Technotrade
Forecourt automation

www.technotrade.ua
mail@technotrade.ua
+380 44 502 46 55
Ukraine, 04082 Kyiv, Mrii str. 17

jsonPTS
communication protocol specification
for the PTS-2 forecourt controller

Review date: 04 Nobember, 2025
Revision number: R129

Approved by: _____ *Date: _____*

Technotrade LLC

CONTENT

REVISION HISTORY

REV	DATE	BY	SECTION	DESCRIPTION
R01	9.03.2019	Evgeniy Vasyliev	All	First release
R02	15.08.2019	Evgeniy Vasyliev	Configuration requests and responses Pumps control requests and responses	1. Changed responses to each Get… response: now response is either OK or Error. 2. Changed responses to each PumpGet… response: now response is either OK or Error. 3. Authorization request updated: added field "*AutoCloseTransaction*", added PumpAuthorizeConfirmation response. 4. Added PumpGetTransactionInformatio request and PumpTransactionInformation response 5. Added PumpEmergencyStop request 6. General document formatting
R03	11.09.2019	Evgeniy Vasyliev	PumpAuthorize request ProbeMeasurements response	Added "*Transaction*" parameter Corrected description
R04	01.10.2019	Evgeniy Vasyliev	PumpGetDisplayData request PumpDisplayData response	Added request and response for getting pump display data: dispensed volume and amount
R05	17.10.2019	Evgeniy Vasyliev	ReportGetPumpTransactions request, ReportPumpTransactions response, ReportGetTankMeasurements request, ReportTankMeasurements response	Added volume and amount total counters to ReportPumpTransaction response, made fields *Date* and *Time* to be obligatory in report requests, added total counters in pump transactions response
R06	01.11.2019	Evgeniy Vasyliev	SetPtsNetworkSettings request, PtsNetworkSettings response	Added "*HttpPort*" and "*HttpsPort*" parameters for ports using HTTP and HTTPS connections accordingly, "*Port*" parameter removed
R07	05.11.2019	Evgeniy Vasyliev	PumpDisplayData response	Added "*LastTransaction*" and "*LastNozzle*" parameters
R08	09.11.2019	Evgeniy Vasyliev	SetPortLoggingConfiguration request, GetPortLoggingConfiguration request, PortLoggingConfiguration response, SetDateTime request, DateTime response, ProbeMeasurements response, ReportGetPumpTransactions request, ReportPumpTransactions response, ReportGetTankMeasurements request, ReportTankMeasurements response, FirmwareInformation request, PumpTransactionInformation response	1. Added requests and response for port logging. 2. Datetime in format *YYYY-MM-DDThh:mm:ss* in accordance with ISO 8601. 3. General document formatting.
R09	17.11.2019	Evgeniy Vasyliev	PumpGetTotals request, PumpAuthorize request	Field name "*GradeCode*" changed to "*FuelGradeId*".
R10	16.12.2019	Evgeniy Vasyliev	BackupConfiguration request, RestoreConfiguration request	Added requests for backup and restore configuration. Corrected document for correct datetime format displaying in *YYYY-MM-DDThh:mm:ss* in accordance with ISO 8601.
R11	27.12.2019	Evgeniy Vasyliev	SetUsersConfiguration request, UsersConfiguration response	1. Monitoring permission added. 2. Added required permissions list to requests description
R12	03.01.2020	Evgeniy Vasyliev	SetUsersConfiguration request, UsersConfiguration response, ReportGetTankMeasurements request	1. Added remote server configuration parameters 2. Added value 0 in ReportGetTankMeasurements request to get values for all tanks
R13	28.01.2020	Evgeniy Vasyliev	SetUsersConfiguration request, UsersConfiguration response, UserInformation response, description of confirmation/error response message, UploadPumpTransaction request, UploadTankMeasurement request.	1. Updated remote server configuration parameters: added URI field 2. Added description of confirmation/error message. 3. Added description of UploadPumpTransaction request. 4. Added description of UploadTankMeasurement request.
R14	05.02.2020	Evgeniy Vasyliev	SetTanksConfiguration request, TanksConfiguration response, SetPtsNetworkSettings request, PtsNetworkSettings request, ProbeGetTankVolumeForHeight request,	1. Added an option to use tank calibration chard request. 2. Added DNS configuration. 3. Added request for getting tank volume from height based on tank's calibration chart.

			ProbeTankVolumeForHeight response Testing using Postman utility	
R15	21.05.2020	Evgeniy Vasyliev	Logout request	Logout request removed
R16	10.07.2020	Evgeniy Vasyliev	PumpSetAutomaticOperation request	New requests added
R17	28.07.2020	Evgeniy Vasyliev	ProbeMeasurements response updated	ProbeMeasurements response format updated for in-tank delivery
			FileDelete request added	FileDelete request added
			PumpAuthorize request updated	PumpAuthorize request updated: added a possibility to authorize a group of nozzles by nozzle numbers or fuel grades
R18	01.10.2020	Evgeniy Vasyliev	SetFuelGradesConfiguration request FuelGradesConfiguration response	Added "*ExpansionCoefficient*" field in order to be able to calculate temperature-compensated volume of fuel grade in tank.
			SetTanksConfiguration request TanksConfiguration response	Removed "*UseCalibrationChart*" field as it is now common parameter for probes.
			SetPumpNozzlesConfiguration request PumpNozzlesConfiguration response	Added array "*TankIds*" for specifying linkage between pump nozzle and tank.
			SetSystemDecimalDigits request	Request added.
			PumpTransactionInformation response	Added "*DateTimeStart*" field, added "*TCVolume*" field.
			PumpFillingStatus response	Added "*TCVolume*" field.
			PumpEndOfTransactionStatus response	Added "*TCVolume*" field.
			ReportPumpTransactions response	Added "*DateTimeStart*" field, added "*TCVolume*" field.
			UploadPumpTransaction request UploadPumpTransaction request UploadTankMeasurement request	Added "*DateTimeStart*" field, added "*TCVolume*" field. Each packet should contain a unique packet ID, Fields "*DateTimeStart*" and "*TCVolume*" added to UploadPumpTransaction request
			Restart request	Added response
R19	15.11.2020	Evgeniy Vasyliev	ProbeMeasurements response	Probe alarms for tank added
			SetTanksConfiguration request TanksConfiguration response	Fields for specification of product maximum and minimum allowed height added
R20	07.12.2020	Evgeniy Vasyliev	PumpGetTotals request PumpGetPrices request PumpSetPrices request PumpGetTag request	Corrected notes Corrected notes Corrected notes Corrected notes
			GetSystemOperationInformation request SystemOperationInformation response	Added request Added response
R21	01.01.2021	Evgeniy Vasyliev	GetLanguage request Language response	Added request Added response
			SetDateTime request DateTime response	Added fields for automatic time synchronization wit time server
			BackupConfiguration request	Added field for automatic configuration restore on startup
			Error messages section	List of error messages and their descriptions
R22	25.01.2021	Evgeniy Vasyliev	PumpGetAutomaticOperation request PumpAutomaticOperation response PumpTransactionInformation response	Added request Added response Added "*TotalVolume*" and "*TotalAmount*" fields
R23	27.02.2021	Evgeniy Vasyliev	GetMeasurementUnits request MeasurementUnits response	Added request Added response
			PumpGetTotals request PumpGetPrices request PumpSetPrices request PumpGetTag request PumpGetDisplayData request	Corrected request notes
			GetRemoteServerConfiguration request SetRemoteServerConfiguration request RemoteServerConfiguration response	Added request Added request Added response
			SetUsersConfiguration request, UsersConfiguration response	Removed remote server configuration
R24	15.03.2021	Evgeniy Vasyliev	PollForNewRequest message RequestPresent message ReadyToReceiveRequest message	Added message for communication with remote server and allowing it to get/set any data to PTS-2 controller

			GetRemoteServerConfiguration request SetRemoteServerConfiguration request RemoteServerConfiguration response	Removed "*Login*" and "*Password*" fields, added "*UserId*" field
R25	21.03.2021	Evgeniy Vasyliev	GetGpsData request GpsData response	Added GPS request and response when GPS module is applied
R26	06.04.2021	Evgeniy Vasyliev	UploadPumpTransaction message UploadTankMeasurement message	Added fields "*FuelGradeId*" and "*FuelGradeName*" Added fields "*Status*" and "*Alarms*"
R27	25.04.2021	Evgeniy Vasyliev	UploadPumpTransaction message UploadTankMeasurement message	Added field "*ConfigurationId*" to indicate whether configuration was changed
R28	15.05.2021	Evgeniy Vasyliev	ReportGetGpsRecords request, ReportGpsRecords response UploadGpsRecord message	Added request and responses
			Language response	Added Turkish language
R29	31.05.2021	Evgeniy Vasyliev	SetRemoteServerConfiguration request RemoteServerConfiguration response	Added "*ProtocolType*" field
	03.06.2021	Evgeniy Vasyliev	COMMUNICATION TO REMOTE SERVER	Added description of HTTP request and response
R30	07.06.2021	Evgeniy Vasyliev	ProbeMeasurements response ReportTankMeasurements response UploadTankMeasurements message	New alarms for tank added
R31	17.06.2021	Evgeniy Vasyliev	ProbeMeasurements response	Added "*PumpsDispensedVolume*" field to indicate how much volume was dispensed through pumps during last in-tank delivery
R32	15.07.2021	Evgeniy Vasyliev	GpsData response	Modified notes
			GetCpuTemperature request CpuTemperature response	Added request Added response
R33	25.09.2021	Evgeniy Vasyliev	SetTanksConfiguration request TanksConfiguration response	Added "*CriticalHighProductAlarmHeight*", "*CriticalLowProductAlarmHeight*" and "*HighWaterAlarmHeight*" fields
			Error message	Added "*Code*" field for error code, added *Data* object for error additional specification
			Error messages section	Error message descriptions reviewed
			All sections	Changed maximal number of pumps from 64 to 50, changed maximal number of probes and tanks from 32 to 50
R34	1.11.2021	Evgeniy Vasyliev	GetPriceBoardsConfiguration request SetPriceBoardsConfiguration request PriceBoardsConfiguration response FirmwareInformation response	Added support for price boards. Added requests and responses.
R35	7.11.2021	Evgeniy Vasyliev	GetReadersConfiguration request SetReadersConfiguration request ReadersConfiguration response FirmwareInformation response	Added support for readers. Added requests and responses.
R36	17.11.2021	Evgeniy Vasyliev	GetTagsList request SetTagsList request TagsList response GetTagInformation request TagInformation response ReaderGetTag request ReaderTag response Confirmation/error response message	Added support for tags. Added requests and responses. Added "*Type*" field in Confirmation message.
R37	04.12.2021	Evgeniy Vasyliev	SetTanksConfiguration request TanksConfiguration response	Added "*StopPumpsAtCriticalLowProductHeight*" field
R38	15.12.2021	Evgeniy Vasyliev	PumpTransactionInformation response	Added a new pump state "*WaitingNozzleUpForAuthorization*" in for cases when controller already received authorization, but is waiting for pump to take nozzle up before sending authorization command to it (applicable for cases when nozzle/grade is not specified in "*PumpAuthorize*" request or there is a pump parameter checked to send the authorization only on nozzle up).
R39	21.12.2021	Evgeniy Vasyliev	PriceBoardGetStatus request PriceBoardStatus response	Added requests and responses.

			ReaderGetStatus request ReaderStatus response	
R40	02.02.2022	Evgeniy Vasyliev	COMMUNICATION TO REMOTE SERVER section SetPortLoggingConfiguration request	Communication to a remote server updated, support for WebSocket Protocol added. Field "*DateTimeStop"* may not be present in request when stopping the log.
			SetRemoteServerConfiguration request RemoteServerConfiguration response	Updated request and response to support new features like HMAC, test of upload data, WebSocket communication.
R41	15.02.2022	Evgeniy Vasyliev	COMMUNICATION TO REMOTE SERVER section	UploadConfiguration request added. *"X-Pts-Id"* header added for all upload requests with unique identifier of the PTS-2 controller.
			UploadTest request	Added "*ConfigurationId"* field
			GetDailyProcessingTime request SetDailyProcessingTime request DailyProcessingTime response	Requests and response added.
R42	04.03.2022	Evgeniy Vasyliev	COMMUNICATION TO REMOTE SERVER section	Confirmation response with a single data setting request added.
R43	03.04.2022	Evgeniy Vasyliev	UploadTest request	Added information on found alarms and problems for registration and diagnostics remotely on server.
R44	27.04.2022	Evgeniy Vasyliev	SetRemoteServerConfiguration request RemoteServerConfiguration response	Added "*UploadTestRequestsPeriodSeconds*" and "*ServerResponseTimeoutSeconds*" fields.
R45	30.04.2022	Evgeniy Vasyliev	GetMeasurementUnits request MeasurementUnits response	Added temperature measurement units.
R46	07.05.2022	Evgeniy Vasyliev	PumpAuthorize request	Added field "*Tag"* to enable the control system to work with external tags reader.
			TagInformation response	Added field "*Present"* to show if the tag is present in tags list.
			PumpFillingStatus response, PumpEndOfTransactionStatus response, PumpTransactionInformation response	Added field "*Tag"* to show the tag used to authorized the pump.
R47	20.05.2022	Evgeniy Vasyliev	PumpGetAdditionalMeasurements request PumpAdditionalMeasurements response	Request and response added.
R48	10.06.2022	Evgeniy Vasyliev	AddUserConfiguration request EditUserConfiguration request DeleteUserConfiguration request	Requests added.
			PumpGetTransactionInformation request	Made field "*Transaction"* to be optional.
R49	14.06.2022	Evgeniy Vasyliev	AddTagToList request EditTagInList request DeleteTagFromList request	Requests added.
R50	30.06.2022	Evgeniy Vasyliev	GetUploadedRecordsInformation request UploadedRecordsInformation response ClearUploadedRecords request	Requests and responses added.
			GetTagsList request GetTagInformation request	Permissions reviewed.
R51	17.07.2022	Evgeniy Vasyliev	Language response	Added Ukrainian language.
R52	30.08.2022	Evgeniy Vasyliev	Diagnostics response	Added check of EXT and DEBUG ports.
R53	03.09.2022	Evgeniy Vasyliev	ReportGetInTankDeliveries request ReportInTankDeliveries response	Added request and response.
			SetRemoteServerConfiguration request RemoteServerConfiguration response	Field '*UseUploadTestRequests*' renamed to "*UploadStatus*", field "*UploadTestRequestsPeriodSeconds*" renamed to "*UploadStatusRequestsPeriodSeconds*". Field "*UploadInTankDeliveries*" added.
			UploadStatus request UploadedRecordsInformation response	"*UploadTest*" request renamed to "*UploadStatus*" All fields reviewed to allow realtime live monitoring of controller activity from a remote server.
			UploadInTankDelivery request	Request added.

			UploadPumpTransaction request UploadTankMeasurement request UploadGpsRecord request	Changed description: data does not contain arrays of objects, only 1 object is present in data.
			ClearUploadedRecords request	Field "*ClearInTankDeliveries*" added.
			COMMUNICATION TO REMOTE SERVER section	Added a confirmation response from the PTS-2 controller to confirmation response from a remote server with a single data setting request.
			SetDailyProcessingTime request DailyProcessingTime response	Added "*BackupConfiguration*" field.
R54	25.09.2022	Evgeniy Vasyliev	ReportGetAlertRecords request ReportAletRecords response	Added request and response.
			SetRemoteServerConfiguration request RemoteServerConfiguration response	Fields "*UploadAlertRecords*", "*UploadConfiguration*" added.
			UploadAlertRecord request	Request added.
			ClearUploadedRecords request	Field "*ClearAlertRecords*" added.
R55	29.09.2022	Evgeniy Vasyliev	UploadStatus request	Field "*SdState*" added.
			PumpCloseTransaction request	Notes updated.
R56	04.10.2022	Evgeniy Vasyliev	SetRemoteServerConfiguration request RemoteServerConfiguration response	Fields "*UploadPumpTransactionsUri*", "*UploadTankMeasurementsUri*", "*UploadInTankDeliveriesUri*", "*UploadGpsRecordsUri*", "*UploadAlertRecordsUri*", "*UploadConfigurationUri*", "*UploadStatusUri*" added. Field "*Uri*" removed.
R57	01.12.2022	Evgeniy Vasyliev	TagInformation response TagsList response SetTagsList request AddTagToList request EditTagInList request DeleteTagFromList request PumpIdleStatus response PumpFillingStatus response PumpEndOfTransactionStatus response PumpAuthorize request PumpTransactionInformation response PumpTag response ReaderTag response ReportPumpTransactions response UploadPumpTransaction request UploadStatus request	Field "*Tag*" extended to have up to 48 hexadecimal symbols.
R58	27.12.2022	Evgeniy Vasyliev	Diagnostics response	Added check of RTC (realtime clock), DEBUG port is now checked separately by connection it to the COM-port.
R59	15.01.2023	Evgeniy Vasyliev	ProbeMeasurement response, ReportTankMeasurements response, UploadTankMeasurement request, UploadStatus request	Added "*TankFillingPercentage*" field to inform percentage of the tank filling. Providing only measured values in responses, other values are not informed now.
R60	21.01.2023	Evgeniy Vasyliev	SetFuelGradesConfiguration request, FuelGradesConfiguration response	Added fields "*BlendTank1Id*", "*BlendTank2Id*", "*BlendTank1Percentage*" for configuration of the blended fuel grades.
R61	05.02.2023	Evgeniy Vasyliev	GetWirelessDevicesConfiguration request, SetWirelessDevicesConfiguration request, WirelessDevicesConfiguration response	Added requests and response.
R62	18.03.2023	Evgeniy Vasyliev	Language response	Added Spanish and Arabic languages.
R63	31.03.2023	Evgeniy Vasyliev	COMMUNICATION TO A REMOTE SERVER USING WEBSOCKET PROTOCOL section COMMUNICATION TO A REMOTE SERVER USING HTTP REQUESTS section	Added an individual description for each of the ways for communication to a remote server.
R64	30.04.2023	Evgeniy Vasyliev	PumpIdleStatus response PumpEndOfTransactionStatus response PumpTransactionInformation response	Added values for nozzle totals (when automatic readout of totals is configured in controller).
R65	14.05.2023	Evgeniy Vasyliev	SystemOperationInformation response	Added more indicators for heap memory allocations and frees.

