case that the platform is initially configured using factory default settings. Applying all commands in one example and then immediately following this with the commands from another example may result in changes to the first set of behavior that are no longer in line with the expected results.

You can return a module to factory defaults as a baseline configuration at any time by using the `system_factory_reset (/RFAC, ID=2/5)` API command. This reset command is not explicitly included in any of the configuration snippets within this section.

## 10.1 Factory default settings

While you can return to the factory default settings on the module by performing a factory reset, it is also helpful to know those settings for comparison or to explicitly change one or more individual settings to the default value without reverting all customizations at once. 0 provides a comprehensive list of commands that will return the EZ-Serial firmware platform for Ezurio Vela IF820 series module to default behavior.

### : List of commands

Text content	Binary content
SPPM,M=01	C0 01 01 01 01 5D
SSLP,L=01	C0 01 02 13 01 70
STXP,P=07	C0 01 02 15 07 78
STU,B=0001C200,A=00,C=00,F=00,D=08,P=00,S=01	C0 0A 02 19 00 C2 01 00 00 00 00 08 00 01 4A
SDN,N=EZ-Serial %M4:%M5:%M6	C0 16 04 0F 15 45 5A 2D 53 65 72 69 61 6C 20 25 4D 34 3A 25 4D 35 3A 25 4D 36 4C
SDA,A=0000	C0 02 04 11 00 00 70
SAD,D=	C0 01 04 13 00 71
SSRD,D=	C0 01 04 15 00 73
SAP, M=02,T=03,C=07,H=0030,D=001E,L=0800,O=003C, F=00,A=000000000000,Y=00	C0 13 04 17 02 03 07 30 00 1E 00 00 08 3C 00 00 00 00 00 00 00 00 00 00 25
SGSP,F=01	C0 01 05 0E 01 6E
SPRV,M=00,I=012C	C0 03 07 09 00 2C 01 99
SSBP,M=41,B=01,K=10,P=00,I=03,F=01	C0 06 07 0B 11 01 10 00 03 01 97
.CYSPPSP,E=02,G=00,C=0131,L=00000000,R=00000000, M=00000000,P=01,S=00,F=02	C0 13 0A 03 02 00 31 01 00 00 00 00 00 00 00 00 00 00 00 00 01 00 02 B0
.CYSPPSK,M=01,W=05,L=14,E=0D	C0 04 0A 07 01 05 14 0D 95

Remember that the commands in 0 affect only RAM. To make these command values permanent, apply all settings to flash using the `system_store_config (/SCFG, ID=2/4)` API command.

## 10.2 Adopted Bluetooth SIG GATT profile structure snippets

The snippets below demonstrate how to add various GATT service and characteristic structural elements to support official profiles defined by the Bluetooth® SIG, and some other common services.

---

**Note:** These database structures concern only the GATT Server side of the profiles in question. GATT Client operations depend on the client device.

---

---

**Note:** The information provided in this section only covers the basic GATT structure, but does not include any specific values which may be necessary or helpful for specific functionality. Many characteristics also have flexible length values which depend on application design. See the official Bluetooth® SIG documentation or other related resources linked under each service for further detail.

---

### 10.2.1 Generic access service (0x1800)

Official documentation for this service can be found on the [Bluetooth® SIG Developer website](#).

---

**Note:** This service is included in the EZ-Serial firmware platform for Ezurio Vela IF820 series module. It is always present in the fixed, non-removable part of the GATT structure. Do not add another instance of this service to the EZ-Serial firmware platform for Ezurio Vela IF820 series module.

---

```
/CAC, T=0, P=02, L=04, D=00280018
/CAC, T=0, P=02, L=07, D=0328020300002A
/CAC, T=1, P=0B, L=40, D=
/CAC, T=0, P=02, L=07, D=0328020500012A
/CAC, T=1, P=02, L=02, D=
/CAC, T=0, P=02, L=07, D=0328020700042A
/CAC, T=1, P=02, L=08, D=
/CAC, T=0, P=02, L=07, D=0328020900A62A
/CAC, T=1, P=02, L=01, D=
```

---

**Note:** EZ-Serial firmware platform for Ezurio Vela IF820 series module assumes that the attribute handle is starting from 1. Data item of characteristic attribute include the attribute handle (0x0003, 0x0005, 0x0007 and 0x0009 respectively in this example) which corresponding to the characteristic value attribute.

---



### 10.2.2 Generic Attribute Service (0x1801)

Official documentation for this service can be found on the [Bluetooth® SIG Developer website](#).

**Note:** This service is included in the EZ-Serial firmware platform for Ezurio Vela IF820 series module. It is always present in the fixed, non-removable part of the GATT structure. Do not add another instance of this service to the EZ-Serial firmware platform for Ezurio Vela IF820 series module.

```
/CAC, T=0, P=02, L=04, D=00280018
/CAC, T=0, P=02, L=07, D=0328200300052A
/CAC, T=1, P=02, L=04, D=
/CAC, T=0, P=0A, L=04, D=0229
```

**Note:** EZ-Serial firmware platform for Ezurio Vela IF820 series module assumes that the attribute is handled starting from 1. Attribute handle (0x0003) corresponding to the value attribute.