			GetControllerType request, ControllerType response	Added requests and response.
			SetRemoteServerConfiguration request, RemoteServerConfiguration response	Added "*ProtocolType*" field for selection of used protocol between HTTP and HTTPS for the PTS-2 PRO controller.
R66	04.06.2023	Evgeniy Vasyliev	SetRemoteServerConfiguration request, RemoteServerConfiguration response	"*WebsocketsReconnectPeriod*" field removed as now WebSocket connection can be constant, no need to make its interruptions.
			COMMUNICATION TO A REMOTE SERVER USING WEBSOCKET PROTOCOL section	New firmware allows parallel communication of the PTS-2 controller with clients as a web-server and with remote server as a client using WebSocket protocol, so previous limitations were removed.
R67	11.06.2023	Evgeniy Vasyliev	PumpIdleStatus response, PumpFillingStatus response, PumpEndOfTransactionStatus response, PumpTransactionInformation response, PumpTotals response, PumpTag response, PumpDisplayData response, ProbeMeasurements response, ReportPumpTransactions response, ReportTankMeasurements response, ReportInTankDeliveries response, UploadTankMeasurement request, UploadInTankDelivery request	Added fields "*FuelGradeId*" ("*LastFuelGradeId*") and "*FuelGradeName*" ("*LastFuelGradeName*").
			UploadConfiguration request	Added "*ConfigurationId*" field.
R68	21.06.2023	Evgeniy Vasyliev	PumpTransactionInformation response	Added "*EndOfTransaction*" state to indicate that the filling is finished, but the transaction is still opened. "*DateTime*" field is added to response only when the filling has already finished.
			PumpEndOfTransactionStatus response	Added "*DateTimeStart*" and "*DateTime*" fields.
			PumpFillingStatus response	Added "*DateTimeStart*" field.
			PumpIdleStatus response	Added "*Nozzle*" field. Fields "*NozzleUp*" and "*Nozzle*" show the same value, field "*Nozzle*" was added for homogeneity with other requests/responses, field "*NozzleUp*" is left for compatibility with previous versions of this protocol. Added "*LastDateTimeStart*" and "*LastDateTime*" fields.
R69	05.07.2023	Evgeniy Vasyliev	PumpOfflineStatus response	Added more fields information to response in order to indicate existing status of the pump before it lost communication.
R70	08.07.2023	Evgeniy Vasyliev	UploadStatus request	Added fuel grades configuration.
R71	21.07.2023	Evgeniy Vasyliev	PumpIdleStatus response	Added "*Transaction*" field to indicate present transaction if the pump is authorized, but the filling has not started yet.
R72	22.07.2023	Evgeniy Vasyliev	ProbeMeasurements response, ReporrTankMeasurements response, ReportAlertRecords response, UploadTankMeasurement request, UploadAlertRecord request, UploadStatus request	Added tank leakage alert.
R73	22.08.2023	Evgeniy Vasyliev	COMMUNICATION TO A REMOTE SERVER USING WEBSOCKET PROTOCOL section COMMUNICATION TO A REMOTE SERVER USING HTTP REQUESTS section	Added a header named "*X-Pts-Firmware-Version-DateTime*" to show the PTS-2 controller firmware version date/time to each of the message sent to a remote server.
R74	08.09.2023	Evgeniy Vasyliev	COMMUNICATION TO A REMOTE SERVER USING WEBSOCKET PROTOCOL section COMMUNICATION TO A REMOTE SERVER USING HTTP REQUESTS section	Added a header named "*X-Pts-Configuration-Identifier*" used to point if configuration was changed anyhow.
R75	10.10.2023	Evgeniy Vasyliev	PumpIdleStatus response PumpOfflineStatus response	Added "*LastUser*" field.
R76	17.11.2023	Evgeniy Vasyliev	SetRemoteServerConfiguration request, RemoteServerConfiguration response	Added fields"*WebsocketsUploadPumpTransactions*", "*WebsocketsUploadTankMeasurements*", "*WebsocketsUploadInTankDeliveries*",

				"*WebsocketsUploadGpsRecords*", "*WebsocketsUploadAlertRecords*", "*WebsocketsUploadConfiguration*", "*WebsocketsUploadStatus*", "*WebsocketsUploadStatusRequestsPeriodSeconds*" in order to send the correspondent data automatically to a remote server when using WebSocket protocol.
			COMMUNICATION TO A REMOTE SERVER USING WEBSOCKET PROTOCOL section	Added support for upload request to be sent from the PTS-2 controller automatically to the remote server.
			REQUESTS UPLOADED TO A REMOTE SERVER AND RESPONSES FROM A REMOTE SERVER section	New section added: it describes the requests/responses sent to the remote server, which are the same for both types of communication with a remote server: using WebSocket protocol and using HTTP requests.
P77	15.12.2023	Evgeniy Vasyliev	PtsNetworkSettings response	Added "*UsedProtocolType*" and "*UsedAuthenticationType*" fields.
			PortLoggingConfiguration response	Added "*LogIsRunning*" field.
P78	11.01.2024	Evgeniy Vasyliev	ReportGetPumpTransactions request	Added "*Transaction*" field.
P79	15.02.2024	Evgeniy Vasyliev	GetTagsList request	Added "*StartNumber*" and "*TotalNumber*" fields.
			GetTagsTotalNumber request TagsTotalNumber response	Request and response added.
			AddTagsToList request	Request added.
			SetTagsList request	Notes added.
P80	21.02.2024	Evgeniy Vasyliev	PumpAuthorize request	Added "*AuthorizeWithoutTag*" and "*NoTagVerification*" fields.
P81	10.03.2024	Evgeniy Vasyliev	ProbeGetTankCalibrationChartTotalRecordsNumber request ProbeTankCalibrationChartTotalRecordsNumber response ProbeGetTankCalibrationChartRecordsList request ProbeTankCalibrationChartRecordsList response ProbeSetTankCalibrationChartRecordsList request ProbeAddTankCalibrationChartRecordsList request	Requests and responses added.
P82	09.04.2024	Evgeniy Vasyliev	UploadPumpTransaction request	Added "*Tank*" field.
			PumpSetAutomaticOperation request	Added "*AutoCloseTransaction*" field
			PumpIdleStatus response	Added "*NozzlePrices*" field
			SetUploadedRecordsNumber request	Request added.
P83	26.06.2024	Evgeniy Vasyliev	SetDailyProcessingTime request, DailyProcessingTime response	Added "*ProcessFiles*" field.
P84	15.07.2024	Evgeniy Vasyliev	ProbeMeasurement response	Added "*DateTime*" field.
P85	18.07.2024	Evgeniy Vasyliev	PumpIdleStatus response	Added fields "*LastReceivedTotalNozzle*", "*LastReceivedTotalVolume*" and "*LastReceivedTotalAmount*".
P86	25.07.2024	Evgeniy Vasyliev	PumpAdditionalMeasurements response	Added "*CurrentDensity*" field.
P87	09.08.2024	Evgeniy Vasyliev	ProbeGetTankIntervalVolumeChartTotalRecordsNumber request ProbeTankIntervalVolumeChartTotalRecordsNumber response ProbeGetTankIntervalVolumeChartRecordsList request ProbeTankIntervalVolumeChartRecordsList response ProbeGetTankAutomaticCalibrationChartTotalRecordsNumber request ProbeTankAutomaticCalibrationChartTotalRecordsNumber response ProbeGetTankAutomaticCalibrationChartRecord	Added new requests and responses to support automatic tanks calibration.

			sList request ProbeTankAutomaticCalibrationChartRecordsList response ProbeGenerateTankAutomaticCalibrationChart request	
			SetDailyProcessingTime request DailyProcessingTime response	Added "*RegenerateTankAutoCalibrationCharts*" field.
P88	27.08.2024	Evgeniy Vasyliev	PumpLock request	Added request.
			PumpAuthorize request	Added "*NotApplyUser*" field.
P89	05.09.2024	Evgeniy Vasyliev	ProbeMeasurements response	Added "*InTankDeliveryDetected*" field.
P90	10.09.2024	Evgeniy Vasyliev	PumpEndOfTransactionStatus response	Added "*NozzleUp*" field.
P91	19.09.2024	Evgeniy Vasyliev	PumpUnlock request	Added request.
P92	04.10.2024	Evgeniy Vasyliev	ProbeAddTankCalibrationChartRecordToList request, ProbeEditTankCalibrationChartRecordInList request, ProbeDeleteTankCalibrationChartRecordFromList request	Added requests.
P93	06.10.2024	Evgeniy Vasyliev	GetPortsState request PortsState response	Added request and response
P94	19.10.2024	Evgeniy Vasyliev	SetProbesConfiguration request, ProbesConfiguration response SetPriceBoardsConfiguration request, PriceBoardsConfiguration response SetReadersConfiguration request, ReadersConfiguration response SetWirelessDevicesConfiguration request, WirelessDevicesConfiguration response Diagnostics response, SetPortLoggingConfiguration request, PortLoggingConfiguration response	Added "*PC*" port
P95	04.11.2024	Evgeniy Vasyliev	RemoteServerConfiguration response	Added "*IsConnectionSuccessful*" field.
P96	11.11.2024	Evgeniy Vasyliev	FirmwareInformation response	Added "*BootloaderVersion*" field.
			ControllerType response	Updated response values.
			UploadStatus request	Added "*LastTotalVolumes*" and "*LastTotalAmounts*" fields.
P97	23.11.2024	Evgeniy Vasyliev	MakeDailyProcessing request	Added request.
P98	29.11.2024	Evgeniy Vasyliev	GetPaymentFormsConfiguration request, SetPaymentFormsConfiguration request, PaymentFormsConfiguration response	Added requests and response.
			PumpAuthorize request	Added "*PaymentFormId*" field.
			UploadPumpTransaction request	Added "*PaymentFormId*" and "*PaymentFormName*" fields.
			PumpCloseTransaction request	Added "*PaymentFormId*" field.
			ReportPumpTransactions response	Added "*PaymentFormId*" and "*PaymentFormName*" fields.
			PumpTransactionInformation response	Added "*PaymentFormId*" and "*PaymentFormName*" fields.
			UploadStatus request	Added "*LastPaymentFormIds*" and "*PaymentFormIds*" fields.
			SetPumpNozzlesConfiguration request, PumpNozzlesConfiguration response	Added "*PaymentFormIds*" field.
			PumpIdleStatus response, PumpOfflineStatus response	Added "*LastPaymentFormId*" and "*LastPaymentFormName*" fields.
			PumpFillingStatus response, PumpEndOfTransactionStatus response	Added "*PaymentFormId*" and "*PaymentFormName*" fields.
P99	01.12.2024	Evgeniy Vasyliev	All requests and responses	Corrected a maximal number of devices to be configured:

				- for pumps: up to 120 - for probes/tanks: up to 20 - for price boards: up to 5 - for readers: up to 120 Length of field "*Tag*" changed to have up to 32 hexadecimal symbols.
			PumpGetLastSavedTotals request, PumpLastSavedTotals response	Added request and response.
			RequestTagBalance request, TagBalance response	Added request and response.
P100	05.12.2024	Evgeniy Vasyliev	TagInformation response	Added "*PaymentFormId*" and "*PaymentFormName*" fields.
			TagsList response, SetTagsList request, AddTagsToList request, AddTagToList request, EditTagInList request	Added "*PaymentFormId*" field.
P101	09.12.2024	Evgeniy Vasyliev	SetPaymentFormsConfiguration request, PaymentFormsConfiguration response	Added "*AutoCloseTransaction*" field.
P102	14.12.2024	Evgeniy Vasyliev	GetPricesSchedulerConfiguration request, SetPricesSchedulerConfiguration request, PricesSchedulerConfiguration response	Added requests and response.
P103	07.01.2025	Evgeniy Vasyliev	GetFuelGradesPrices request, SetFuelGradesPrices request, FuelGradesPrices response	Added requests and response.
P104	12.03.2025	Evgeniy Vasyliev	SetRemoteServerConfiguration request, RemoteServerConfiguration response	Added "*VerifyTags*", "*VerifyTagsUri*", "*WebsocketsVerifyTags*" fields.
P105	01.04.2025	Evgeniy Vasyliev	RequestTagsInformation request TagsInformation response	RequestTagBalance request renamed to RequestTagsInformation. Format and list of fields updated. TagBalance response renamed to TagsInformation. Format and list of fields updated.
			SetRemoteServerConfiguration request, RemoteServerConfiguration response	Field "*VerifyTags*" renamed to "*RequestTagsInformation*", "*VerifyTagsUri*" renamed to "*RequestTagsInformationUri*", "*WebsocketsVerifyTags*" renamed to "*WebSocketsRequestTagsInformation*".
			PumpAuthorize request	Added "*VerifyTagOnServer*" field.
P106	04.04.2025	Evgeniy Vasyliev	RequestTagsInformation request	Added "*FuelGradeName*" and "*Price*" fields.
P107	05.04.2025	Evgeniy Vasyliev	PumpAuthorize request	Field "*VerifyTagOnServer*" renamed to "*RequestTagInformationOnServer*".
P108	19.04.2025	Evgeniy Vasyliev	UsersConfiguration response	Presence "*Permissions*" objects for each of the users in response depends on presence of "*Configuration*" permission for the user sending GetUsersConfiguration request: if the user has this permission – then this user receives a list of permissions of each of the users, otherwise no.
P109	21.04.2025	Evgeniy Vasyliev	ALERT CODES response	Added alert for pump: filling in offline mode, updated codes for pump nozzles.
			ReportAlertRecords response	Added "*Detected*" state.
			UploadAlertRecord request	Added "*Detected*" state.
P110	26.04.2025	Evgeniy Vasyliev	GetRemoteServerLoggingConfiguration request, SetRemoteServerLoggingConfiguration request, RemoteServerLoggingConfiguration response	Added requests and response.
P111	04.05.2025	Evgeniy Vasyliev	ProbeTankIntervalVolumeChartRecordsList response	Added "*PassesNumber*" field.
P112	13.06.2025	Evgeniy Vasyliev	SetReadersConfiguration request, ReadersConfiguration request	Added "*NozzleNumberForNozzleReader*" and "*NozzleReaderIdentifier*" field.
P113	07.07.2025	Evgeniy Vasyliev	PaymentCreateTransaction request, PaymentCreateTransactionConfirmation response, PaymentGetTransactionInformation request, PaymentTransactionInformation request, PaymentRefundTransaction request,	Added requests and responses for PAYMENTS REQUESTS AND RESPONSES and SHIFT MANAGEMENT REQUESTS AND RESPONSES sections.

			ShiftOpen request, ShiftOpenConfirmation response, ShiftClose request, ShiftGetInformation request, ShiftInformation response	
			ReportPumpTransactions response, PumpTransactionInformation response, UploadPumpTransaction request	Added "*IsOffline*", "*IsPaid*" and "*FlowRate*" fields. Removed "*PaymentFormId*" and "*PaymentFormName*" fields.
			SetUsersConfiguration request, UsersConfiguration response, UserInformation response, AddUserConfiguration request EditUserConfiguration request	Added "*Payments*" and "*Shifts*" permissions.
			PumpAuthorize request, PumpCloseTransaction request	Removed "*PaymentFormId*" field.
			UploadStatus request	Removed "*PaymentFormIds*" and "*LastPaymentFormIds*" fields. Added "*FlowRate*" and "*LastFlowRate*" fields.
			PumpIdleStatus response, PumpOfflineStatus response	Removed "*LastPaymentFormId*" and "*LastPaymentFormName*" fields. Added "*FlowRate*" and "*LastFlowRate*" fields.
			PumpFillingStatus response, PumpEndOfTransactionStatus response	Removed "*PaymentFormId*" and "*PaymentFormName*" fields. Added "*FlowRate*" field.
			TagsInformation response	Removed "*PaymentFormId*" field.
			ReportGetPayments request, ReportPayments response, ReportGetShifts request, ReportShifts response	Added requests and response.
P114	17.08.2025	Evgeniy Vasyliev	UploadedRecordsInformation response, SetUploadedRecordsNumber request, ClearUploadedRecords request, SetRemoteServerConfiguration request, RemoteServerConfiguration response	Added fields of payments and working shifts.
			UploadPayment request, UploadShift request	Added requests.
			SetPaymentFormsConfiguration request, PaymentFormsConfiguration response	Removed "*AutoCloseTransaction*" field.
			PumpGetTransactionInformation request	Limited the request to process last 1000 records in database to avoid long processing of the request.
			GetShiftsOperation request, SetShitsOperation request, ShiftsOperation response	Added requests and response.
			PumpFillingStatus response, PumpOfflineStatus response, PumpTransactionInformation response	Added "*IsSuspended*" field.
			ShiftInformation response	Added "*Enabled*" field.
P125	19.09.2025	Evgeniy Vasyliev	PtsNetworkSettings response	Added "*MAC*" field.
P126	13.10.2025	Evgeniy Vasyliev	BatteryVoltage response	Added "*State*" field.
			ProbeMeasurements response	Added fields "*ProducVolumeFloat*", "*WaterVolumeFloat*", "*ProductUllage*", "*ProductTCVolumeFloat*".
P127	27.10.2025	Evgeniy Vasyliev	PaymentTransactionInformation response	Format updated.
P128	31.10.2025	Evgeniy Vasyliev	ShiftOpenConfirmation response, ShiftGetInformation request, ShiftInformation response, ReportShifts response, UploadShift request	Field "*ShiftNumber*" renamed to "*Number*".
			PumpFillingStatus response, PumpOfflineStatus response, PumpTransactionInformation response	Added "*OrderedType*" and "*OrderedDose*" fields.
			PumpIdleStatus response, PumpOfflineStatus response, PumpTransactionInformation response	Added "*OrderedNozzle*", "*OrderedNozzles*", "*OrderedType*" and "*OrderedDose*" fields.

P129	04.11.2025	Evgeniy Vasyliev	ShiftInformation response	If field "*IsEnabled*" has value *false* then other fields are absent.
			UploadStatus request	Added "*ShiftInformation*" field

INTRODUCTION AND SCOPE

This document is intended to be used by OEMs who wish to interface to the PTS-2 forecourt controller. The PTS-2 forecourt controller Ethernet link allows master devices to remotely control operation of petrol, diesel, LPG and CNG dispensers and automatic tank gauge systems.

Technotrade LLC hereby permits reproduction of this document as may be required by any of our customers or OEMs wishing to use it.

This document has been carefully prepared and is believed to be accurate. However, Technotrade LLC, its employees and its agents do not assume responsibility for its use either directly or indirectly. Technotrade LLC reserves a right to make changes to this document at any time without notice. Prospective users of this document should contact Technotrade LLC at the time they wish to implement this protocol on their products to become aware of any updates that may apply.

Complete information regarding the PTS-2 forecourt controller can be found on its web-page: *http://www.technotrade.ua/pts2-forecourt-controller.html*.

All technical questions regarding the PTS-2 forecourt controller are welcome to be asked on support mailbox: *support@technotrade.ua*. Our support team will be glad to help you.

In case if you find any mistakes, omissions in this document or have any suggestions on improvements to this document, please feel free to e-mail them our support mailbox: *support@technotrade.ua*. We will be thankful to you for this valuable information.

HTTP LAYER OF COMMUNICATION

jsonPTS communication protocol is based on HTTP protocol version 1.1 and messages are always transmitted using POST requests.

Generally, at making a request the HTTP header should contain following minimum data (path */jsonPTS* is register dependent and should be written as shown here):

```
POST /jsonPTS HTTP/1.1
Host: IP-address
Accept: */*
Authorization: type-of-authorization
Content-Length: number-of-bytes
Content-Type: application/json; charset=utf-8
(other optional request headers)

<jsonPTS protocol request>
```

At response HTTP headers should contains following minimum data:

```
HTTP/1.1 200 OK
Content-Length: number-of-bytes
Content-Type: application/json; charset=utf-8
Connection: close

<jsonPTS protocol response>
```

Responses can also be sent in chunks (*https://en.wikipedia.org/wiki/Chunked_transfer_encoding*):

```
HTTP/1.1 200 OK
Transfer-Encoding: chunked
Content-Type: application/json; charset=utf-8
Connection: close

chunk-1-number-of-bytes
<jsonPTS protocol response chunk 1>

chunk-2-number-of-bytes
<jsonPTS protocol response chunk 2>

chunk-3-number-of-bytes
<jsonPTS protocol response chunk 3>

...

0
```

Input buffer has 16 KB length.

HTTPS LAYER OF COMMUNICATION

Communication to the PTS-2 forecourt controller as to the web server can be done through TLS layer (TLS 1.2), for this DIP-1 switch should be set to OFF position at the controller board on startup. At this the PTS-2 web server is using self-signed certificate. If the DIP-1 switch is set to ON position – then TLS layer is not applied and all communication is done using non-secured HTTP.

Communication of the PTS-2 controller as a web client to a remote web server can be done through TLS layer (TLS 1.2), for this appropriate configuration of the communication with the remote server is be set using SetRemoteServerConfiguration request.

AUTHENTICATION

PTS-2 controller web-server can use one of the following authentication types:

- digest (*https://en.wikipedia.org/wiki/Digest_access_authentication*)
- basic (*https://en.wikipedia.org/wiki/Basic_access_authentication*)

Authentication type is selected using switch 2 of configuration DIP-switch on board of the PTS-2 controller:

- OFF position: digest authentication
- ON position: basic authentication

PTS-2 controller accepts a request only if it contains authentication information of a user, which is valid and present in configuration of PTS-2 controller:

- if there is no authentication information in the received request – the PTS-2 controller returns a response with HTTP status code 401 (Unauthorized)
- if the user does not have rights to perform requested action – the PTS-2 controller returns a response with HTTP status code 403 (Forbidden)

JSON MESSAGE STRUCTURE

General structure of JSON request (on example for 3 requests: GetDateTime, SetDateTime and non-existing request):

```
{
    "Protocol": "jsonPTS",
    "Packets": [{
            "Id": 1,
            "Type": "GetDateTime"
        },{
            "Id": 2,
            "Type": "SetDateTime",
            "Data": {
                "DateTime": "2021-01-01T12:34:56",
                "AutoSynchronize": true,
                "UTCOffset": 120
            }
        },{
            "Id": 3,
            "Type": "MakeNonExistingRequest"
        }
    ]
}
```

The above message means 3 requests:

- `GetDateTime` request is used for getting date and time used in PTS-2 controller
- `SetDateTime` request is used for setting date and time in PTS-2 controller
- `MakeNonExistingRequest` request is used for demonstration of non-existing request

Multiple requests can be united in a single JSON message, requests represent packets in a message. PTS-2 controller should accept and correctly response to JSON requests containing up to 600 tokens.

Fields used in the request are the following:

- `Protocol` – specifier of used communication protocol (always has value “jsonPTS”)
- `Packets` – array of packet objects (requests), each packet is an object and has the following fields:
  - `Id` – unique identifier of packet in message (integer, range from 1 till 2147483647)
  - `Type` – specifier of request type
  - `Data` – specific data fields corresponding to the type of request, field is optional and is present only in requests, where data is needed

General structure of JSON response (on example on response for above stated request message):

```
{
      "Protocol": "jsonPTS",
      "Packets": [{
            "Id": 1,
            "Type": "DateTime",
            "Data": {
                "DateTime": "2021-01-01T12:34:56",
                "AutoSynchronize": true,
                "UTCOffset": 120
            }
        }, {
            "Id": 2,
            "Type": "SetDateTime",
            "Message": "OK"
        }, {
            "Id": 3,
            "Type": "MakeNonExistingRequest",
            "Error": true,
            "Code": 1,
            "Message": "JSONPTS_ERROR_NOT_FOUND"
        }
    ]
}
```

Multiple responses can be united in a single JSON message, responses represent packets in a message.

Fields used in the response are the following:

- `Protocol` – specifier of used communication protocol (always has value “jsonPTS”)
- `Error` – Boolean value meaning presence/absence of error in request message, field is optional and is present in case of an error
- `Code` – integer value for error code, field is optional and is present in case of an error
- `Message` – specifies description of the error, field is optional and is present in case of an error
- `Packets` – array of packet objects (responses), each packet is an object and has the following fields:
  - `Id` – identifier of the packet in message, the value equals to the value of `Id` field from request (integer, range from 1 till 2147483647)
  - `Error` – Boolean value meaning presence/absence of error in request packet, field is optional and present in case of any error
  - `Code` – integer value for error code, field is optional and is present in case of an error
  - `Message` – specifies description of the error in case of any error, also specifies successful execution of setter request (value “OK”), field is optional and present in case of any error or at successful response to setter request
  - `Type` – specifier of the response
  - `Data` – specific data fields corresponding to the type of response

CONFIRMATION/ERROR RESPONSE MESSAGE

PTS-2 controller responses with confirmation response message or error message to requests requiring update of configuration or any action. Format of the response is the following:

Confirmation response message example:

```
{
    "Protocol": "jsonPTS",
    "Packets": [{
            "Id": 1,
            "Type": "RequestMessageType",
            "Message": "OK"
        }
    ]
}
```

Error response message example:

```
{
    "Protocol": "jsonPTS",
    "Packets": [{
            "Id": 1,
            "Type": "RequestMessageType",
            "Error": true,
            "Code": 1,
            "Message": "JSONPTS_ERROR_NOT_FOUND"
        }
    ]
}
```

In error message values of fields "*Code*" and "*Message*" depend on the error found. For possible list of error codes and messages please see Error messages section.

Some errors may contain additional data, which helps to fully describe the problem. This additional data is located in "*Data*" object. Presence and types of field in "*Data*" object depend on the error and are described in Error messages section. Example of error response message with additional fields:

```
{
    "Protocol": "jsonPTS",
    "Packets": [{
            "Id": 1,
            "Type": "RequestMessageType",
            "Error": true,
            "Code": 57,
            "Message": "JSONPTS_ERROR_BUSY_OTHER_USER",
            "Data": {
                "Pump": 1,
                "User": "admin"
            }
        }
    ]
}
```

GENERAL NOTES FOR COMMUNICATION

1. PTS-2 controller serves as a web-server for incoming requests and responses only upon reception of requests from a client.

2. PTS-2 controller can also serve as a web-client for connection to a remote server if it is configured for this (see COMMUNICATION TO A REMOTE SERVER USING WEBSOCKET PROTOCOL and COMMUNICATION TO A REMOTE SERVER USING HTTP REQUESTS sections).

3. Decimal point is used as a separator between integer and decimal parts in float numbers.

GENERAL REQUESTS AND RESPONSES

This part describes requests and responses used for getting information from PTS-2 controller.

1. GetBatteryVoltage request

Purpose	Gets battery voltage
Data	Absent
Permission required	Any permission among listed below: – *Configuration* – *Control* – *Monitoring* – *Reports* – *Payments* – *Shifts*
Response	BatteryVoltage response

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"GetBatteryVoltage"
	}]
	}

2. BatteryVoltage response

Purpose	Returns battery voltage
Data	"*Voltage*" – value of battery voltage in millivolts (mV) DC (integer) "*State*" – state of the battery voltage (string, 3 ASCII characters), possible values: – "*OK*" – battery voltage is good – "*Low*" – battery voltage is low, needs replacement – "*Bad*" – battery voltage is too low or batter is not found

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"BatteryVoltage",
	"Data":{
	"Voltage":2950,
	"State":"OK"
	}
	}]
	}

3. GetCpuTemperature request

Purpose	Gets CPU temperature
Data	Absent
Permission required	Any permission among listed below: – *Configuration* – *Control* – *Monitoring* – *Reports* – *Payments* – *Shifts*
Response	CpuTemperature response

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"GetCpuTemperature"
	}]
	}

4. CpuTemperature response

Purpose	Returns CPU temperature
Data	"*Temperature*" – value of CPU temperature in degrees Celsius (°C) (integer)

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"CpuTemperature",
	"Data":{
	"Temperature":37
	}
	}]
	}

5. GetUniqueIdentifier request

Purpose	Gets controller unique identifier
Data	Absent
Permission required	Any permission among listed below: – *Configuration* – *Control* – *Monitoring* – *Reports* – *Payments* – *Shifts*
Response	UniqueIdentifier response

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"GetUniqueIdentifier"
	}]
	}

6. UniqueIdentifier response

Purpose	Returns controller unique identifier
Data	"*ID*" – device unique identifier (string, up to 24 hexadecimal digits)

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"UniqueIdentifier",
	"Data":{
	"Id":"003A003A3435510938393730"
	}
	}]
	}

7. GetControllerType request

Purpose	Gets controller type
Data	Absent
Permission required	Any permission among listed below: – *Configuration* – *Control* – *Monitoring* – *Reports* – *Payments* – *Shifts*
Response	UniqueIdentifier response

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"GetControllerType"
	}]
	}

8. ControllerType response

Purpose	Returns controller unique identifier
Data	"*Type*" – controller type (string, possible values: “PTS-2 F427”, “PTS-2 PRO” and “PTS-2 H753”)

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"ControllerType",
	"Data":{
	"Type":"PTS-2 F427"
	}
	}]
	}

9. GetFirmwareInformation request

Purpose	Gets information on current firmware: version, release date and time, list of supported communication protocols
Data	Absent
Permission required	Any permission among listed below: – *Configuration* – *Control* – *Monitoring* – *Reports* – *Payments* – *Shifts*
Response	FirmwareInformation response

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"GetFirmwareInformation"
	}]
	}

10. FirmwareInformation response

Purpose	Returns information on current firmware: version, release date and time, list of supported communication protocols
Data	"*BootloaderVersion*" – bootloader version (string, 15 symbols) "*DateTime*" – firmware release date and time in format *YYYY-MM-DDThh:mm:ss* (string, 19 symbols), where: – *YYYY* – year (range from 2000 to 2099) – *MM* – month (range from 01 to 12) – *DD* – day (range from 01 to 31) – *T* – separator between date and time fields – *hh* – hours (range from 00 to 23) – *mm* – minutes (range from 00 to 59) – *ss* – seconds (range from 00 to 59) "*PumpProtocols*" – array of integers, which mean identifiers of pumps' protocols (each element is integer, range from 1 to 99) "*ProbeProtocols*" – array of integers, which mean identifiers of probes' protocols (each element is integer, range from 1 to 99) "*PriceBoardProtocols*" – array of integers, which mean identifiers of price boards' protocols (each element is integer, range from 1 to 99) "*ReaderProtocols*" – array of integers, which mean identifiers of readers' protocols (each element is integer, range from 1 to 99) "*IsDebug*" – debug version of the firmware (Boolean, possible values: "*true*", "*false*")
Note	Field "*BootloaderVersion*" is optional and is present only in firmware versions starting from 2024.11.11.

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"FirmwareInformation",
	"Data":{
	"BootloaderVersion":"743.00001.UA.0.1",
	"DateTime":"2024-11-12T08:05:41",
	"PumpProtocols":[1,2,3],
	"ProbeProtocols":[1,2,3],
	"PriceBoardProtocols":[1,2,3],
	"ReaderProtocols":[1,2],
	"IsDebug":false
	}
	}]
	}

11. GetSdInformation request

Purpose	Gets information about SD flash disk
Data	Absent
Permission required	Any permission among listed below: – *Configuration* – *Control* – *Monitoring* – *Reports* – *Payments* – *Shifts*
Response	SdInformation response

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"GetSdInformation"
	}]
	}

12. SdInformation response

Purpose	Returns information about SD flash disk
Data	"*FreeMemoryKB*" – free memory space on SD flash disk in kilobytes (integer) "*TotalMemoryKB*" – total memory space on SD flash disk in kilobytes (integer) "*Files*" – array of objects for each file in SD flash disk root, each containing: – "*Name*" – file name (string, up to 12 ASCII characters in 8.3 format) – "*Size*" – file size in bytes (integer)

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"SdInformation",
	"Data":{
	"FreeMemoryKB":7780544,
	"TotalMemoryKB":7781376,
	"Files":[
	{"Name":"PtsLog.txt", "Size":"12205"},
	{"Name":"PortLog.bin", "Size":"5445"},
	{"Name":"PumpTrn.txt", "Size":"1232"},
	{"Name":"TankM.txt", "Size":"52344"},
	{"Name":"SD_test.txt", "Size":"123"}
	]
	}
	}]
	}

13. FileDelete request

Purpose	Deletes a specified file from SD flash disk
Data	"*Name*" – file name (string, up to 12 ASCII characters in 8.3 format)
Permission required	Any permission among listed below: – *Configuration*
Response	Confirmation response message (in case of successful execution) or error response message (in case of any problem)

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"FileDelete",
	"Data":{
	"Name":"PtsLog.txt"
	}
	}]
	}

14. GetSystemOperationInformation request

Purpose	Gets information about system operation
Data	Absent
Permission required	Any permission among listed below: – *Configuration* – *Control* – *Monitoring* – *Reports* – *Payments* – *Shifts*
Response	SystemOperationInformation response

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"GetSystemOperationInformation"
	}]
	}

15. SystemOperationInformation response

Purpose	Returns information about system operation
Data	"*SystemUpTime*" – total system up time in milliseconds (integer) "*Tasks*" – array of objects for each task running, each containing: – "*Name*" – task name (string, up to 20 ASCII characters) – "*Priority*" – task priority (integer) – "*State*" – task state (string, up to 20 ASCII characters) – "*StackHighWaterMark*" – minimum amount of stack space that has remained for the task since the task was created, the closer this value is to zero the closer the task has come to overflowing its stack (integer) "*AvailableHeapFreeSize*" – available heap space in bytes (integer) "*MinimalEverHeapFreeSize*" – minimum ever free bytes remaining in heap space (integer) "*NumberOfFreeBlocks*" – number of free blocks in heap space (integer) "*NumberOfSuccessfulAllocations*" – number of successful memory allocations from heap (integer) "*NumberOfSuccessfulFrees*" – number of successful memory frees from heap (integer) "*SizeOfLargestFreeBlockInBytes*" – size of the largest free block in bytes (integer) "*SizeOfSmallestFreeBlockInBytes*" – size of smallest free block in bytes (integer)
Note	Some of the values relates to heap memory depend on the controller version and might be absent in response.

Example	{
	"Protocol": "jsonPTS",
	"Packets": [
	{
	"Id": 1,
	"Type": "SystemOperationInformation",
	"Data": {
	"SystemUpTime": 4500,
	"Tasks": [
	{
	"Name": "WebServerTask",
	"Priority": 3,
	"State": "Running",
	"StackHighWaterMark": 111,
	"RunTimeCounter": 0
	},
	{
	"Name": "IDLE",
	"Priority": 0,
	"State": "Ready",
	"StackHighWaterMark": 108,
	"RunTimeCounter": 0

```
            }
          ],
          "AvailableHeapFreeSize": 51864,
          "MinimalEverHeapFreeSize": 45312,
          "NumberOfFreeBlocks": 6,
          "NumberOfSuccessfulAllocations": 34207,
          "NumberOfSuccessfulFrees": 34083,
          "SizeOfLargestFreeBlockInBytes": 48312,
          "SizeOfSmallestFreeBlockInBytes": 32
        }
      }
    ]
}
```

16. GetUserInformation request

Purpose	Gets current user information
Data	Absent
Permission required	Any permission among listed below: – *Configuration* – *Control* – *Monitoring* – *Reports* – *Payments* – *Shifts*
Response	UserInformation response

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"GetUserInformation"
	}]
	}

17. UserInformation response

Purpose	Returns current user information
Data	"*Id*" – identifier of user (integer, range from 1 to 30) "*Login*" – user login (string, up to 10 ASCII characters) "*Permissions*" – object with user permissions: – "*Configuration*" – permission to provide configuration (Boolean, possible values: "*true*", "*false*") – "*Control*" – permission to provide control (Boolean, possible values: "*true*", "*false*") – "*Monitoring*" – permission to monitor pumps and tanks (Boolean, possible values: "*true*", "*false*") – "*Reports*" – permission to view reports (Boolean, possible values: "*true*", "*false*") – "*Payments*" – permission to perform payments (Boolean, possible values: "*true*", "*false*") – "*Shifts*" – permission to perform shift management (Boolean, possible values: "*true*", "*false*")

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"UserInformation",
	"Data":{
	"Id":1,
	"Login":"admin",
	"Permissions":{
	"Configuration":true,
	"Control":true,
	"Monitoring":true,
	"Reports":true,
	"Payments":true,
	"Shifts":true
	}
	}
	}]
	}

18. GetTagInformation request

Purpose	Returns configured tag
Data	"*Tag*" – value of tag ID (string, up to 32 hexadecimal symbols)
Permission required	Any permission among listed below: – *Configuration* – *Control*
Response	TagInformation response

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"GetTagInformation",
	"Data":{
	"Tag":"0123456789ABCDEF"
	}
	}]
	}

19. TagInformation response

Purpose	Returns a configured tag information
Data	"*Tag*" – value of tag ID (string, up to 32 hexadecimal symbols) "*Name*" – name of person, to whom the tag belongs (string, up to 20 ASCII symbols) "*Valid*" – permission for operation (Boolean, possible values: "*true*", "*false*") "*Present*" – presence in a tag list (Boolean, possible values: "*true*", "*false*") "*PaymentFormId*" – identifier of payment form (integer, range from 0 to 10) "*PaymentFormName*" – name of payment form (string, up to 10 ASCII characters)
Note	Fields "*PaymentFormId*" and "*PaymentFormName*" are optional, they are present in response only when there is a payment form set for the requested tag.

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"TagInformation",
	"Data":{
	"Tag":"1234567890ABCDEF",
	"Name":"Marry Plum",
	"Valid":true,
	"PaymentFormId":1,
	"PaymentFormName":"Cash",
	"Present":true
	}
	}]
	}

20. GetTagsTotalNumber request

Purpose	Gets total number of configured tags from controller
Data	Absent
Permission required	Any permission among listed below: – *Configuration* – *Control* – *Reports*
Response	TagsTotalNumber response

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"GetTagsTotalNumber"
	}]
	}

21. TagsTotalNumber response

Purpose	Returns total number of configured tags
Data	"*TotalNumber*" – total number of configured tags (integer, range from 0 and higher)

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"TagsTotalNumber",
	"Data":{
	"TotalNumber":123
	}
	}]
	}

22. GetLanguage request

Purpose	Gets language used for graphical user interface
Data	Absent
Permission required	Any permission among listed below: – *Configuration* – *Control* – *Monitoring* – *Reports* – *Payments* – *Shifts*
Response	Language response

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"GetLanguage"
	}]
	}

23. Language response

Purpose	Returns language used for graphical user interface
Data	"*Language*" – language (string, 2 ASCII characters), possible values: – "*EN*" – English language – "*RU*" – Russian language – "*UK*" – Ukrainian language – "*ES*" – Spanish language – "*AR*" – Arabic language

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"Language",
	"Data":{
	"Language":"EN"
	}
	}]
	}

24. GetMeasurementUnits request

Purpose	Gets measurement units used for graphical user interface
Data	Absent
Permission required	Any permission among listed below: – *Configuration* – *Control* – *Monitoring* – *Reports* – *Payments* – *Shifts*
Response	MeasurementUnits response

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"GetMeasurementUnits"
	}]
	}

25. MeasurementUnits response

Purpose	Returns measurement units used for graphical user interface
Data	"*Volume*" – volume units (char, 1 ASCII character), possible values: – "*L*" – liters – "*G*" – Gallons "*Temperature*" – temperature units (char, 1 ASCII character), possible values: – "*C*" – degrees Celcius (°C) – "*F*" – degrees Fahrenheit (°F)

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"MeasurementUnits",
	"Data":{
	"Volume":"L",
	"Temperature":"C",
	}
	}]
	}

26. Restart request

Purpose	Restarts the PTS-2 controller
Data	Absent
Permission required	Any permission among listed below: – *Configuration*
Response	Confirmation response message (in case of successful execution) or error response message (in case of any problem)

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"Restart"
	}]
	}

CONFIGURATION REQUESTS AND RESPONSES

This part describes requests and responses used for getting and setting configuration in PTS-2 controller.

27. GetConfigurationIdentifier request

Purpose	Gets configuration unique identifier, which is used to point if configuration was changed anyhow: given identifier is assigned a new unique value at any change of configuration, thus an application may know if the configuration was changed based on value of this identifier.
Data	Absent
Permission required	Any permission among listed below: – *Configuration* – *Control* – *Monitoring*
Response	ConfigurationIdentifier response

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"GetConfigurationIdentifier"
	}]
	}

28. ConfigurationIdentifier response

Purpose	Returns configuration unique identifier, which is used to point if configuration was changed anyhow: given identifier is assigned a new unique value at any change of configuration, thus an application may know if the configuration was changed based on value of this identifier.
Data	"*Id*" – unique configuration identifier (string, up to 8 hexadecimal digits)

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"ConfigurationIdentifier",
	"Data":{
	"Id":"1234ABCD"
	}
	}]
	}

29. GetDateTime request

Purpose	Gets date and time
Data	Absent
Permission required	Any permission among listed below: – *Configuration* – *Control* – *Monitoring* – *Reports* – *Payments* – *Shifts*
Response	DateTime response

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"GetDateTime"
	}]
	}

30. SetDateTime request

Purpose	Sets date and time
Data	"*DateTime*" – date and time in format *YYYY-MM-DDThh:mm:ss* (string, 19 symbols), where: – *YYYY* – year (range from 2000 to 2099) – *MM* – month (range from 01 to 12) – *DD* – day (range from 01 to 31) – *T* – separator between date and time fields – *hh* – hours (range from 00 to 23) – *mm* – minutes (range from 00 to 59) – *ss* – seconds (range from 00 to 59) "*AutoSynchronize*" – option to periodically synchronize time with time server (Boolean, possible values: "*true*", "*false*") "*UTCOffset*" – difference in minutes from Coordinated Universal Time (UTC) for a particular place and date (signed integer, range from -1440 to 1440, where 1440 = 24 hours x 60 minutes)
Permission required	Any permission among listed below: – *Configuration*
Response	Confirmation response message (in case of successful execution) or error response message (in case of any problem)

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"SetDateTime",
	"Data":{
	"DateTime":"2020-12-29T08:05:41",
	"AutoSynchronize":true,
	"UTCOffset":60
	}
	}]
	}

31. DateTime response

Purpose	Returns date and time
Data	"*DateTime*" – date and time in format *YYYY-MM-DDThh:mm:ss* (string, 19 symbols), where: – *YYYY* – year (range from 2000 to 2099) – *MM* – month (range from 01 to 12) – *DD* – day (range from 01 to 31) – *T* – separator between date and time fields – *hh* – hours (range from 00 to 23) – *mm* – minutes (range from 00 to 59) – *ss* – seconds (range from 00 to 59) "*AutoSynchronize*" – option to periodically synchronize time with time server (Boolean, possible values: "*true*", "*false*") "*UTCOffset*" – difference in minutes from Coordinated Universal Time (UTC) for a particular place and date (signed integer, range from -1440 to 1440, where 1440 = 24 hours x 60 minutes)

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"DateTime",
	"Data":{
	"DateTime":"2020-12-29T08:05:41",
	"AutoSynchronize":true,
	"UTCOffset":60
	}
	}]
	}

32. GetPtsNetworkSettings request

Purpose	Gets the PTS-2 controller's network settings: IP-address, network mask and gateway
Data	Absent
Permission required	Any permission among listed below: – *Configuration*
Response	PtsNetworkSettings response

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"GetPtsNetworkSettings"
	}]
	}

33. SetPtsNetworkSettings request

Purpose	Sets the PTS-2 controller's network settings: IP-address, network mask and gateway
Data	"*IpAddress*" – array of IPv4 octets (each is integer, range from 0 to 255) "*NetMask*" – array of network mask octets (each is integer, range from 0 to 255) "*Gateway*" – array of network gateway octets (each is integer, range from 0 to 255) "*HttpPort*" – network port for HTTP (integer, range from 0 to 65535) "*HttpsPort*" – network port for HTTPS (integer, range from 0 to 65535) "*Dns1*" – array of IPv4 octets (each is integer, range from 0 to 255) "*Dns2*" – array of IPv4 octets (each is integer, range from 0 to 255)
Permission required	Any permission among listed below: – *Configuration*
Response	Confirmation response message (in case of successful execution) or error response message (in case of any problem)

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"SetPtsNetworkSettings",
	"Data":{
	"IpAddress":[192,168,1,117],
	"NetMask":[255,255,255,0],
	"Gateway":[192,168,1,13],
	"HttpPort":80,
	"HttpsPort":443,
	"Dns1":[8,8,8,8],
	"Dns2":[8,8,4,4]
	}
	}]
	}

34. PtsNetworkSettings response

Purpose	Returns the PTS-2 controller's network settings: IP-address, network mask and gateway
Data	"*IpAddress*" – array of IPv4 octets (each is integer, range from 0 to 255) "*NetMask*" – array of network mask octets (each is integer, range from 0 to 255) "*Gateway*" – array of network gateway octets (each is integer, range from 0 to 255) "*HttpPort*" – network port for HTTP (integer, range from 0 to 65535) "*HttpsPort*" – network port for HTTPS (integer, range from 0 to 65535) "*Dns1*" – array of IPv4 octets (each is integer, range from 0 to 255) "*Dns2*" – array of IPv4 octets (each is integer, range from 0 to 255) "*UsedProtocolType*" – type of presently used protocol (string, possible values: "*HTTP*", "*HTTPS*") "*UsedAuthenticationType*" – type of presently used authentication (string, possible values: "*Basic*", "*Digest*")

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"PtsNetworkSettings",
	"Data":{
	"IpAddress":[192,168,1,117],
	"NetMask":[255,255,255,0],
	"Gateway":[192,168,1,13],
	"HttpPort":80,
	"HttpsPort":443,
	"Dns1":[8,8,8,8],
	"Dns2":[8,8,4,4],
	"UsedProtocolType":"HTTPS",
	"UsedAuthenticationType":"Basic",
	"MAC":"56:C6:68:C2:8B:D3"
	}
	}]
	}

35. GetRemoteServerConfiguration request

Purpose	Gets remote server configuration
Data	Absent
Permission required	Any permission among listed below: – *Configuration*
Response	RemoteServerConfiguration response