### 10.2.3 Immediate alert service (0x1802)

Official documentation for this service can be found on the [Bluetooth® SIG Developer website](#).

```
/CAC, T=0, P=02, L=04, D=00280218
/CAC, T=0, P=02, L=07, D=0328041800062A
/CAC, T=1, P=0A, L=01, D=
```

### 10.2.4 Link loss service (0x1803)

Official documentation for this service can be found on the [Bluetooth® SIG Developer website](#).

```
/CAC, T=0, P=02, L=04, D=00280318
/CAC, T=0, P=02, L=07, D=03280A1800062A
/CAC, T=1, P=0A, L=01, D=
```

### 10.2.5 TX power service (0x1804)

Official documentation for this service can be found on the [Bluetooth® SIG Developer website](#).

```
/CAC, T=0, P=02, L=04, D=00280418
/CAC, T=0, P=02, L=07, D=0328021800072A
/CAC, T=1, P=02, L=01, D=
/CAC, T=0, P=0A, L=04, D=0229
```

## 11 EZ-Serial firmware platform for Ezurio Vela IF820 series module MAC address

The EZ-Serial firmware platform for Ezurio Vela IF820 series module includes a static random MAC address when they are shipped from Ezurio. The static random MAC address is configured during Infineon manufacturing programming process, and this address does not change for the life of the programmed image (Exception command /RFAC which will regenerate static random address).

During the Ezurio programming process, the EZ-Serial firmware platform for Ezurio Vela IF820 series module generates a static random address and stores it in flash. The address format follows the *Bluetooth® Core Specification 5.0 Volume 6, part B, Section 1.3.2.1 Static Device Address*. This address is persistent during module power cycle or reset operations.

---

**Note:** EZ-Serial firmware platform for Ezurio Vela IF820 series module internally controls the address type by using `smp_set_privacy_mode (SPRV, ID=7/9)`. If this mode bit 2 is set to 0, it advertises as a public address type. If this mode bit 2 set to 1, it advertises as a static random address type. The default for the EZ-Serial firmware platform for Ezurio Vela IF820 series module address is 1 (static random address).

---

In all cases, you should be familiar with the rules set forth by the Bluetooth® SIG for MAC address generation, format and usage as documented in the *Bluetooth® Core Specification 5.0, Volume 6, section 1.3*.

If you want to use your own public address (using an assigned IEEE OUI), use the `system_set_bluetooth_address (SBA, ID=2/13)` command to configure the address to your OUI plus three additional random bytes, and then use the `smp_set_privacy_mode (SPRV, ID=7/9)` command to change the address type to public.

If you modify the type and format of the address and then want to revert to the EZ-Serial firmware platform for Ezurio Vela IF820 series module initial static random address, use the `system_set_bluetooth_address (SBA, ID=2/13)` command with the parameter address equal to 0. Using this command reverts the advertising address to the factory-provided static random address.

In all cases, you should be familiar with the rules set forth by the Bluetooth® SIG for MAC address generation, format and usage as documented in the *Bluetooth® Core Specification 5.0, Volume 6, section 1.3*.

## 12 Glossary

Acronym/Abbreviation	Expanded Form
ADC	Analog-to-Digital Converter
API	Application Program Interface
BLE	Bluetooth Low Energy
BR	Basic Rate
BT	Blue Tooth
CCCD	Client Characteristic Configuration Descriptor
CPU	Central Processing Unit
CTS	Clear to Send,
CYSPP	Cypress Serial Port Profile Cypress is an Infineon Technologies Company.
EDR	Enhanced Data Rate
EVAL	Evaluation
GAP	Generic Access Protocol
GATT	Generic Attribute Profile
GCC	GNU Compiler Collection
GND	Ground
GPIO	General Purpose Input/Output.
HCI	Host Controller Interface
HID	Human Interface Device
JSON	JavaScript Object Notation
LL	Link Layer
MAC	Media Access Control
MCU	Microcontroller
MITM	Man In The Middle
MSb	Most Significant bit
MSB	Most Significant Byte
MTU	Maximum Transmission Unit
OTA	Over-the-Air programming
PUART	Peripheral UART
PWM	Pulse Width Modulation
RAM	Random Access Memory
RSSI	Received Signal Strength Indicator
RTS	Request to Send
RXD	Receive Data
SDK	Software Development Kit
SIG	Special Interest Group
SMP	Security Manager Protocol

Acronym/Abbreviation	Expanded Form
SPP	Serial Port Profile
TXD	Transmit Data
UART	Universal Asynchronous Receiver Transmitter
UTF-8	Unicode Transformation Format 8
UUID	Universally Unique Identifier
VDD	Voltage Drain Drain
WCO	Watch Crystal Oscillator
WICED	Wireless Internet Connectivity for Embedded Devices

## 13 Additional Information

Please contact your local sales representative or our support team for further assistance:

Headquarters	Ezurio 50 S. Main St. Suite 1100 Akron, OH 44308 USA
Website	<a href="http://www.ezurio.com/">www.ezurio.com/</a>
Technical Support	<a href="http://www.ezurio.com/resources/support">http://www.ezurio.com/resources/support</a>
Sales Contact	<a href="http://www.ezurio.com/contact">http://www.ezurio.com/contact</a>

**Note:** Information contained in this document is subject to change.

Ezurio's products are subject to standard [Terms & Conditions](#).

© Copyright 2025 Ezurio. All Rights Reserved. Patent pending. Any information furnished by Ezurio and its agents is believed to be accurate and reliable. All specifications are subject to change without notice. Responsibility for the use and application of Ezurio materials or products rests with the end user since Ezurio and its agents cannot be aware of all potential uses. Ezurio makes no warranties as to non-infringement nor as to the fitness, merchantability, or sustainability of any Ezurio materials or products for any specific or general uses. Ezurio or any of its affiliates or agents shall not be liable for incidental or consequential damages of any kind. All Ezurio products are sold pursuant to the Ezurio Terms and Conditions of Sale in effect from time to time, a copy of which will be furnished upon request. Nothing herein provides a license under any Ezurio or any third-party intellectual property right.