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"GetRemoteServerConfiguration"
	}]
	}

36. SetRemoteServerConfiguration request

Purpose	Sets remote server configuration
Data	"*IpAddress*" – server address, array of IPv4 octets (each is integer, range from 0 to 255) "*DomainName*" – server address (string, up to 30 ASCII characters) "*UserId*" – identifier of user, who authorized the transaction (integer, range from 1 to 30) "*ProtocolType*" – type of protocol used (string, possible values: "*HTTP*" and "*HTTPS*") "*ServerResponseTimeoutSeconds*" – timeout of expecting a response from server in seconds (integer, range from 1 to 65535) "*UseDeviceIdentifierAsLogin*" – option to use device unique identifier as login at connection to server to server (Boolean, possible values: "*true*", "*false*") "*UploadPumpTransactions*" – option to upload pump transactions to server (Boolean, possible values: "*true*", "*false*") "*UploadPumpTransactionsUri*" – server request URI for upload of pump transactions (string, up to 40 ASCII characters) "*UploadTankMeasurements*" – option to upload tank measurements to server (Boolean, possible values: "*true*", "*false*") "*UploadTankMeasurementsUri*" – server request URI for upload of tank measurements (string, up to 40 ASCII characters) "*UploadInTankDeliveries*" – option to upload in-tank deliveries to server (Boolean, possible values: "*true*", "*false*") "*UploadInTankDeliveriesUri*" – server request URI for upload of in-tank deliveries (string, up to 40 ASCII characters) "*UploadGpsRecords*" – option to upload GPS records to server (Boolean, possible values: "*true*", "*false*") "*UploadGpsRecordsUri*" – server request URI for upload of GPS records (string, up to 40 ASCII characters) "*UploadAlertRecords*" – option to upload alert records to server (Boolean, possible values: "*true*", "*false*") "*UploadAlertRecordsUri*" – server request URI for upload of alert records (string, up to 40 ASCII characters) "*UploadPayments*" – option to upload payments to server (Boolean, possible values: "*true*", "*false*") "*UploadPaymentsUri*" – server request URI for upload of payments (string, up to 40 ASCII characters) "*UploadShifts*" – option to upload working shifts to server (Boolean, possible values: "*true*", "*false*") "*UploadShiftsUri*" – server request URI for upload of working shifts (string, up to 40 ASCII characters) "*UploadConfiguration*" – option to send upload configuration requests to server (Boolean, possible values: "*true*", "*false*") "*UploadConfigurationUri*" – server request URI for upload of configuration (string, up to 40 ASCII characters) "*UploadStatus*" – option to send upload status requests to server (Boolean, possible values: "*true*", "*false*") "*UploadStatusUri*" – server request URI for upload of status (string, up to 40 ASCII characters)

	"*UploadStatusRequestsPeriodSeconds*" – period of sending upload status requests to server in seconds (integer, range from 1 to 65535) "*RequestTagsInformation*" – option to send requests to request tags information from server (Boolean, possible values: "*true*", "*false*") "*RequestTagsInformationUri*" – server request URI for requesting tags information (string, up to 40 ASCII characters) "*Port*" – server port for data upload (integer, range from 1 to 65535) "*SecretKey*" – secret key used for message signatures (string, up to 20 characters) "*UpdateSecretKey*" – option to update the previously entered value of secret key (Boolean, possible values: "*true*", "*false*") "W*ebsocketsUri*" – server request URI for Websocket protocol communication (string, up to 40 ASCII characters) "*WebsocketsPort*" – server port for Websocket protocol communication (integer, range from 1 to 65535) "*WebsocketsUploadPumpTransactions*" – option to send pump transactions to server using Websocket protocol (Boolean, possible values: "*true*", "*false*") "*WebsocketsUploadTankMeasurements*" – option to send tank measurements to server using Websocket protocol (Boolean, possible values: "*true*", "*false*") "*WebsocketsUploadInTankDeliveries*" – option to send in-tank deliveries to server using Websocket protocol (Boolean, possible values: "*true*", "*false*") "*WebsocketsUploadGpsRecords*" – option to send GPS records to server using Websocket protocol (Boolean, possible values: "*true*", "*false*") "*WebsocketsUploadAlertRecords*" – option to send alert records to server using Websocket protocol (Boolean, possible values: "*true*", "*false*") "*WebsocketsUploadPayments*" – option to send payments to server using Websocket protocol (Boolean, possible values: "*true*", "*false*") "*WebsocketsUploadShifts*" – option to send working shifts to server using Websocket protocol (Boolean, possible values: "*true*", "*false*") "*WebsocketsUploadConfiguration*" – option to send configuration to server using Websocket protocol (Boolean, possible values: "*true*", "*false*") "*WebsocketsUploadStatus*" – option to send status to server using Websocket protocol (Boolean, possible values: "*true*", "*false*") "*WebsocketsUploadStatusRequestsPeriodSeconds*" – period of sending status requests to server using Websocket protocol in seconds (integer, range from 1 to 65535) "*WebsocketsRequestTagsInformation*" – option to send requests to request tags information from server (Boolean, possible values: "*true*", "*false*") "*UseWebsocketsCommunication*" – option to enable full communication using Websocket protocol (Boolean, possible values: "*true*", "*false*")
Notes	1. Login and password are taken from user by the "*UserId*" field, which should be already configured in the PTS-2 controller using SetUsersConfiguration request. 2. If the field "*UseDeviceIdentifierAsLogin*" is set – then login is taken as device unique identifier (can be checked using GetUniqueIdentifier request).
Permission required	Any permission among listed below: – *Configuration*
Response	Confirmation response message (in case of successful execution) or error response message (in case of any problem)
Example	{

```
"Protocol":"jsonPTS",
"Packets":[{
  "Id":1,
  "Type":"SetRemoteServerConfiguration",
  "Data":{
    "IpAddress":[195,234,4,79],
    "DomainName":"technotrade.ua",
    "UserId":1,
    "ServerResponseTimeoutSeconds":1,
    "UseDeviceIdentifierAsLogin":true,
    "UploadPumpTransactions":true,
    "UploadPumpTransactionsUri":"ptsPumpTransactions",
    "UploadTankMeasurements":true,
    "UploadTankMeasurementsUri":"ptsTankMeasurements",
    "UploadInTankDeliveries":true,
    "UploadInTankDeliveriesUri":"ptsInTankDeliveries",
    "UploadGpsRecords":true,
    "UploadGpsRecordsUri":"ptsGpsRecords",
    "UploadAlertRecords":true,
    "UploadAlertRecordsUri":"ptsAlertRecords",
    "UploadPayments":true,
    "UploadPaymentsUri":"ptsPayments",
    "UploadShifts":true,
    "UploadShiftsUri":"ptsShifts",
    "UploadConfiguration":true,
    "UploadConfigurationUri":"ptsConfiguration",
    "UploadStatus":true,
    "UploadStatusUri":"ptsStatus",
    "UploadStatusRequestsPeriodSeconds":10,
    "RequestTagsInformation":true,
    "RequestTagsInformationUri":"ptsTags",
    "Port":80,
    "SecretKey":"qwerty12345",
    "UpdateSecretKey":false,
    "WebsocketsUri":"jsonPTSWebsockets",
    "WebsocketsPort":8080,
    "WebsocketsUploadPumpTransactions":false,
    "WebsocketsUploadTankMeasurements":false,
    "WebsocketsUploadInTankDeliveries":false,
    "WebsocketsUploadGpsRecords":false,
    "WebsocketsUploadAlertRecords":false,
    "WebsocketsUploadPayments":false,
    "WebsocketsUploadShifts":false,
```

```
        "WebsocketsUploadConfiguration":false,
        "WebsocketsUploadStatus":false,
        "WebsocketsUploadStatusRequestsPeriodSeconds":10,
        "WebsocketsRequestTagsInformation":false,
        "UseWebsocketsCommunication":true
      }
    }]
}
```

37. RemoteServerConfiguration response

Purpose	Returns remote server configuration
Data	"*IpAddress*" – server address, array of IPv4 octets (each is integer, range from 0 to 255) "*DomainName*" – server address (string, up to 30 ASCII characters) "*UserId*" – identifier of user, who authorized the transaction (integer, range from 1 to 30) "*ProtocolType*" – type of protocol used (string, possible values: "*HTTP*" and "*HTTPS*") "*ServerResponseTimeoutSeconds*" – timeout of expecting a response from server in seconds (integer, range from 1 to 65535) "*UseDeviceIdentifierAsLogin*" – option to use device unique identifier as login at connection to server to server (Boolean, possible values: "*true*", "*false*") "*UploadPumpTransactions*" – option to upload pump transactions to server (Boolean, possible values: "*true*", "*false*") "*UploadPumpTransactionsUri*" – server request URI for upload of pump transactions (string, up to 40 ASCII characters) "*UploadTankMeasurements*" – option to upload tank measurements to server (Boolean, possible values: "*true*", "*false*") "*UploadTankMeasurementsUri*" – server request URI for upload of tank measurements (string, up to 40 ASCII characters) "*UploadInTankDeliveries*" – option to upload in-tank deliveries to server (Boolean, possible values: "*true*", "*false*") "*UploadInTankDeliveriesUri*" – server request URI for upload of in-tank deliveries (string, up to 40 ASCII characters) "*UploadGpsRecords*" – option to upload GPS records to server (Boolean, possible values: "*true*", "*false*") "*UploadGpsRecordsUri*" – server request URI for upload of GPS records (string, up to 40 ASCII characters) "*UploadAlertRecords*" – option to upload alert records to server (Boolean, possible values: "*true*", "*false*") "*UploadAlertRecordsUri*" – server request URI for upload of alert records (string, up to 40 ASCII characters) "*UploadPayments*" – option to upload payments to server (Boolean, possible values: "*true*", "*false*") "*UploadPaymentsUri*" – server request URI for upload of payments (string, up to 40 ASCII characters) "*UploadShifts*" – option to upload working shifts to server (Boolean, possible values: "*true*", "*false*") "*UploadShiftsUri*" – server request URI for upload of working shifts (string, up to 40 ASCII characters) "*UploadConfiguration*" – option to send upload configuration requests to server (Boolean, possible values: "*true*", "*false*") "*UploadConfigurationUri*" – server request URI for upload of configuration (string, up to 40 ASCII characters) "*UploadStatus*" – option to send upload status requests to server (Boolean, possible values: "*true*", "*false*") "*UploadStatusUri*" – server request URI for upload of status (string, up to 40 ASCII characters)

	"*UploadStatusRequestsPeriodSeconds*" – period of sending upload status requests to server in seconds (integer, range from 1 to 65535) "*RequestTagsInformation*" – option to send requests to request tags information from server (Boolean, possible values: "*true*", "*false*") "*RequestTagsInformationUri*" – server request URI for requesting tags information (string, up to 40 ASCII characters) "*Port*" – server port for data upload (integer, range from 1 to 65535) "*IsUploadSuccessful*" – result of data upload (Boolean, possible values: "*true*", "*false*") "W*ebsocketsUri*" – server request URI for Websocket protocol communication (string, up to 40 ASCII characters) "*WebsocketsPort*" – server port for Websocket protocol communication (integer, range from 1 to 65535) "*WebsocketsUploadPumpTransactions*" – option to send pump transactions to server using Websocket protocol (Boolean, possible values: "*true*", "*false*") "*WebsocketsUploadTankMeasurements*" – option to send tank measurements to server using Websocket protocol (Boolean, possible values: "*true*", "*false*") "*WebsocketsUploadInTankDeliveries*" – option to send in-tank deliveries to server using Websocket protocol (Boolean, possible values: "*true*", "*false*") "*WebsocketsUploadGpsRecords*" – option to send GPS records to server using Websocket protocol (Boolean, possible values: "*true*", "*false*") "*WebsocketsUploadAlertRecords*" – option to send alert records to server using Websocket protocol (Boolean, possible values: "*true*", "*false*") "*WebsocketsUploadPayments*" – option to send payments to server using Websocket protocol (Boolean, possible values: "*true*", "*false*") "*WebsocketsUploadShifts*" – option to send working shifts to server using Websocket protocol (Boolean, possible values: "*true*", "*false*") "*WebsocketsUploadConfiguration*" – option to send configuration to server using Websocket protocol (Boolean, possible values: "*true*", "*false*") "*WebsocketsUploadStatus*" – option to send status to server using Websocket protocol (Boolean, possible values: "*true*", "*false*") "*WebsocketsUploadStatusRequestsPeriodSeconds*" – period of sending status requests to server using Websocket protocol in seconds (integer, range from 1 to 65535) "*WebsocketsRequestTagsInformation*" – option to send requests to request tags information from server (Boolean, possible values: "*true*", "*false*") "*IsWebsocketsCommunicationSuccessful*" – result of Websocket protocol communication (Boolean, possible values: "*true*", "*false*") "*UseWebsocketsCommunication*" – option to enable full communication using Websocket protocol (Boolean, possible values: "*true*", "*false*") "*IsConnectionSuccessful*" – result of connection to a remote server (Boolean, possible values: "*true*", "*false*")

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"RemoteServerConfiguration",
	"Data":{

```
"IpAddress":[195,234,4,79],
"DomainName":"technotrade.ua",
"UserId":1,
"ServerResponseTimeoutSeconds":1,
"UseDeviceIdentifierAsLogin":true,
"UploadPumpTransactions":true,
"UploadPumpTransactionsUri":"ptsPumpTransactions",
"UploadTankMeasurements":true,
"UploadTankMeasurementsUri":"ptsTankMeasurements",
"UploadInTankDeliveries":true,
"UploadInTankDeliveriesUri":"ptsInTankDeliveries",
"UploadGpsRecords":true,
"UploadGpsRecordsUri":"ptsGpsRecords",
"UploadAlertRecords":true,
"UploadAlertRecordsUri":"ptsAlertRecords",
"UploadPayments":true,
"UploadPaymentsUri":"ptsPayments",
"UploadShifts":true,
"UploadShiftsUri":"ptsShifts",
"UploadConfiguration":true,
"UploadConfigurationUri":"ptsConfiguration",
"UploadStatus":true,
"UploadStatusUri":"ptsStatus",
"UploadStatusRequestsPeriodSeconds":10,
"RequestTagsInformation":true,
"RequestTagsInformationUri":"ptsTags",
"Port":80,
"IsUploadSuccessful":true,
"WebsocketsUri":"jsonPTSWebsockets",
"WebsocketsPort":8080,
"WebsocketsUploadPumpTransactions":false,
"WebsocketsUploadTankMeasurements":false,
"WebsocketsUploadInTankDeliveries":false,
"WebsocketsUploadGpsRecords":false,
"WebsocketsUploadAlertRecords":false,
"WebsocketsUploadPayments":false,
"WebsocketsUploadShifts":false,
"WebsocketsUploadConfiguration":false,
"WebsocketsUploadStatus":false,
"WebsocketsUploadStatusRequestsPeriodSeconds":10,
"WebsocketsRequestTagsInformation":false,
"IsWebsocketsCommunicationSuccessful":true,
"UseWebsocketsCommunication":true,
```

```
        "IsConnectionSuccessful":true
      }
   }]
}
```

38. GetDailyProcessingTime request

Purpose	Gets daily processing time
Data	Absent
Permission required	Any permission among listed below: – *Configuration* – *Control* – *Monitoring* – *Reports* – *Payments* – *Shifts*
Response	DailyProcessingTime response

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"GetDailyProcessingTime"
	}]
	}

39. SetDailyProcessingTime request

Purpose	Sets daily processing time
Data	"*Time*" – time in format *hh:mm* (string, 5 symbols), where: – *hh* – hours (range from 00 to 23) – *mm* – minutes (range from 00 to 59) "*ProcessFiles*" – option to process files stored on SD flash disk (Boolean, possible values: "*true*", "*false*") "*BackupConfiguration*" – option to make regular backup of configuration during processing (Boolean, possible values: "*true*", "*false*") "*RegenerateTankAutoCalibrationCharts*" – option to regenerate automatic tank calibration charts from tank interval volume tables (Boolean, possible values: "*true*", "*false*")
Permission required	Any permission among listed below: – *Configuration*
Response	Confirmation response message (in case of successful execution) or error response message (in case of any problem)

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"SetDailyProcessingTime",
	"Data":{
	"Time":"02:00",
	"ProcessFiles":true,
	"BackupConfiguration":true,
	"RegenerateTankAutoCalibrationCharts":true
	}
	}]
	}

40. DailyProcessingTime response

Purpose	Returns daily processing time
Data	"*Time*" – time in format *hh:mm* (string, 5 symbols), where: – *hh* – hours (range from 00 to 23) – *mm* – minutes (range from 00 to 59) "*ProcessFiles*" – option to process files stored on SD flash disk (Boolean, possible values: "*true*", "*false*") "*BackupConfiguration*" – option to make regular backup of configuration during processing (Boolean, possible values: "*true*", "*false*") "*RegenerateTankAutoCalibrationCharts*" – option to regenerate automatic tank calibration charts from tank interval volume tables (Boolean, possible values: "*true*", "*false*")

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"DailyProcessingTime",
	"Data":{
	"Time":"02:00",
	"ProcessFiles":true,
	"BackupConfiguration":true,
	"RegenerateTankAutoCalibrationCharts":true
	}
	}]
	}

41. MakeDailyProcessing request

Purpose	Immediately makes daily processing
Data	"*ProcessFiles*" – option to process files stored on SD flash disk (Boolean, possible values: "*true*", "*false*") "*BackupConfiguration*" – option to make regular backup of configuration during processing (Boolean, possible values: "*true*", "*false*") "*RegenerateTankAutoCalibrationCharts*" – option to regenerate automatic tank calibration charts from tank interval volume tables (Boolean, possible values: "*true*", "*false*")
Permission required	Any permission among listed below: – *Configuration*
Note	The process can take some time depending on options selected, while performing the controller might be busy and not accessible
Response	Confirmation response message (in case of successful execution) or error response message (in case of any problem)

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"MakeDailyProcessing",
	"Data":{
	"ProcessFiles":true,
	"BackupConfiguration":true,
	"RegenerateTankAutoCalibrationCharts":true
	}
	}]
	}

42. GetSystemDecimalDigits request

Purpose	Gets system decimal digits settings
Data	Absent
Permission required	Any permission among listed below: – *Configuration* – *Control* – *Monitoring* – *Reports* – *Payments* – *Shifts*
Response	SystemDecimalDigits response

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"GetSystemDecimalDigits"
	}]
	}

43. SetSystemDecimalDigits request

Purpose	Sets system decimal digits settings
Data	"*Price*" – number of decimal digits in price value (integer, range from 0 to 3) "*Amount*" – number of decimal digits in amount value (integer, range from 0 to 3) "*Volume*" – number of decimal digits in volume value (integer, range from 0 to 3) "*AmountTotal*" – number of decimal digits in amount total counter value (integer, range from 0 to 3) "*VolumeTotal*" – number of decimal digits in volume total counter value (integer, range from 0 to 3)
Permission required	Any permission among listed below: – *Configuration*
Response	Confirmation response message (in case of successful execution) or error response message (in case of any problem)

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"SetSystemDecimalDigits",
	"Data":{
	"Price":2,
	"Amount":2,
	"Volume":2,
	"AmountTotal":2,
	"VolumeTotal":2
	}
	}]
	}

44. SystemDecimalDigits response

Purpose	Returns system decimal digits settings
Data	"*Price*" – number of decimal digits in price value (integer, range from 0 to 3) "*Amount*" – number of decimal digits in amount value (integer, range from 0 to 3) "*Volume*" – number of decimal digits in volume value (integer, range from 0 to 3) "*AmountTotal*" – number of decimal digits in amount total counter value (integer, range from 0 to 3) "*VolumeTotal*" – number of decimal digits in volume total counter value (integer, range from 0 to 3)

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"SystemDecimalDigits",
	"Data":{
	"Price":2,
	"Amount":2,
	"Volume":2,
	"AmountTotal":2,
	"VolumeTotal":2
	}
	}]
	}

45. GetParameter request

Purpose	Gets parameter value
Data	"*Device*" – identifier of parameter object (string, possible values: "*PTS*", "*Pump*", "*Probe*") "*Number*" – device number (integer, required when value of "*Device*" is "*Pump*" or "*Probe*", range from 1 to 99) "*Address*" – parameter address (integer, range from 1 to 9999)
Permission required	Any permission among listed below: – *Configuration*
Response	Parameter response

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"GetParameter",
	"Data":{
	"Device":"Pump",
	"Number":1,
	"Address":1
	}
	}]
	}

46. SetParameter request

Purpose	Sets parameter value
Data	"*Device*" – identifier of parameter object (string, possible values: "*PTS*", "*Pump*", "*Probe*") "*Number*" – device number (integer, required when value of "*Device*" is "*Pump*" or "*Probe*", range from 1 to 99) "*Address*" – parameter address (integer, range from 1 to 9999) "*Value*" – parameter value (string, up to 8 hexadecimal digits)
Permission required	Any permission among listed below: – *Configuration*
Response	Confirmation response message (in case of successful execution) or error response message (in case of any problem)

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"SetParameter",
	"Data":{
	"Device":"Pump",
	"Number":1,
	"Address":1,
	"Value":"00000123"
	}
	}]
	}

47. Parameter response

Purpose	Returns parameter value
Data	"*Device*" – identifier of parameter object (string, possible values: "*PTS*", "*Pump*", "*Probe*") "*Number*" – device number (integer, required when value of "*Device*" is "*Pump*" or "*Probe*", range from 1 to 99) "*Address*" – parameter address (integer, range from 1 to 9999) "*Value*" – parameter value (string, up to 8 hexadecimal digits)

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"Parameter",
	"Data":{
	"Device":"Pump",
	"Number":1,
	"Address":1,
	"Value":"00000123"
	}
	}]
	}

48. GetPumpsConfiguration request

Purpose	Gets pump ports and pumps configuration
Data	Absent
Permission required	Any permission among listed below: – *Configuration* – *Control* – *Monitoring* – *Reports* – *Payments* – *Shifts*
Response	PumpsConfiguration response

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"GetPumpsConfiguration"
	}]
	}

49. SetPumpsConfiguration request

Purpose	Sets pump ports and pumps configuration
Data	"*Ports*" – array of objects for configuration of pumps' ports, elements of each object: – "*Id*" – identifier of pump port (integer, range from 1 to 4) – "*Protocol*" – value of pump port communication protocol (integer, range from 1 to 99) – "*BaudRate*" – value of pump port baud rate (integer, range from 1 to 99) "*Pumps*" – array of objects for configuration of pumps, elements of each object: – "*Id*" – identifier of pump (integer, range from 1 to 120) – "*Port*" – identifier of pump port (integer, range from 1 to 4) – "*Address*" – value of pump communication address (integer, range from 1 to 99)
Note	"*Ports*" objects and "*Pumps*" objects, which are absent in the sent arrays, are considered to be not configured (filled with zeroes)
Permission required	Any permission among listed below: – *Configuration*
Response	Confirmation response message (in case of successful execution) or error response message (in case of any problem)

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"SetPumpsConfiguration",
	"Data":{
	"Ports":[{
	"Id":1,
	"Protocol":2,
	"BaudRate":4
	},{
	"Id":2,
	"Protocol":37,
	"BaudRate":5
	}],
	"Pumps":[{
	"Id":1,
	"Port":1,
	"Address":1
	},{
	"Id":2,
	"Port":2,
	"Address":2
	}]
	}
	}]

	}

50. PumpsConfiguration response

Purpose	Returns pump ports and pumps configuration
Data	"*Ports*" – array of objects for configuration of pumps' ports, elements of each object: – "*Id*" – identifier of pump port (integer, range from 1 to 4) – "*Protocol*" – value of pump port communication protocol (integer, range from 1 to 99) – "*BaudRate*" – value of pump port baud rate (integer, range from 1 to 99) "*Pumps*" – array of objects for configuration of pumps, elements of each object: – "*Id*" – identifier of pump (integer, range from 1 to 120) – "*Port*" – identifier of pump port (integer, range from 1 to 4) – "*Address*" – value of pump communication address (integer, range from 1 to 99)
Note	"*Ports*" objects and "*Pumps*" objects, which are absent in the received arrays, are considered to be not configured (filled with zeroes)

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"PumpsConfiguration",
	"Data":{
	"Ports":[{
	"Id":1,
	"Protocol":2,
	"BaudRate":4
	},{
	"Id":2,
	"Protocol":37,
	"BaudRate":5
	}],
	"Pumps":[{
	"Id":1,
	"Port":1,
	"Address":1
	},{
	"Id":2,
	"Port":2,
	"Address":2
	}]
	}
	}]
	}

51. GetProbesConfiguration request

Purpose	Gets probe ports and probes configuration
Data	Absent
Permission required	Any permission among listed below: – *Configuration* – *Control* – *Monitoring* – *Reports* – *Payments* – *Shifts*
Response	ProbesConfiguration response

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"GetProbesConfiguration"
	}]
	}

52. SetProbesConfiguration request

Purpose	Sets probe ports and probes configuration
Data	"*Ports*" – array of objects for configuration of probes' ports, elements of each object: – "*Id*" – identifier of probe port (string, possible values: "*DISP*", "*LOG*", "*USER*", "*PC*") – "*Protocol*" – value of probe port communication protocol (integer, range from 1 to 99) – "*BaudRate*" – value of probe port baud rate (integer, range from 1 to 99) "*Probes*" – array of objects for configuration of probes, elements of each object: – "*Id*" – identifier of probe (integer, range from 1 to 20) – "*Port*" – identifier of probe port (string, possible values: "*DISP*", "*LOG*", "*USER*", "*PC*") – "*Address*" – value of probe communication address (integer, range from 1 to 999999)
Note	"*Ports*" objects and "*Probes*" objects, which are absent in the sent arrays, are considered to be not configured (filled with zeroes)
Permission required	Any permission among listed below: – *Configuration*
Response	Confirmation response message (in case of successful execution) or error response message (in case of any problem)

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"SetProbesConfiguration",
	"Data":{
	"Ports":[{
	"Id":"USER",
	"Protocol":1,
	"BaudRate":4
	},{
	"Id":"DISP",
	"Protocol":2,
	"BaudRate":4
	}],
	"Probes":[{
	"Id":1,
	"Port":"USER",
	"Address":1
	},{
	"Id":2,
	"Port":"DISP",
	"Address":1979

```
            } ]
        }
    } ]
}
```

53. ProbesConfiguration response

Purpose	Returns probe ports and probes configuration
Data	"*Ports*" – array of objects for configuration of probes' ports, elements of each object: – "*Id*" – identifier of probe port (string, possible values: "*DISP*", "*LOG*", "*USER*", "*PC*") – "*Protocol*" – value of probe port communication protocol (integer, range from 1 to 99) – "*BaudRate*" – value of probe port baud rate (integer, range from 1 to 99) "*Probes*" – array of objects for configuration of probes, elements of each object: – "*Id*" – identifier of probe (integer, range from 1 to 20) – "*Port*" – identifier of probe port (string, possible values: "*DISP*", "*LOG*", "*USER*", "*PC*") – "*Address*" – value of probe communication address (integer, range from 1 to 999999)
Note	"*Ports*" objects and "*Probes*" objects, which are absent in the sent arrays, are considered to be not configured (filled with zeroes)

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"ProbesConfiguration",
	"Data":{
	"Ports":[{
	"Id":"USER",
	"Protocol":1,
	"BaudRate":4
	},{
	"Id":"DISP",
	"Protocol":2,
	"BaudRate":4
	}],
	"Probes":[{
	"Id":1,
	"Port":"USER",
	"Address":1
	},{
	"Id":2,
	"Port":"DISP",
	"Address":1979
	}]
	}
	}]
	}

54. GetFuelGradesPrices request

Purpose	Gets fuel grades prices
Data	Absent
Permission required	Any permission among listed below: – *Configuration* – *Control* – *Monitoring* – *Reports* – *Payments* – *Shifts*
Response	FuelGradesPrices response

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"GetFuelGradesPrices"
	}]
	}

55. SetFuelGradesPrices request

Purpose	Sets fuel grades prices
Data	"*FuelGradesPrices*" – array of objects for prices of fuel grades, elements of each object: – "*FuelGradeId*" – identifier of fuel grade (integer, range from 1 to 10) – "*Price*" – price of fuel grade (float, up to 9 digits with dot as a decimal separator, value without a decimal separator is considered to have no decimal digits)
Note	"*FuelGradesPrices*" objects, which are absent in the sent arrays, are considered to be not configured (filled with zeroes).
Permission required	Any permission among listed below: – *Configuration* – *Control*
Response	Confirmation response message (in case of successful execution) or error response message (in case of any problem)

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"SetFuelGradesPrices",
	"Data":{
	"FuelGradesPrices":[{
	"FuelGradeId":1,
	"Price":27.50
	},{
	"FuelGradeId":2,
	"Price":23.99
	}]
	}
	}]
	}

56. FuelGradesPrices response

Purpose	Returns fuel grades prices
Data	"*FuelGradesPrices*" – array of objects for prices of fuel grades, elements of each object: – "*FuelGradeId*" – identifier of fuel grade (integer, range from 1 to 10) – "*Price*" – price of fuel grade (float, up to 9 digits with dot as a decimal separator, value without a decimal separator is considered to have no decimal digits)
Note	"*FuelGradesPrices*" objects, which are absent in the sent arrays, are considered to be not configured (filled with zeroes).

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"FuelGradesPrices",
	"Data":{
	"FuelGradesPrices":[{
	"FuelGradeId":1,
	"Price":27.50
	},{
	"FuelGradeId":2,
	"Price":23.99
	}]
	}
	}]
	}

57. GetFuelGradesConfiguration request

Purpose	Gets fuel grades configuration: fuel products codes, names and prices
Data	Absent
Permission required	Any permission among listed below: – *Configuration* – *Control* – *Monitoring* – *Reports* – *Payments* – *Shifts*
Response	FuelGradesConfiguration response

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"GetFuelGradesConfiguration"
	}]
	}

58. SetFuelGradesConfiguration request

Purpose	Sets fuel grades configuration: fuel products codes, names and prices
Data	"*FuelGrades*" – array of objects for configuration of fuel grades, elements of each object: – "*Id*" – identifier of fuel grade (integer, range from 1 to 10) – "*Name*" – name of fuel grade (string, up to 20 ASCII characters) – "*Price*" – price of fuel grade (float, up to 9 digits with dot as a decimal separator, value without a decimal separator is considered to have no decimal digits) – "*ExpansionCoefficient*" – thermal coefficient of expansion at 15 °C for fuel grade (float, up to 5 digits after decimal point) – "*BlendTank1Id*" – optional parameter needed only for blended fuel grade, means identifier of first tank for blended fuel grade (integer, value range from 0 to 20, value 0 means no tank configured) – "*BlendTank1Percentage*" – optional parameter needed only for blended fuel grade, means blend percentage from first tank for blended fuel grade (integer, value range from 1 to 99) – "*BlendTank2Id*" – optional parameter needed only for blended fuel grade, means identifier of second tank for blended fuel grade (integer, value range from 0 to 20, value 0 means no tank configured)
Notes	1. "*FuelGrades*" objects, which are absent in the sent arrays, are considered to be not configured (filled with zeroes). 2. Field "*ExpansionCoefficient*" of fuel grade is needed in order to be able to calculate temperature-compensated volume for fuel grade in tank based on present volume and temperature. 3. If the field "*ExpansionCoefficient*" is empty – then its values is assumed to be 0. 4. Fields "*BlendTank1Id*", "*BlendTank2Id*", "*BlendTank1Percentage*" are optional and are needed only when the tank is blended. Blend percentage for tank 1 equals to value of "*BlendTank1Percentage*" field, blend percentage for tank 2 equals (100 – value of "*BlendTank1Percentage*" field).
Permission required	Any permission among listed below: – *Configuration*
Response	Confirmation response message (in case of successful execution) or error response message (in case of any problem)

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"SetFuelGradesConfiguration",
	"Data":{
	"FuelGrades":[{
	"Id":1,
	"Name":"Petrol",
	"Price":27.50,
	"ExpansionCoefficient":0.00110
	},{

```
            "Id":2,
            "Name":"Diesel",
            "Price":23.99,
            "ExpansionCoefficient":0.00082
          }]
        }
    }]
}
```

59. FuelGradesConfiguration response

Purpose	Returns fuel grades configuration: fuel products codes, names and prices
Data	"*FuelGrades*" – array of objects for configuration of fuel grades, elements of each object: – "*Id*" – identifier of fuel grade (integer, range from 1 to 10) – "*Name*" – name of fuel grade (string, up to 20 ASCII characters) – "*Price*" – price of fuel grade (float, up to 9 digits with dot as a decimal separator, value without a decimal separator is considered to have no decimal digits) – "*ExpansionCoefficient*" – thermal coefficient of expansion at 15 °C for fuel grade (float, up to 5 digits after decimal point) – "*BlendTank1Id*" – optional parameter needed only for blended fuel grade, means identifier of first tank for blended fuel grade (integer, value range from 0 to 20, value 0 means no tank configured) – "*BlendTank1Percentage*" – optional parameter needed only for blended fuel grade, means blend percentage from first tank for blended fuel grade (integer, value range from 1 to 99) – "*BlendTank2Id*" – optional parameter needed only for blended fuel grade, means identifier of second tank for blended fuel grade (integer, value range from 0 to 20, value 0 means no tank configured)
Notes	1. "*FuelGrades*" objects, which are absent in the sent arrays, are considered to be not configured (filled with zeroes). 2. Field "*ExpansionCoefficient*" of fuel grade is needed in order to be able to calculate temperature-compensated volume for fuel grade in tank based on present volume and temperature. 3. If the field "*ExpansionCoefficient*" is empty – then its values is assumed to be 0. 4. Fields "*BlendTank1Id*", "*BlendTank2Id*", "*BlendTank1Percentage*" are optional and are needed only when the tank is blended. Blend percentage for tank 1 equals to value of "*BlendTank1Percentage*" field, blend percentage for tank 2 equals (100 – value of "*BlendTank1Percentage*" field).

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"FuelGradesConfiguration",
	"Data":{
	"FuelGrades":[{
	"Id":1,
	"Name":"Petrol",
	"Price":27.50,
	"ExpansionCoefficient":0.00110
	},{
	"Id":2,
	"Name":"Diesel",
	"Price":23.99,
	"ExpansionCoefficient":0.00082
	}]

```
        }
    } ]
}
```

60. GetPricesSchedulerConfiguration request

Purpose	Gets fuel grades' prices scheduler configuration
Data	Absent
Permission required	Any permission among listed below: – *Configuration*
Response	FuelGradesConfiguration response

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"GetPricesSchedulerConfiguration"
	}]
	}

61. SetPricesSchedulerConfiguration request

Purpose	Sets fuel grades' prices scheduler configuration
Data	"*PriceSchedules*" – array of objects for configuration of scheduler, elements of each object: – "*Id*" – identifier of price schedule (integer, range from 1 to 10) – "*Enabled*" – flag stating that the price schedule is enabled (Boolean, possible values: "*true*", "*false*") – "*FuelGradeId*" – identifier of fuel grade (integer, range from 1 to 20) – "*Price*" – price value (float, up to 9 digits with dot as a decimal separator, value without a decimal separator is considered to have no decimal digits) – "*DateTime*" – date and time in format *YYYY-MM-DDThh:mm:ss* (string, 19 symbols), where: • *YYYY* – year (range from 2000 to 2099) • *MM* – month (range from 01 to 12) • *DD* – day (range from 01 to 31) • *T* – separator between date and time fields • *hh* – hours (range from 00 to 23) • *mm* – minutes (range from 00 to 59) • *ss* – seconds (range from 00 to 59) – "*EveryMonday*" – flag stating that this price schedule should be applied every Monday (Boolean, possible values: "*true*", "*false*") – "*EveryTuesday*" – flag stating that this price schedule should be applied every Tuesday (Boolean, possible values: "*true*", "*false*") – "*EveryWednesday*" – flag stating that this price schedule should be applied every Wednesday (Boolean, possible values: "*true*", "*false*") – "*EveryThursday*" – flag stating that this price schedule should be applied every Thursday (Boolean, possible values: "*true*", "*false*") – "*EveryFriday*" – flag stating that this price schedule should be applied every Friday (Boolean, possible values: "*true*", "*false*") – "*EverySaturday*" – flag stating that this price schedule should be applied every Saturday (Boolean, possible values: "*true*", "*false*") – "*EverySunday*" – flag stating that this price schedule should be applied every Sunday (Boolean, possible values: "*true*", "*false*")
Notes	1. "*PriceSchedules*" objects, which are absent in the sent arrays, are considered to be not configured (filled with zeroes). 2. Fields "*EveryMonday*", "*EveryTuesday*", "*EveryWednesday*", "*EveryThursday*", "*EveryFriday*", "*EverySaturday*", "*EverySunday*" are optional. If any of them is set – then the scheduler will be regularly applying the stated price for the stated fuel grade on the corresponding day of week at the time stated in field "*DateTime*". If none of these fields are stated – then the scheduler will apply the stated price for the stated fuel grade only once on the date and time stated in field "*DateTime*".
Permission required	Any permission among listed below: – *Configuration*
Response	Confirmation response message (in case of successful execution) or error response message (in case of any problem)
Example	{

```
  "Protocol":"jsonPTS",
  "Packets":[{
    "Id":1,
    "Type":"SetPricesSchedulerConfiguration",
    "Data":{
      "PriceSchedules":[{
        "Id":1,
        "Enabled":true,
        "FuelGradeId":1,
        "Price":27.50,
        "DateTime":"2024-12-19T12:00:00",
        "EveryMonday": false,
        "EveryTuesday": false,
        "EveryWednesday": false,
        "EveryThursday": false,
        "EveryFriday": false,
        "EverySaturday": false,
        "EverySunday": false
      },{
        "Id":2,
        "Enabled":false,
        "FuelGradeId":2,
        "Price":29.75,
        "DateTime":"2024-12-19T15:00:00",
        "EveryMonday": true,
        "EveryTuesday": true,
        "EveryWednesday": true,
        "EveryThursday": true,
        "EveryFriday": true,
        "EverySaturday": false,
        "EverySunday": false
      }]
    }
  }]
}
```

62. PricesSchedulerConfiguration response

Purpose	Returns fuel grades' prices scheduler configuration
Data	"*PriceSchedules*" – array of objects for configuration of scheduler, elements of each object: – "*Id*" – identifier of price schedule (integer, range from 1 to 10) – "*Enabled*" – flag stating that the price schedule is enabled (Boolean, possible values: "*true*", "*false*") – "*FuelGradeId*" – identifier of fuel grade (integer, range from 1 to 20) – "*Price*" – price value (float, up to 9 digits with dot as a decimal separator, value without a decimal separator is considered to have no decimal digits) – "*DateTime*" – date and time in format *YYYY-MM-DDThh:mm:ss* (string, 19 symbols), where: • *YYYY* – year (range from 2000 to 2099) • *MM* – month (range from 01 to 12) • *DD* – day (range from 01 to 31) • *T* – separator between date and time fields • *hh* – hours (range from 00 to 23) • *mm* – minutes (range from 00 to 59) • *ss* – seconds (range from 00 to 59) – "*EveryMonday*" – flag stating that this price schedule should be applied every Monday (Boolean, possible values: "*true*", "*false*") – "*EveryTuesday*" – flag stating that this price schedule should be applied every Tuesday (Boolean, possible values: "*true*", "*false*") – "*EveryWednesday*" – flag stating that this price schedule should be applied every Wednesday (Boolean, possible values: "*true*", "*false*") – "*EveryThursday*" – flag stating that this price schedule should be applied every Thursday (Boolean, possible values: "*true*", "*false*") – "*EveryFriday*" – flag stating that this price schedule should be applied every Friday (Boolean, possible values: "*true*", "*false*") – "*EverySaturday*" – flag stating that this price schedule should be applied every Saturday (Boolean, possible values: "*true*", "*false*") – "*EverySunday*" – flag stating that this price schedule should be applied every Sunday (Boolean, possible values: "*true*", "*false*")
Notes	1. "*PriceSchedules*" objects, which are absent in the sent arrays, are considered to be not configured (filled with zeroes). 2. Fields "*EveryMonday*", "*EveryTuesday*", "*EveryWednesday*", "*EveryThursday*", "*EveryFriday*", "*EverySaturday*", "*EverySunday*" are optional. If any of them is set – then the scheduler will be regularly applying the stated price for the stated fuel grade on the corresponding day of week at the time stated in field "*DateTime*". If none of these fields are stated – then the scheduler will apply the stated price for the stated fuel grade only once on the date and time stated in field "*DateTime*".

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"PricesSchedulerConfiguration",

```
    "Data":{
      "PriceSchedules":[{
        "Id":1,
        "Enabled":true,
        "FuelGradeId":1,
        "Price":27.50,
        "DateTime":"2024-12-19T12:00:00",
        "EveryMonday": false,
        "EveryTuesday": false,
        "EveryWednesday": false,
        "EveryThursday": false,
        "EveryFriday": false,
        "EverySaturday": false,
        "EverySunday": false
      },{
        "Id":2,
        "Enabled":false,
        "FuelGradeId":2,
        "Price":29.75,
        "DateTime":"2024-12-19T15:00:00",
        "EveryMonday": true,
        "EveryTuesday": true,
        "EveryWednesday": true,
        "EveryThursday": true,
        "EveryFriday": true,
        "EverySaturday": false,
        "EverySunday": false
      }]
    }
  }]
}
```

63. GetPaymentFormsConfiguration request

Purpose	Gets payment forms configuration: payment forms codes and names
Data	Absent
Permission required	Any permission among listed below: – *Configuration* – *Control* – *Monitoring* – *Reports* – *Payments* – *Shifts*
Response	PaymentFormsConfiguration response

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"GetPaymentFormsConfiguration"
	}]
	}

64. SetPaymentFormsConfiguration request

Purpose	Sets payment forms configuration: payment forms codes and names
Data	"*PaymentForms*" – array of objects for configuration of payment forms, elements of each object: – "*Id*" – identifier of fuel grade (integer, range from 1 to 10) – "*Name*" – name of fuel grade (string, up to 10 ASCII characters)
Notes	"*PaymentForms*" objects, which are absent in the sent arrays, are considered to be not configured (filled with zeroes).
Permission required	Any permission among listed below: – *Configuration*
Response	Confirmation response message (in case of successful execution) or error response message (in case of any problem)

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"SetPaymentFormsConfiguration",
	"Data":{
	"PaymentForms":[{
	"Id":1,
	"Name":"Cash"
	},{
	"Id":2,
	"Name":"BankCard"
	},{
	"Id":3,
	"Name":"LoyaltyCard"
	}]
	}
	}]
	}

65. PaymentFormsConfiguration response

Purpose	Returns payment forms configuration: payment forms codes and names
Data	"*PaymentForms*" – array of objects for configuration of payment forms, elements of each object: – "*Id*" – identifier of fuel grade (integer, range from 1 to 10) – "*Name*" – name of fuel grade (string, up to 10 ASCII characters)
Notes	"*PaymentForms*" objects, which are absent in the sent arrays, are considered to be not configured (filled with zeroes).

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"PaymentFormsConfiguration",
	"Data":{
	"PaymentForms":[{
	"Id":1,
	"Name":"Cash"
	},{
	"Id":2,
	"Name":"BankCard"
	},{
	"Id":3,
	"Name":"LoyaltyCard"
	}]
	}
	}]
	}

66. GetPumpNozzlesConfiguration request

Purpose	Gets pump nozzles configuration: assignment of pump nozzles to fuel grades, tanks and payment forms
Data	Absent
Permission required	Any permission among listed below: – *Configuration* – *Control* – *Monitoring* – *Reports* – *Payments* – *Shifts*
Response	PumpNozzlesConfiguration response

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"GetPumpNozzlesConfiguration"
	}]
	}

67. SetPumpNozzlesConfiguration request

Purpose	Sets pump nozzles configuration: assignment of pump nozzles to fuel grades, tanks and payment forms
Data	"*PumpNozzles*" – array of objects for configuration of pump nozzles: – "*PumpId*" – identifier of pump (integer, range from 1 to 120) – "*FuelGradeIds*" – array of fuel grade IDs for nozzles (up to 6 integer elements in each array, value range from 0 to 20) – "*TankIds*" – arrays of tank IDs for nozzles (up to 6 integer elements in each array, value range from 0 to 20) – "*PaymentFormIds*" – arrays of payment form IDs for nozzles (up to 6 integer elements in each array, value range from 0 to 10)
Note	1. "*PumpNozzles*" objects, which are absent in the sent arrays, are considered to be not configured (filled with zeroes). Identifiers of tanks for nozzles, which are absent in the sent arrays, are considered to be not configured (filled with zeroes). 2. Array "*TankIds*" is optional and may be omitted, it brings additional information specifying linkage between pump nozzle and tank. 3. Array "*PaymentFormIds*" is optional and may be omitted, it brings additional information specifying default payment form for pump nozzle. 4. Elements absent in "*FuelGradeIds*", "*TankIds*" and "*PaymentFormIds*" arrays are considered to be not configured (equal to 0). 5. Elements subsequently coming in "*FuelGradeIds*", "*TankIds*" and "*PaymentFormIds*" arrays refer to the pump nozzles from 1 to 6 (first element refers to the first nozzle, second element – to the second nozzle, etc).
Permission required	Any permission among listed below: – *Configuration*
Response	Confirmation response message (in case of successful execution) or error response message (in case of any problem)

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"SetPumpNozzlesConfiguration",
	"Data":{
	"PumpNozzles":[{
	"PumpId":1,
	"FuelGradeIds":[1,2,3],
	"TankIds":[1,2,3]
	},{
	"PumpId":2,
	"FuelGradeIds":[1,2,3]
	}]
	}
	}]
	}

68. PumpNozzlesConfiguration response

Purpose	Returns pump nozzles configuration: assignment of pump nozzles to fuel grades, tanks and payment forms
Data	"*PumpNozzles*" – array of objects for configuration of pump nozzles: – "*PumpId*" – identifier of pump (integer, range from 1 to 120) – "*FuelGradeIds*" – array of fuel grade IDs for nozzles (up to 6 integer elements in each array, value range from 0 to 20) – "*TankIds*" – array of tank IDs for nozzles (up to 6 integer elements in each array, value range from 0 to 20) – "*PaymentFormIds*" – arrays of payment form IDs for nozzles (up to 6 integer elements in each array, value range from 0 to 10)
Note	1. "*PumpNozzles*" objects, which are absent in the sent arrays, are considered to be not configured (filled with zeroes). Identifiers of tanks for nozzles, which are absent in the sent arrays, are considered to be not configured (filled with zeroes). 2. Array "*TankIds*" is optional and may be omitted, it bring additional information specifying linkage between pump nozzle and tank. 3. Array "*PaymentFormIds*" is optional and may be omitted, it brings additional information specifying default payment form for pump nozzle. 4. Elements absent in "*FuelGradeIds*", "*TankIds*" and "*PaymentFormIds*" arrays are considered to be not configured (equal to 0). 5. Elements subsequently coming in "*FuelGradeIds*", "*TankIds*" and "*PaymentFormIds*" arrays refer to the pump nozzles from 1 to 6 (first element refers to the first nozzle, second element – to the second nozzle, etc).

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"PumpNozzlesConfiguration",
	"Data":{
	"PumpNozzles":[{
	"PumpId":1,
	"FuelGradeIds":[1,2,3] ,
	"TankIds":[1,2,3]
	},{
	"PumpId":2,
	"FuelGradeIds":[1,2,3]
	}]
	}
	}]
	}

69. GetTanksConfiguration request

Purpose	Gets tanks configuration: tanks height and assignment to fuel grades
Data	Absent
Permission required	Any permission among listed below: – *Configuration* – *Control* – *Monitoring* – *Reports* – *Payments* – *Shifts*
Response	TanksConfiguration response
Note	It is assumed that tanks' IDs completely correspond to probes' IDs in meaning tank with ID 1 corresponds probe with ID 1, tank with ID 2 corresponds to probe with ID 2 and so on.

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"GetTanksConfiguration"
	}]
	}

70. SetTanksConfiguration request

Purpose	Sets tanks configuration: tanks height and assignment to fuel grades
Data	"*Tanks*" – array of objects for configuration of tanks: – "*Id*" – identifier of tank (integer, range from 1 to 20) – "*FuelGradeId*" – identifier of fuel grade (integer, range from 1 to 20) – "*Height*" – height of tank in mm (integer, range from 1 to 99999) – "*CriticalHighProductAlarmHeight*"– maximum critical product height in mm (integer, range from 1 to 99999), if product height is more than this value – alarm should be triggered – "*HighProductAlarmHeight*" – maximum allowed product height in mm (integer, range from 1 to 99999), if product height is more than this value – alarm should be triggered – "*LowProductAlarmHeight*" – minimum allowed product height in mm (integer, range from 1 to 99999), if product height is lesser than this value – alarm should be triggered – "*CriticalLowProductAlarmHeight*" – minimum critical product height in mm (integer, range from 1 to 99999), if product height is lesser than this value – alarm should be triggered – "*HighWaterAlarmHeight*" – maximum allowed water height in mm (integer, range from 1 to 99999), if water height is more than this value – alarm should be triggered – "*StopPumpsAtCriticalLowProductHeight*" – option to automatically stop pumps and not to allows pumps sales from the tank when the tank's fuel height equals to configured minimum critical product height in tank or is lower than it (Boolean, possible values: "*true*", "*false*") – "*AutomaticCalibrationEnabled*" – flag to enable tank automatic calibration (Boolean, possible values: "*true*", "*false*")
Note	1. "*Tanks*" objects, which are absent in the sent arrays, are considered to be not configured (filled with zeroes). 2. It is assumed that tanks' IDs completely correspond to probes' IDs in meaning tank with ID 1 corresponds probe with ID 1, tank with ID 2 corresponds to probe with ID 2 and so on. 3. Fields "*CriticalHighProductAlarmHeight*", "*HighProductAlarmHeight*", "*LowProductAlarmHeight*", "*CriticalLowProductAlarmHeight*", "*HighWaterAlarmHeight*" are optional. 4. If any of the fields "*CriticalHighProductAlarmHeight*", "*HighProductAlarmHeight*", "*LowProductAlarmHeight*", "*CriticalLowProductAlarmHeight*", "*HighWaterAlarmHeight*" is used – then its value should be smaller than value of field "*Height*". 5. If both fields "*CriticalHighProductAlarmHeight*" and "*HighProductAlarmHeight*" are used – then value of field "*CriticalHighProductAlarmHeight*" should be bigger than value of field "*HighProductAlarmHeight*". 6. If both fields "*CriticalLowProductAlarmHeight*" and "*LowProductAlarmHeight*" are used – then value of field "*lLowProductAlarmHeight*" should be bigger than value of field "*CriticalLowProductAlarmHeight*". 7. If field "*LowWaterAlarmHeight*" is used – then its value should be smaller than values of fields "*CriticalHighProductAlarmHeight*", "*HighProductAlarmHeight*", "*LowProductAlarmHeight*", "*CriticalLowProductAlarmHeight*", if any of them is used.

	8. Field "*AutomaticCalibrationEnabled*" is optional.
Permission required	Any permission among listed below: – *Configuration*
Response	Confirmation response message (in case of successful execution) or error response message (in case of any problem)

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"SetTanksConfiguration",
	"Data":{
	"Tanks":[{
	"Id":1,
	"FuelGradeId":1,
	"Height":3500,
	"CriticalHighProductAlarmHeight":3475,
	"HighProductAlarmHeight":3450,
	"LowProductAlarmHeight":50,
	"CriticalLowProductAlarmHeight":25,
	"HighWaterAlarmHeight":20,
	"StopPumpsAtCriticalLowProductHeight":true
	},{
	"Id":2,
	"FuelGradeId":2,
	"Height":3300,
	"CriticalHighProductAlarmHeight":3275,
	"HighProductAlarmHeight":3250,
	"LowProductAlarmHeight":50,
	"CriticalLowProductAlarmHeight":25,
	"HighWaterAlarmHeight":20,
	"StopPumpsAtCriticalLowProductHeight":false
	}]
	}
	}]
	}

71. TanksConfiguration response

Purpose	Returns tanks configuration: tanks height and assignment to fuel grades
Data	"*Tanks*" – array of objects for configuration of tanks: – "*Id*" – identifier of tank (integer, range from 1 to 20) – "*FuelGradeId*" – identifier of fuel grade (integer, range from 1 to 20) – "*Height*" – height of tank in mm (integer, range from 1 to 99999) – "*CriticalHighProductAlarmHeight*"– maximum critical product height in mm (integer, range from 1 to 99999), if product height is more than this value – alarm should be triggered – "*HighProductAlarmHeight*" – maximum allowed product height in mm (integer, range from 1 to 99999), if product height is more than this value – alarm should be triggered – "*LowProductAlarmHeight*" – minimum allowed product height in mm (integer, range from 1 to 99999), if product height is lesser than this value – alarm should be triggered – "*CriticalLowProductAlarmHeight*" – minimum critical product height in mm (integer, range from 1 to 99999), if product height is lesser than this value – alarm should be triggered – "*HighWaterAlarmHeight*" – maximum allowed water height in mm (integer, range from 1 to 99999), if water height is more than this value – alarm should be triggered – "*StopPumpsAtCriticalLowProductHeight*" – option to automatically stop pumps and not to allows pumps sales from the tank when the tank's fuel height equals to configured minimum critical product height in tank or is lower than it (Boolean, possible values: "*true*", "*false*") – "*AutomaticCalibrationEnabled*" – flag showing whether tank automatic calibration is enabled (Boolean, possible values: "*true*", "*false*") – "*AutomaticCalibrationReadyForGeneration*" – flag showing whether tank automatic calibration chart is ready to be generated (Boolean, possible values: "*true*", "*false*")
Note	1. "*Tanks*" objects, which are absent in the sent arrays, are considered to be not configured (filled with zeroes). 2. It is assumed that tanks' IDs completely correspond to probes' IDs in meaning tank with ID 1 corresponds probe with ID 1, tank with ID 2 corresponds to probe with ID 2 and so on. 3. Fields "*CriticalHighProductAlarmHeight*", "*HighProductAlarmHeight*", "*LowProductAlarmHeight*", "*CriticalLowProductAlarmHeight*", "*HighWaterAlarmHeight*" are optional.

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"TanksConfiguration",
	"Data":{
	"Tanks":[{
	"Id":1,
	"FuelGradeId":1,

```
          "Height":3500,
          "CriticalHighProductAlarmHeight":3475,
          "HighProductAlarmHeight":3450,
          "LowProductAlarmHeight":50,
          "CriticalLowProductAlarmHeight":25,
          "HighWaterAlarmHeight":20,
          "StopPumpsAtCriticalLowProductHeight":true,
          "AutomaticCalibrationEnabled":false,
          "AutomaticCalibrationReadyForGeneration":false
        },{
          "Id":2,
          "FuelGradeId":2,
          "Height":3300,
          "CriticalHighProductAlarmHeight":3275,
          "HighProductAlarmHeight":3250,
          "LowProductAlarmHeight":50,
          "CriticalLowProductAlarmHeight":25,
          "HighWaterAlarmHeight":20,
          "StopPumpsAtCriticalLowProductHeight":false,
          "AutomaticCalibrationEnabled":false,
          "AutomaticCalibrationReadyForGeneration":false
        }]
      }
    }]
}
```

72. GetPriceBoardsConfiguration request

Purpose	Gets price boards configuration
Data	Absent
Permission required	Any permission among listed below: – *Configuration* – *Control* – *Monitoring* – *Reports* – *Payments* – *Shifts*
Response	PriceBoardsConfiguration response

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"GetPriceBoardsConfiguration"
	}]
	}

73. SetPriceBoardsConfiguration request

Purpose	Sets price boards configuration
Data	"*Ports*" – array of objects for configuration of price boards' ports, elements of each object: – "*Id*" – identifier of price board port (string, possible values: "*DISP*", "*LOG*", "*USER*", "*PC*") – "*Protocol*" – value of price board port communication protocol (integer, range from 1 to 99) – "*BaudRate*" – value of price board port baud rate (integer, range from 1 to 99) "*PriceBoards*" – array of objects for configuration of price boards, elements of each object: – "*Id*" – identifier of price board (integer, range from 1 to 5) – "*Port*" – identifier of price board port (string, possible values: "*DISP*", "*LOG*", "*USER*", "*PC*") – "*Address*" – value of price board communication address (integer, range from 1 to 99) – "*FuelGradeIds*" – array of fuel grade IDs for nozzles (up to 10 integer elements in each array, value range from 0 to 20)
Note	1. "*Ports*" objects and "*PriceBoards*" objects, which are absent in the sent arrays, are considered to be not configured (filled with zeroes). 2. Elements absent in "*FuelGradeIds*" are considered to be not configured (equal to 0). 3. Prices for price boards are taken from fuel grades configuration in the PTS-2 controller, which are configured using SetFuelGradesConfiguration request. Once configuration for price boards and fuel grades is set – then the PTS-2 controller will automatically send the prices to the price boards without any additional request or action needed.
Permission required	Any permission among listed below: – *Configuration*
Response	Confirmation response message (in case of successful execution) or error response message (in case of any problem)

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"SetPriceBoardsConfiguration",
	"Data":{
	"Ports":[{
	"Id":"DISP",
	"Protocol":1,
	"BaudRate":4
	}],
	"PriceBoards":[{
	"Id":1,
	"Port":"DISP",

```
            "Address":1,
            "FuelGradeIds":[1,2,3]
          }]
        }
    }]
}
```

74. PriceBoardsConfiguration response

Purpose	Returns price boards configuration
Data	"*Ports*" – array of objects for configuration of price boards' ports, elements of each object: – "*Id*" – identifier of price board port (string, possible values: "*DISP*", "*LOG*", "*USER*", "*PC*") – "*Protocol*" – value of price board port communication protocol (integer, range from 1 to 99) – "*BaudRate*" – value of price board port baud rate (integer, range from 1 to 99) "*PriceBoards*" – array of objects for configuration of price boards, elements of each object: – "*Id*" – identifier of price board (integer, range from 1 to 5) – "*Port*" – identifier of price board port (string, possible values: "*DISP*", "*LOG*", "*USER*", "*PC*") – "*Address*" – value of price board communication address (integer, range from 1 to 99) – "*FuelGradeIds*" – array of fuel grade IDs for nozzles (up to 10 integer elements in each array, value range from 0 to 20)
Note	1. "*Ports*" objects and "*PriceBoards*" objects, which are absent in the sent arrays, are considered to be not configured (filled with zeroes). 2. Elements absent in "*FuelGradeIds*" are considered to be not configured (equal to 0).

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"PriceBoardsConfiguration",
	"Data":{
	"Ports":[{
	"Id":"DISP",
	"Protocol":1,
	"BaudRate":4
	}],
	"PriceBoards":[{
	"Id":1,
	"Port":"DISP",
	"Address":1,
	"FuelGradeIds":[1,2,3]
	}]
	}
	}]
	}

75. GetReadersConfiguration request

Purpose	Gets readers configuration
Data	Absent
Permission required	Any permission among listed below: – *Configuration* – *Control* – *Monitoring* – *Reports* – *Payments* – *Shifts*
Response	ReadersConfiguration response

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"GetReadersConfiguration"
	}]
	}

76. SetReadersConfiguration request

Purpose	Sets readers configuration
Data	"*Ports*" – array of objects for configuration of readers' ports, elements of each object: – "*Id*" – identifier of reader port (string, possible values: "*DISP*", "*LOG*", "*USER*", "*PC*") – "*Protocol*" – value of reader port communication protocol (integer, range from 1 to 99) – "*BaudRate*" – value of reader port baud rate (integer, range from 1 to 99) "*Readers*" – array of objects for configuration of readers, elements of each object: – "*Id*" – identifier of reader (integer, range from 1 to 120) – "*Port*" – identifier of reader port (string, possible values: "*DISP*", "*LOG*", "*USER*", "*PC*") – "*Address*" – value of reader communication address (integer, range from 1 to 99) – "*PumpId*" – identifier of pump, to which the reader is linked (integer, value range from 0 to 120) – "*NozzleNumberForNozzleReader*" – nozzle number for nozzle reader (integer, range from 0 to 6, value 0 means that reader is not used for nozzle) – "*NozzleReaderIdentifier*" – identifier for nozzle reader (string, up to 8 hexadecimal symbols)
Notes	1. "*Ports*" objects and "*Readers*" objects, which are absent in the sent arrays, are considered to be not configured (filled with zeroes). 2. In case if field "*PumpId*" is set to value 0 – then given reader is used to operate with any pump. 3. Fields "*IsNozzleReaderPresent*", "*NozzleNumberForNozzleReader*" and "*NozzleReaderIdentifier*" are optional and can be absent. These fields are used only for the nozzle readers used for automatic vehicles identification.
Permission required	Any permission among listed below: – *Configuration*
Response	Confirmation response message (in case of successful execution) or error response message (in case of any problem)

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"SetReadersConfiguration",
	"Data":{
	"Ports":[{
	"Id":"DISP",
	"Protocol":1,
	"BaudRate":4
	}],
	"Readers":[{
	"Id":1,
	"Port":"DISP",

```
            "Address":1,
            "PumpId":1,
            "NozzleNumberForNozzleReader":0,
            "NozzleReaderIdentifier":""
        }]
      }
    }]
}
```

77. ReadersConfiguration response

Purpose	Returns readers configuration
Data	"*Ports*" – array of objects for configuration of readers' ports, elements of each object: – "*Id*" – identifier of reader port (string, possible values: "*DISP*", "*LOG*", "*USER*", "*PC*") – "*Protocol*" – value of reader port communication protocol (integer, range from 1 to 99) – "*BaudRate*" – value of reader port baud rate (integer, range from 1 to 99) "*Readers*" – array of objects for configuration of readers, elements of each object: – "*Id*" – identifier of reader (integer, range from 1 to 120) – "*Port*" – identifier of reader port (string, possible values: "*DISP*", "*LOG*", "*USER*", "*PC*") – "*Address*" – value of reader communication address (integer, range from 1 to 99) – "*PumpId*" – identifier of pump, to which the reader is linked (integer, value range from 0 to 120) – "*NozzleNumberForNozzleReader*" – nozzle number for nozzle reader (integer, range from 0 to 6, value 0 means that reader is not used for nozzle) – "*NozzleReaderIdentifier*" – identifier for nozzle reader (string, up to 8 hexadecimal symbols)
Note	"*Ports*" objects and "*Readers*" objects, which are absent in the sent arrays, are considered to be not configured (filled with zeroes).

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"ReadersConfiguration",
	"Data":{
	"Ports":[{
	"Id":"DISP",
	"Protocol":1,
	"BaudRate":4
	}],
	"Readers":[{
	"Id":1,
	"Port":"DISP",
	"Address":1,
	"PumpId":1,
	"NozzleNumberForNozzleReader":0,
	"NozzleReaderIdentifier":""
	}]
	}
	}]
	}

78. GetUsersConfiguration request

Purpose	Gets users configuration: logins and permissions
Data	Absent
Permission required	Any permission among listed below: – *Configuration* – *Control* – *Monitoring* – *Reports* – *Payments* – *Shifts*
Response	UsersConfiguration response

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"GetUsersConfiguration"
	}]
	}

79. SetUsersConfiguration request

Purpose	Sets users configuration: logins, passwords, permissions
Data	"*Users*" – array of objects for configuration of users: – "*Id*" – identifier of user (integer, range from 1 to 30) – "*Login*" – user login for accessing web server (string, up to 10 ASCII characters) – "*Password*" – user password for accessing web server (string, up to 10 ASCII characters) – "*Permissions*" – object with user permissions: – "*Configuration*" – permission to provide configuration (Boolean, possible values: "*true*", "*false*") – "*Control*" – permission to provide control over pumps and view tanks (Boolean, possible values: "*true*", "*false*") – "*Monitoring*" – permission to monitor pumps and tanks (Boolean, possible values: "*true*", "*false*") – "*Reports*" – permission to view reports (Boolean, possible values: "*true*", "*false*") – "*Payments*" – permission to perform payments (Boolean, possible values: "*true*", "*false*") – "*Shifts*" – permission to perform shift management (Boolean, possible values: "*true*", "*false*")
Note	1. "*Users*" objects, which are absent in the sent arrays, are considered to be not configured (filled with zeroes). 2. Permissions, which are absent in the sent arrays, are considered to be not configured (not set). 3. If the field "*Password*" is empty – then the password of the user is not updated and previous password of the user is not changed. 4. Logins sent In given request should not start from any of system user logins: "*PTS*", "*Unipump*", "*Dart1*", "*Dart2*", "*Dart3*", "*Dart4*".
Permission required	Any permission among listed below: – *Configuration*
Response	Confirmation response message (in case of successful execution) or error response message (in case of any problem)

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"SetUsersConfiguration",
	"Data":{
	"Users":[{
	"Id":1,
	"Login":"admin",
	"Password":"admin",
	"Permissions":{
	"Configuration":true,
	"Control":true,

```
            "Monitoring":true,
            "Reports":true,
            "Payments":true,
            "Shifts":true
          }
        },{
          "Id":2,
          "Login":"Service",
          "Password":"Serviceman",
          "Permissions":{
            "Configuration":true
          }
        }]
      }
    }]
}
```

80. AddUserConfiguration request

Purpose	Adds a new user to users' configuration
Data	"*Login*" – user login (string, up to 10 ASCII characters) "*Password*" – user password (string, up to 10 ASCII characters) "*Permissions*" – object with user permissions: – "*Configuration*" – permission to provide configuration (Boolean, possible values: "*true*", "*false*") – "*Control*" – permission to provide control over pumps and view tanks (Boolean, possible values: "*true*", "*false*") – "*Monitoring*" – permission to monitor pumps and tanks (Boolean, possible values: "*true*", "*false*") – "*Reports*" – permission to view reports (Boolean, possible values: "*true*", "*false*") – "*Payments*" – permission to perform payments (Boolean, possible values: "*true*", "*false*") – "*Shifts*" – permission to perform shift management (Boolean, possible values: "*true*", "*false*")
Note	1. Logins sent In given request should not start from any of system user logins: "*PTS*", "*Unipump*", "*Dart1*", "*Dart2*", "*Dart3*", "*Dart4*". 2. Permissions, which are absent in the sent arrays, are considered to be not configured (not set).
Permission required	Any permission among listed below: – *Configuration*
Response	Confirmation response message (in case of successful execution) or error response message (in case of any problem)

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"AddUserConfiguration",
	"Data":{
	"Login":"service",
	"Password":"service",
	"Permissions":{
	"Configuration":true,
	"Control":true,
	"Monitoring":true,
	"Reports":false,
	"Payments":true,
	"Shifts":true
	}
	}
	}]
	}

81. EditUserConfiguration request

Purpose	Edits existing user from users' configuration
Data	"*UserId*" – identifier of user in UsersConfiguration (integer, range from 1 to 30) "*Login*" – new user login (string, up to 10 ASCII characters) "*Password*" – new user password (string, up to 10 ASCII characters) "*Permissions*" – object with user permissions: – "*Configuration*" – permission to provide configuration (Boolean, possible values: "*true*", "*false*") – "*Control*" – permission to provide control over pumps and view tanks (Boolean, possible values: "*true*", "*false*") – "*Monitoring*" – permission to monitor pumps and tanks (Boolean, possible values: "*true*", "*false*") – "*Reports*" – permission to view reports (Boolean, possible values: "*true*", "*false*") – "*Payments*" – permission to perform payments (Boolean, possible values: "*true*", "*false*") – "*Shifts*" – permission to perform shift management (Boolean, possible values: "*true*", "*false*")
Notes	1. Logins sent In given request should not start from any of these system user logins: "*PTS*", "*Unipump*", "*Dart1*", "*Dart2*", "*Dart3*", "*Dart4*" 2. If the field "*Password*" is empty – then the password of the user is not updated and previous password of the user is not changed. 3. Permissions, which are absent in the sent arrays, are considered to be not configured (not set).
Permission required	Any permission among listed below: – *Configuration*
Response	Confirmation response message (in case of successful execution) or error response message (in case of any problem)

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"EditUserConfiguration",
	"Data":{
	"UserId":2,
	"Login":"service",
	"Password":"service",
	"Permissions":{
	"Configuration":true,
	"Control":true,
	"Monitoring":true,
	"Reports":false,
	"Payments":true,
	"Shifts":true
	}

```
        }
    } ]
}
```

82. DeleteUserConfiguration request

Purpose	Deletes existing user from users' configuration
Data	"*UserId*" – identifier of user in UsersConfiguration (integer, range from 1 to 30)
Permission required	Any permission among listed below: – *Configuration*
Response	Confirmation response message (in case of successful execution) or error response message (in case of any problem)

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"DeleteUserConfiguration",
	"Data":{
	"UserId":2
	}
	}]
	}

83. UsersConfiguration response

Purpose	Returns users configuration: logins, permissions
Data	"*Users*" – array of objects for configuration of users: – "*Id*" – identifier of user (integer, range from 1 to 30) – "*Login*" – user login for accessing web server (string, up to 10 ASCII characters) – "*Permissions*" – object with user permissions: – "*Configuration*" – permission to provide configuration (Boolean, possible values: "*true*", "*false*") – "*Control*" – permission to provide control over pumps and view tanks (Boolean, possible values: "*true*", "*false*") – "*Monitoring*" – permission to monitor pumps and tanks (Boolean, possible values: "*true*", "*false*") – "*Reports*" – permission to view reports (Boolean, possible values: "*true*", "*false*") – "*Payments*" – permission to perform payments (Boolean, possible values: "*true*", "*false*") – "*Shifts*" – permission to perform shift management (Boolean, possible values: "*true*", "*false*")
Note	1. "*Users*" objects, which are absent in the sent arrays, are considered to be not configured (filled with zeroes). 2. Presence "*Permissions*" objects for each of the users in response depends on presence of "*Configuration*" permission for the user sending GetUsersConfiguration request: if the user has this permission – then this user receives a list of permissions of each of the users, otherwise no.

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"UsersConfiguration",
	"Data":{
	"Users":[{
	"Id":1,
	"Login":"admin",
	"Permissions":{
	"Configuration":true,
	"Control":true,
	"Monitoring":true,
	"Reports":true,
	"Payments":true,
	"Shifts":true
	}
	},{
	"Id":2,
	"Login":"Service",

```
          "Permissions":{
            "Configuration":true,
            "Control":false,
            "Monitoring":false,
            "Reports":false
          }
        }]
      }
    }]
}
```

84. GetWirelessDevicesConfiguration request

Purpose	Gets configuration for the wireless connected devices
Data	Absent
Permission required	Any permission among listed below: – *Configuration*
Response	WirelessDevicesConfiguration response

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"GetWirelessDevicesConfiguration"
	}]
	}

85. SetWirelessDevicesConfiguration request

Purpose	Sets configuration for the wireless connected devices
Data	"*Ports*" – array of objects for configuration of controller's ports, elements of each object: – "*Id*" – identifier of port (string, possible values: "*1*", "*2*", "*3*", "*4*", "*DISP*", "*LOG*", "*USER*", "*PC*") – "*WirelessCommunication*" – option to use wireless communication (Boolean, possible values: "*true*", "*false*") "*Devices*" – array of devices configured to controller's port, elements of each object: – "*Id*" – identifier of device (integer, range from 1 to 120) – "*Type*" – type of device (string, possible values: "*Pump*", "*Probe*", "*PriceBoard*", "*Reader*") – "*PortId*" – identifier of device port (string, possible values: "*1*", "*2*", "*3*", "*4*", "*DISP*", "*LOG*", "*USER*", "*PC*") – "*IpAddress*" – IP-address of device (string, format XXX.XXX.XXX.XXX, leading zeroes can be removed) – "*Port*" – number of port (integer, range from 0 to 65535)
Note	"*Ports*" objects and "*Devices*" objects, which are absent in the sent arrays, are considered to be not configured (filled with zeroes)
Permission required	Any permission among listed below: – *Configuration*
Response	Confirmation response message (in case of successful execution) or error response message (in case of any problem)

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"SetWirelessDevicesConfiguration",
	"Data":{
	"Ports":[{
	"Id":"1",
	"WirelessCommunication":true
	},{
	"Id":"USER",
	"WirelessCommunication":false
	}],
	"Devices":[{
	"Type":"Pump",
	"Id":1,
	"PortId":"1",
	"IpAddress":"192.168.1.101",
	"Port":"45454"
	},{
	"Type":"Pump",

```
            "Id":2,
            "PortId":"1",
            "IpAddress":"192.168.1.102",
            "Port":"45454"
          },{
            "Type":"Probe",
            "Id":1,
            "PortId":"USER",
            "IpAddress":"0.0.0.0",
            "Port":"0"
          }]
      }
    }]
}
```

86. WirelessDevicesConfiguration response

Purpose	Returns configuration for the wireless connected devices
Data	"*Ports*" – array of objects for configuration of controller's ports, elements of each object: – "*Id*" – identifier of port (string, possible values: "*1*", "*2*", "*3*", "*4*", "*DISP*", "*LOG*", "*USER*", "*PC*") – "*WirelessCommunication*" – option to use wireless communication (Boolean, possible values: "*true*", "*false*") "*Devices*" – array of devices configured to controller's port, elements of each object: – "*Id*" – identifier of device (integer, range from 1 to 120) – "*Type*" – type of device (string, possible values: "*Pump*", "*Probe*", "*PriceBoard*", "*Reader*") – "*PortId*" – identifier of device port (string, possible values: "*1*", "*2*", "*3*", "*4*", "*DISP*", "*LOG*", "*USER*", "*PC*") – "*IpAddress*" – IP-address of device (string, format XXX.XXX.XXX.XXX, leading zeroes can be removed) – "*Port*" – number of port (integer, range from 0 to 65535)
Note	"*Ports*" objects and "*Devices*" objects, which are absent in the sent arrays, are considered to be not configured (filled with zeroes)

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"WirelessDevicesConfiguration",
	"Data":{
	"Ports":[{
	"Id":"1",
	"WirelessCommunication":true
	},{
	"Id":"USER",
	"WirelessCommunication":false
	}],
	"Devices":[{
	"Type":"Pump",
	"Id":1,
	"PortId":"1",
	"IpAddress":"192.168.1.101",
	"Port":"45454"
	},{
	"Type":"Pump",
	"Id":2,
	"PortId":"1",
	"IpAddress":"192.168.1.102",
	"Port":"45454"

```
        },{
          "Type":"Probe",
          "Id":1,
          "PortId":"USER",
          "IpAddress":"0.0.0.0",
          "Port":"0"
        }]
      }
    }]
}
```

87. GetTagsList request

Purpose	Gets configured tags from controller
Data	"*StartNumber*" – starting number of tag in list to read (integer, range from 1 and higher) "*TotalNumber*" – total number of tags to read (integer, range from 1 and till 100)
Permission required	Any permission among listed below: – *Configuration* – *Control* – *Reports*
Response	TagsList response
Note	1. Field "*StartNumber*" is optional, if it is not present in request – then its value is automatically set to 1. 2. Field "*TotalNumber*" is optional, if it is not present in request – then its value is automatically limited with 100.

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"GetTagsList",
	"Data":{
	"StartNumber":1,
	"TotalNumber":100
	}
	}]
	}

88. TagsList response

Purpose	Returns list of configured tags
Data	Array of objects for each tag: – "*Tag*" – value of tag ID (string, up to 32 hexadecimal symbols) – "*Name*" – name of person, to whom the tag belongs (string, up to 20 ASCII symbols) – "*Valid*" – permission for operation (Boolean, possible values: "*true*", "*false*") – "*PaymentFormId*" – identifier of payment form (integer, range from 0 to 10) – "*PaymentFormName*" – name of payment form (string, up to 10 ASCII characters)
Note	Fields *"PaymentFormId"* and *"PaymentFormName"* are optional, they are present in response only when there is a payment form set for the tag.

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"TagsList",
	"Data":[{
	"Tag":"1234567890ABCDEF",
	"Name":"Marry Plum",
	"PaymentFormId":1,
	"PaymentFormName":"Cash",
	"Valid":true
	},{
	"Tag":"1122334455667788",
	"Name":"Mr. Brown",
	"Valid":false
	}]
	}]
	}

89. SetTagsList request

Purpose	Sets a list of configured tags
Data	Array of objects for each tag: – "*Tag*" – value of tag ID (string, up to 32 hexadecimal symbols) – "*Name*" – name of person, to whom the tag belongs (string, up to 20 ASCII symbols) – "*Valid*" – permission for operation (Boolean, possible values: "*true*", "*false*") – "*PaymentFormId*" – identifier of payment form (integer, range from 1 to 10)
Permission required	Any permission among listed below: – *Configuration*
Response	Confirmation response message (in case of successful execution) or error response message (in case of any problem)
Notes	1. This request clears all previously saved tags in configuration. 2. This request has a limitation of maximum 100 tags to be saved. To save more tags additionally use AddTagsToList or AddTagToList requests. 3. Field "*PaymentFormId*" is optional and may be omitted, in case if it is present – then it Is used to save payment form identifier for the tag, in this case the PTS-2 controller should have configuration of payment forms (see SetPaymentFormsConfiguration request).

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"SetTagsList",
	"Data":[{
	"Tag":"1234567890ABCDEF",
	"Name":"Marry Plum",
	"Valid":true
	},{
	"Tag":"1122334455667788",
	"Name":"Mr. Brown",
	"Valid":false
	}]
	}]
	}

90. AddTagsToList request

Purpose	Adds a new tags to list of configured tags
Data	Array of objects for each tag: – "*Tag*" – value of tag ID (string, up to 32 hexadecimal symbols) – "*Name*" – name of person, to whom the tag belongs (string, up to 20 ASCII symbols) – "*Valid*" – permission for operation (Boolean, possible values: "*true*", "*false*") – "*PaymentFormId*" – identifier of payment form (integer, range from 1 to 10)
Permission required	Any permission among listed below: – *Configuration*
Note	1. This request has a limitation of maximum 100 tags to be saved at a single request. 2. In case if any of the tags is already present in configuration – then it is overwritten. 3. Field "*PaymentFormId*" is optional and may be omitted, in case if it is present – then it Is used to save payment form identifier for the tag, in this case the PTS-2 controller should have configuration of payment forms (see SetPaymentFormsConfiguration request).
Response	Confirmation response message (in case of successful execution) or error response message (in case of any problem)

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"AddTagsToList",
	"Data":[{
	"Tag":"AABBCCDDEEFF",
	"Name":"Ms. Kitty",
	"Valid":true
	},{
	"Tag":"ABCDEF1234567890",
	"Name":"Berry Towel",
	"Valid":false
	}]
	}]
	}

91. AddTagToList request

Purpose	Adds a new tag to list of configured tags
Data	"*Tag*" – value of tag ID (string, up to 32 hexadecimal symbols) "*Name*" – name of person, to whom the tag belongs (string, up to 20 ASCII symbols) "*Valid*" – permission for operation (Boolean, possible values: "*true*", "*false*") "*PaymentFormId*" – identifier of payment form (integer, range from 1 to 10)
Permission required	Any permission among listed below: – *Configuration*
Note	Field "*PaymentFormId*" is optional and may be omitted, in case if it is present – then it Is used to save payment form identifier for the tag, in this case the PTS-2 controller should have configuration of payment forms (see SetPaymentFormsConfiguration request).
Response	Confirmation response message (in case of successful execution) or error response message (in case of any problem)

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"AddTagToList",
	"Data":{
	"Tag":"1234567890ABCDEF",
	"Name":"Marry Plum",
	"Valid":true
	}
	}]
	}

92. EditTagInList request

Purpose	Edits existing tag in list of configured tags
Data	"*Tag*" – value of tag ID (string, up to 32 hexadecimal symbols) "*Name*" – name of person, to whom the tag belongs (string, up to 20 ASCII symbols) "*Valid*" – permission for operation (Boolean, possible values: "*true*", "*false*") "*PaymentFormId*" – identifier of payment form (integer, range from 1 to 10)
Permission required	Any permission among listed below: – *Configuration*
Note	Field "*PaymentFormId*" is optional and may be omitted, in case if it is present – then it Is used to save payment form identifier for the tag, in this case the PTS-2 controller should have configuration of payment forms (see SetPaymentFormsConfiguration request).
Response	Confirmation response message (in case of successful execution) or error response message (in case of any problem)

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"EditTagInList",
	"Data":{
	"Tag":"1234567890ABCDEF",
	"Name":"Marry Plum",
	"Valid":true
	}
	}]
	}

93. DeleteTagFromList request

Purpose	Deletes existing tag from list of configured tags
Data	"*Tag*" – value of tag ID (string, up to 32 hexadecimal symbols)
Permission required	Any permission among listed below: – *Configuration*
Response	Confirmation response message (in case of successful execution) or error response message (in case of any problem)

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"DeleteTagFromList",
	"Data":{
	"Tag":"1234567890ABCDEF"
	}
	}]
	}

94. GetPortLoggingConfiguration request

Purpose	Gets port logging configuration
Data	Absent
Permission required	Any permission among listed below: – *Configuration*
Response	PortLoggingConfiguration response

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"GetPortLoggingConfiguration"
	}]
	}

95. SetPortLoggingConfiguration request

Purpose	Sets port logging configuration
Data	"*Port*" – identifier of port (string, possible values: "*None*", "*PumpPort1*", "*PumpPort2*", "*PumpPort3*", "*PumpPort4*", "*DISP*", "*LOG*", "*USER*", "*PC*") "*DateTimeStop*" – date and time of port logging stop in format *YYYY-MM-DDThh:mm:ss* (string, 19 symbols), where: – *YYYY* – year (range from 2000 to 2099) – *MM* – month (range from 01 to 12) – *DD* – day (range from 01 to 31) – *T* – separator between date and time fields – *hh* – hours (range from 00 to 23) – *mm* – minutes (range from 00 to 59) – *ss* – seconds (range from 00 to 59)
Permission required	Any permission among listed below: – *Configuration*
Response	Confirmation response message (in case of successful execution) or error response message (in case of any problem)
Note	Field "*DateTimeStop*" may not be present in case if field "*Port*" is sent with value "*None*" – it leads to stopping of the logging process.

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"SetPortLoggingConfiguration",
	"Data":{
	"Port":"PumpPort1",
	"DateTimeStop":"2019-11-10T12:34:56"
	}
	}]
	}

96. PortLoggingConfiguration response

Purpose	Returns port logging configuration
Data	"*Port*" – identifier of port (string, possible values: "*None*", "*PumpPort1*", "*PumpPort2*", "*PumpPort3*", "*PumpPort4*", "*DISP*", "*LOG*", "*USER*", "*PC*") "*DateTimeStop*" – date and time of port logging stop in format *YYYY-MM-DDThh:mm:ss* (string, 19 symbols), where: – *YYYY* – year (range from 2000 to 2099) – *MM* – month (range from 01 to 12) – *DD* – day (range from 01 to 31) – *T* – separator between date and time fields – *hh* – hours (range from 00 to 23) – *mm* – minutes (range from 00 to 59) – *ss* – seconds (range from 00 to 59) "*LogIsRunning*" – present state of logging process (Boolean, possible values: "*true*", "*false*")

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"PortLoggingConfiguration",
	"Data":{
	"Port":"PumpPort1",
	"DateTimeStop":"2019-11-10T12:34:56",
	"LogIsRunning":true
	}
	}]
	}

97. GetRemoteServerLoggingConfiguration request

Purpose	Gets state of logging process of communication with a remote server
Data	Absent
Permission required	Any permission among listed below: – *Configuration*
Response	RemoteServerLoggingConfiguration response

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"GetRemoteServerLoggingConfiguration"
	}]
	}

98. SetRemoteServerLoggingConfiguration request

Purpose	Sets state of logging process of communication with a remote server
Data	"*Enabled*" – present state of logging process (Boolean, possible values: "*true*", "*false*")
Permission required	Any permission among listed below: – *Configuration*
Response	Confirmation response message (in case of successful execution) or error response message (in case of any problem)

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"SetRemoteServerLoggingConfiguration",
	"Data":{
	"Enabled":true
	}
	}]
	}

99. RemoteServerLoggingConfiguration response

Purpose	Returns state of logging process of communication with a remote server
Data	"*Enabled*" – present state of logging process (Boolean, possible values: "*true*", "*false*")

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"RemoteServerLoggingConfiguration",
	"Data":{
	"Enabled":true
	}
	}]
	}

100. BackupConfiguration request

Purpose	Backups configuration (except users' configuration) and saves to SD flash disk file named *Config.js*
Data	"*RestoreConfigurationOnStartup*" – option to automatically restore configuration on next startup (Boolean, possible values: "*true*", "*false*")
Permission required	Any permission among listed below: – *Configuration*
Response	Confirmation response message (in case of successful execution) or error response message (in case of any problem)

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"BackupConfiguration",
	"Data":{
	"RestoreConfigurationOnStartup":true
	}
	}]
	}

101. RestoreConfiguration request

Purpose	Restores configuration (except users' configuration) from SD flash disk file named *Config.js*
Data	Absent
Permission required	Any permission among listed below: – *Configuration*
Response	Confirmation response message (in case of successful execution) or error response message (in case of any problem)

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"RestoreConfiguration"
	}]
	}

102. GetUploadedRecordsInformation request

Purpose	Gets information about the number of all types of uploaded records to a remote server and total number of records in database
Data	Absent
Permission required	Any permission among listed below: – *Configuration* – *Control* – *Monitoring* – *Reports* – *Payments* – *Shifts*
Response	UploadedRecordsInformation response

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"GetUploadedRecordsInformation"
	}]
	}

103. UploadedRecordsInformation response

Purpose	Returns information about the number of all types of uploaded records to a remote server and total number of records in database
Data	"*PumpTransactionsUploaded*" – number of pump transactions uploaded to a remote server (integer) "*PumpTransactionsTotal*" – total number of pump transactions stored in controller (integer) "*TankMeasurementsUploaded*" – number of tank measurements uploaded to a remote server (integer) "*TankMeasurementsTotal*" – total number of tank measurements stored in controller (integer) "*InTankDeliveriesUploaded*" – number of in-tank deliveries uploaded to a remote server (integer) "*InTankDeliveriesTotal*" – total number of in-tank deliveries stored in controller (integer) "*GpsRecordsUploaded*" – number of GPS records uploaded to a remote server (integer) "*GpsRecordsTotal*" – total number of GPS records stored in controller (integer) "*AlertRecordsUploaded*" – number of alert records uploaded to a remote server (integer) "*AlertRecordsTotal*" – total number of alert records stored in controller (integer) "*PaymentsUploaded*" – number of payments uploaded to a remote server (integer) "*PaymentsTotal*" – total number of payments records stored in controller (integer) "*ShiftsUploaded*" – number of working shifts uploaded to a remote server (integer) "*ShiftsTotal*" – total number of working shifts stored in controller (integer)
Notes	1. Value of field '*PumpTransactionsUploaded'* should always be lesser or equal to value of field '*PumpTransactionsTotal*'. 2. Value of field '*TankMeasurementsUploaded*' should always be lesser or equal to value of field '*TankMeasurementsTotal*'. 3. Value of field '*InTankDeliveriesUploaded'* should always be lesser or equal to value of field '*InTankDeliveriesTotal'*. 4. Value of field '*GpsRecordsUploaded*' should always be lesser or equal to value of field '*GpsRecordsTotal*'. 5. Value of field '*AlertRecordsUploaded*' should always be lesser or equal to value of field '*AlertRecordsTotal*'. 6. Value of field '*PaymentsUploaded'* should always be lesser or equal to value of field '*PaymentsTotal'*. 7. Value of field '*ShiftsUploaded'* should always be lesser or equal to value of field '*ShiftsTotal'*.

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"UploadedRecordsInformation",
	"Data":{
	"PumpTransactionsUploaded":10,
	"PumpTransactionsTotal":15,

```
        "TankMeasurementsUploaded":712,
        "TankMeasurementsTotal":712,
        "InTankDeliveriesUploaded":112,
        "InTankDeliveriesTotal":112,
        "GpsRecordsUploaded":130,
        "GpsRecordsTotal":130,
        "AlertRecordsUploaded":130,
        "AlertRecordsTotal":130,
        "PaymentsUploaded":130,
        "PaymentsTotal":130,
        "ShiftsUploaded":130,
        "ShiftsTotal":130
      }
    }]
}
```

104. SetUploadedRecordsNumber request

Purpose	Sets number of uploaded records to a remote server
Data	"*PumpTransactions*" – number of pump transactions uploaded to a remote server (integer) "*TankMeasurements*" – number of tank measurements uploaded to a remote server (integer) "*InTankDeliveries*" – number of in-tank deliveries uploaded to a remote server (integer) "*GpsRecords*" – number of GPS records uploaded to a remote server (integer) "*AlertRecords*" – number of alert records uploaded to a remote server (integer) "*Payments*" – number of payments uploaded to a remote server (integer) "*Shifts*" – number of working shifts uploaded to a remote server (integer)
Permission required	Any permission among listed below: – *Configuration*
Response	Confirmation response message (in case of successful execution) or error response message (in case of any problem)
Note	Each of the fields is optional.

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"SetUploadedRecordsNumber",
	"Data":{
	"PumpTransactions":114
	}
	}]
	}

105. ClearUploadedRecords request

Purpose	Clears number of uploaded records to a remote server
Data	"*ClearPumpTransactions*" – option to clear a number of pump transactions uploaded to a remote server (Boolean, possible values: "*true*", "*false*") "*ClearTankMeasurements*" – option to clear a number of tank measurements uploaded to a remote server (Boolean, possible values: "*true*", "*false*") "*ClearInTankDeliveries*" – option to clear a number of in-tank deliveries uploaded to a remote server (Boolean, possible values: "*true*", "*false*") "*ClearGpsRecords*" – option to clear a number of GPS records uploaded to a remote server (Boolean, possible values: "*true*", "*false*") "*ClearAlertRecords*" – option to clear a number of alert records uploaded to a remote server (Boolean, possible values: "*true*", "*false*") "*ClearPayments*" – option to clear a number of payments uploaded to a remote server (Boolean, possible values: "*true*", "*false*") "*ClearShifts*" – option to clear a number of working shifts uploaded to a remote server (Boolean, possible values: "*true*", "*false*")
Permission required	Any permission among listed below: – *Configuration*
Response	Confirmation response message (in case of successful execution) or error response message (in case of any problem)

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"ClearUploadedRecords",
	"Data":{
	"ClearPumpTransactions":true,
	"ClearTankMeasurements":false,
	"ClearInTankDeliveries":false,
	"ClearGpsRecords":false,
	"ClearAlertRecords":false,
	"ClearPayments":false,
	"ClearShifts":false
	}
	}]
	}

106. GetShiftsOperation request

Purpose	Returns state of leading of working shifts in the controller
Data	Absent
Permission required	Any permission among listed below: – *Configuration* – *Control* – *Monitoring* – *Reports* – *Payments* – *Shifts*
Response	Confirmation response message (in case of successful execution) or error response message (in case of any problem)

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"GetShiftsOperation"
	}]
	}

107. SetShiftsOperation request

Purpose	Enables or disables leading of working shifts in the controller
Data	"*Enabled*" – flag showing if leading of working shift is enabled (Boolean, possible values: "*true*", "*false*")
Permission required	Any permission among listed below: – Configuration
Response	Confirmation response message (in case of successful execution) or error response message (in case of any problem)

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"SetShiftsOperation",
	"Data":{
	"Enabled":true
	}
	}]
	}

108. ShiftsOperation response

Purpose	Returns state of leading of working shifts in the controller
Data	"*Enabled*" – flag showing if leading of working shift is enabled (Boolean, possible values: "*true*", "*false*")

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"ShiftsOperation",
	"Data":{
	"Enabled":true
	}
	}]
	}

109. GetPortsState request

Purpose	Gets state of ports
Data	Absent
Permission required	Any permission among listed below: – *Configuration* – *Control* – *Monitoring*
Response	PortsState response

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"GetPortsState"
	}]
	}

110. PortsState response

Purpose	Returns state of ports
Data	"*PumpPort1*" – object with data related to port: – "*Busy*" – state of port (Boolean, possible values: "*true*", "*false*") "*PumpPort2*" – object with data related to port: – "*Busy*" – state of port (Boolean, possible values: "*true*", "*false*") "*PumpPort3*" – object with data related to port: – "*Busy*" – state of port (Boolean, possible values: "*true*", "*false*") "*PumpPort4*" – object with data related to port: – "*Busy*" – state of port (Boolean, possible values: "*true*", "*false*") "*DISP*" – object with data related to port: – "*Busy*" – state of port (Boolean, possible values: "*true*", "*false*") – "*DeviceType*" – type of device used by the port (string, possible values: "*Probe*", "*PriceBoard*" and "*Reader*") "*LOG*" – object with data related to port: – "*Busy*" – state of port (Boolean, possible values: "*true*", "*false*") – "*DeviceType*" – type of device used by the port (string, possible values: "*Probe*", "*PriceBoard*" and "*Reader*") "*USER*" – object with data related to port: – "*Busy*" – state of port (Boolean, possible values: "*true*", "*false*") – "*DeviceType*" – type of device used by the port (string, possible values: "*Probe*", "*PriceBoard*" and "*Reader*") "*PC*" – object with data related to port: – "*Busy*" – state of port (Boolean, possible values: "*true*", "*false*") – "*DeviceType*" – type of device used by the port (string, possible values: "*Probe*", "*PriceBoard*" and "*Reader*")
Note	Field "*DeviceType*" may be absent if the field "*Busy*" has value "*false*".

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"PortsState",
	"Data":{
	"PumpPort1":{
	"Busy": true
	},
	"PumpPort2":{
	"Busy": true
	},
	"PumpPort3":{
	"Busy": false
	},
	"PumpPort4":{
	"Busy": false

```
        },
        "DISP":{
          "Busy": true,
          "DeviceType": "Probe"
        },
        "LOG":{
          "Busy": false
        },
        "USER":{
          "Busy": false
        },
        "PC":{
          "Busy": false
        }
      }
    }]
}
```

PUMPS CONTROL REQUESTS AND RESPONSES

This part describes requests and responses used for provision of control and monitoring over pumps. PumpGetStatus request is the main request to be sent regularly by control system to each pump for getting information on its status. PTS-2 controller will response with appropriate pump status whether it is in idle, filling or offline state or in end of transaction. Also, information on requested pump totals, pump prices or read tag ID is returned on this request.

111. PumpGetStatus request

Purpose	Gets pump status, active nozzle, currently executed request, user locking the pump
Data	"*Pump*" – logical number of the pump (decimal, range from 1 to 120)
Permission required	Any permission among listed below: – *Control* – *Monitoring*
Response	Depending on pump status response can be: – PumpIdleStatus response – in case if the pump is in idle state – PumpFillingStatus response – in case if the pump is in filling state – PumpEndOfTransactionStatus response – in case if the pump finished filling – PumpOfflineStatus response – in case if the pump is not connected or not responding – PumpTotals response – in case if total counters were received from the pump – PumpPrices response – in case if prices were sent to or received from the pump – PumpTag response – in case if tag ID was received from the pump – PumpDisplayData response – in case if pump display data was received from the pump

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"PumpGetStatus",
	"Data":{
	"Pump":1
	}
	}]
	}

112. PumpIdleStatus response

Purpose	Returns pump status, active nozzle, currently executed request
Data	"*Pump*" – logical number of the pump (decimal, range from 1 to 120) "*NozzleUp*" – logical number of the nozzle taken up on pump (decimal, range from 1 to 6, 0 means that nozzle is down) "*Nozzle*" – logical number of the nozzle taken up on pump (decimal, range from 1 to 6, 0 means that nozzle is down) "*FuelGradeId*" – identifier of fuel grade (integer, range from 1 to 20) "*FuelGradeName*" – name of fuel grade (string, up to 20 ASCII characters) "*Transaction*" – transaction number (integer, range from 1 to 65535) "*Tag*" – (optional) tag identifier currently brought to pump reader (string, up to 32 hexadecimal symbols) "*NozzlePrices*" – array of floats, which mean pump nozzle prices (each element means a price of the corresponding nozzle starting from nozzle 1, values are float, up to 3 digits after decimal point) "*Request*" - currently executed request on pump (string, up to 20 ASCII symbols) "*OrderedNozzle*" – *ordered* nozzle number (integer, range from 1 to 6) "*OrderedNozzles*" – array of ordered nozzle numbers (integer, up to 6 items, range from 1 to 6) "*OrderedType*" – preset type done (string, up to 8 ASCII symbols), possible values: – "*Volume*": preset done by product volume – "*Amount*": preset done by money amount – "*FullTank*": preset done with full tank, in some situations also meaning preset dose to be entered from pump preset keyboard "*OrderedDose*" - preset dose in volume units or in currency units (float, up to 3 digits after decimal point) "*User*" – name of user, who sent request (string, up to 10 ASCII symbols) "*LastDateTimeStart*" – last date and time of transaction start in format *YYYY-MM-DDThh:mm:ss* (string, 19 symbols), where: – *YYYY* – year (range from 2000 to 2099) – *MM* – month (range from 01 to 12) – *DD* – day (range from 01 to 31) – *T* – separator between date and time fields – *hh* – hours (range from 00 to 23) – *mm* – minutes (range from 00 to 59) – *ss* – seconds (range from 00 to 59) "*LastDateTime*" – last date and time of transaction end in format *YYYY-MM-DDThh:mm:ss* (string, 19 symbols) "*LastNozzle*" – last transaction nozzle (integer, range from 0 to 6) "*LastFuelGradeId*" – identifier of fuel grade (integer, range from 1 to 20) "*LastFuelGradeName*" – name of fuel grade (string, up to 20 ASCII characters) "*LastTransaction*" – last transaction number (integer, range from 0 to 65535) "*LastVolume*" - last transaction volume (float, up to 3 digits after decimal point) "*LastPrice*" – last transaction product price (float, up to 3 digits after decimal point) "*LastAmount*" – last transaction money amount (float, up to 3 digits after decimal point)

	"*LastTotalVolume*" – last transaction volume total counter value (float, up to 3 digits after decimal point) "*LastTotalAmount*" – last transaction money amount total counter value (float, up to 3 digits after decimal point) "*LastUser*" – name of user for last transaction (string, up to 10 ASCII symbols) "*LastReceivedTotalNozzle*" – last received total counters nozzle (integer, range from 0 to 6) "*LastReceivedTotalVolume*" – last received volume total counter value (float, up to 3 digits after decimal point) "*LastReceivedTotalAmount*" – last received money amount total counter value (float, up to 3 digits after decimal point) "*LastFlowRate*" - value of the last filling flow rate in volume units per minute (integer, range from 0 till 9999)
Notes	1. Fields "*LastTotalVolume*" and "*LastTotalAmount*" are optional and are present only when automatic reading of pump totals is activated for the pump. 2. Fields *"FuelGradeId", "LastFuelGradeId", "FuelGradeName"* and *"LastFuelGradeName"* are optional, they are present in response only when PTS-2 controller has pump nozzles configured to specific fuel grades. 3. Fields "*NozzleUp*" and "*Nozzle*" show the same value, field "*Nozzle*" was added for homogeneity with other requests/responses, field "*NozzleUp*" is left for compatibility with previous versions of this protocol. 4. Fields "*LastDateTimeStart*" and "*LastDateTime*" are present in response only in case if such values are present in the controller for the last filling. 5. Field "*Transaction*" is present only if the pump is authorized, but the filling has not started yet. 6. Field "*NozzlePrices*" is present only if the pump nozzles are linked to fuel grades in configuration of the PTS-2 controller. 7. Fields "*LastReceivedTotalNozzle*", "*LastReceivedTotalVolume*" and "*LastReceivedTotalAmount*" are optional, they store value of last received nozzle totals from the pump and are present only when total counter values were requested and received from the pump. 8. Fields "*OrderedNozzle*", "*OrderedNozzles*", "*OrderedType*" and "*OrderedDose*" are optional, they are present only if the pump was authorized, but the filling has not started yet. Only one of the fields "*OrderedNozzle*" or "*OrderedNozzles*" can be present in response depending on how the pump was authorized. If value of field "*OrderedType*" is "*FullTank*" then field "*OrderedDose*" is absent.

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"PumpIdleStatus",
	"Data":{
	"Pump":1,
	"NozzleUp":2,
	"Nozzle":2,
	"Transaction":135,
	"Request":"PumpAuthorize",
	"OrderedNozzle":1,

```
        "OrderedType":"Amount",
        "OrderedDose":25.00,
        "User":"admin",
        "LastDateTimeStart":"2019-05-19T12:44:01",
        "LastDateTime":"2019-05-19T12:47:11",
        "LastNozzle":2,
        "LastTransaction":3,
        "LastVolume":5.00,
        "LastPrice":2.50,
        "LastAmount":12.50,
        "LastTotalVolume":15.99,
        "LastTotalAmount":27.87,
        "LastUser":"PTS",
        "LastReceivedTotalNozzle":2,
        "LastReceivedTotalVolume":15.99,
        "LastReceivedTotalAmount":27.87,
        "LastFlowRate":27
      }
    }]
}
```

113. PumpFillingStatus response

Purpose	Returns pump status during filling process
Data	"*Pump*" – logical number of the pump (integer, range from 1 to 120) "*Nozzle*" – logical number of active nozzle (integer, range from 1 to 6) "*FuelGradeId*" – identifier of fuel grade (integer, range from 1 to 20) "*FuelGradeName*" – name of fuel grade (string, up to 20 ASCII characters) "*Volume*" - currently dispensed volume (float, up to 3 digits after decimal point) "*TCVolume*" - currently dispensed temperature-compensated volume (float, up to 3 digits after decimal point) "*Price*" – product price (float, up to 3 digits after decimal point) "*Amount*" – currently dispensed product money amount (float, up to 3 digits after decimal point) "*Transaction*" – current transaction number (integer, range from 1 to 65535) "*DateTimeStart*" – date and time of transaction start in format *YYYY-MM-DDThh:mm:ss* (string, 19 symbols), where: – *YYYY* – year (range from 2000 to 2099) – *MM* – month (range from 01 to 12) – *DD* – day (range from 01 to 31) – *T* – separator between date and time fields – *hh* – hours (range from 00 to 23) – *mm* – minutes (range from 00 to 59) – *ss* – seconds (range from 00 to 59) "*FlowRate*" - value of the filling flow rate in volume units per minute (integer, range from 0 till 9999) "*Tag*" – (optional) tag identifier used to authorize the pump (string, up to 32 hexadecimal symbols) "*IsSuspended*" – flag showing if the filling is suspended (Boolean, possible values: "*true*", "*false*") "*OrderedType*" – preset type done (string, up to 8 ASCII symbols), possible values: – "*Volume*": preset done by product volume – "*Amount*": preset done by money amount – "*FullTank*": preset done with full tank, in some situations also meaning preset dose to be entered from pump preset keyboard "*OrderedDose*" - preset dose in volume units or in currency units (float, up to 3 digits after decimal point) "*User*" – name of user, to whom current filling belongs (string, up to 10 ASCII symbols)
Notes	1. Value of field "*TCVolume*" is non-zero only in case if: a). pump nozzle is linked to tank and tank has installed probe, measuring temperature and b). pump nozzle is linked to fuel grade with configured temperature-expansion coefficient and c). value of dispensed volume is non-zero. 2. Field "*Tag*" is optional and is present only when there was a tag identifier used to authorize the pump. 3. Fields "*FuelGradeId*" and "*FuelGradeName*" are optional, they are present in response only when PTS-2 controller has pump nozzles configured to specific fuel grades. 4. Field "*DateTimeStart*" is present in response only in case if its value is present in

	the controller for the present filling. 5. If value of field "*OrderedType*" is "*FullTank*" then field "*OrderedDose*" is absent.

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":"1",
	"Type":"PumpFillingStatus",
	"Data":{
	"Pump":1,
	"Nozzle":2,
	"Volume":1.34,
	"TCVolume":1.33,
	"Price":2.50,
	"Amount":12.50,
	"Transaction":37,
	"DateTimeStart":"2019-05-19T12:44:01",
	"FlowRate":27,
	"IsSuspended":false,
	"OrderedType":"Amount",
	"OrderedDose":25.00,
	"User":"admin"
	}
	}]
	}

114. PumpEndOfTransactionStatus response

Purpose	Returns transaction information in the end of filling.
Data	"*Pump*" – logical number of the pump (integer, range from 1 to 120) "*Nozzle*" – transaction nozzle (integer, range from 1 to 6) "*FuelGradeId*" – identifier of fuel grade (integer, range from 1 to 20) "*FuelGradeName*" – name of fuel grade (string, up to 20 ASCII characters) "*Volume*" - transaction volume (float, up to 3 digits after decimal point) "*TCVolume*" - transaction temperature-compensated volume (float, up to 3 digits after decimal point) "*Price*" – product price (float, up to 3 digits after decimal point) "*Amount*" – transaction money amount (float, up to 3 digits after decimal point) "*Transaction*" – transaction number (integer, range from 1 to 65535) "*TotalVolume*" – volume total counter value (float, up to 3 digits after decimal point) "*TotalAmount*" – money amount total counter value (float, up to 3 digits after decimal point) "*DateTimeStart*" – date and time of transaction start in format *YYYY-MM-DDThh:mm:ss* (string, 19 symbols), where: – *YYYY* – year (range from 2000 to 2099) – *MM* – month (range from 01 to 12) – *DD* – day (range from 01 to 31) – *T* – separator between date and time fields – *hh* – hours (range from 00 to 23) – *mm* – minutes (range from 00 to 59) – *ss* – seconds (range from 00 to 59) "*DateTime*" – date and time of transaction end in format *YYYY-MM-DDThh:mm:ss* (string, 19 symbols) , where: – *YYYY* – year (range from 2000 to 2099) – *MM* – month (range from 01 to 12) – *DD* – day (range from 01 to 31) – *T* – separator between date and time fields – *hh* – hours (range from 00 to 23) – *mm* – minutes (range from 00 to 59) – *ss* – seconds (range from 00 to 59) "*FlowRate*" - value of the filling flow rate in volume units per minute (integer, range from 0 till 9999) "*Tag*" – (optional) tag identifier used to authorize the pump (string, up to 32 hexadecimal symbols) "*NozzleUp*" – logical number of the nozzle taken up on pump (decimal, range from 1 to 6, 0 means that nozzle is down) "*OrderedType*" – preset type done (string, up to 8 ASCII symbols), possible values: – "*Volume*": preset done by product volume – "*Amount*": preset done by money amount – "*FullTank*": preset done with full tank, in some situations also meaning preset dose to be entered from pump preset keyboard "*OrderedDose*" - preset dose in volume units or in currency units (float, up to 3 digits after decimal point)

	"*User*" – name of user, to whom current filling belongs (string, up to 10 ASCII symbols)
Notes	1. The PTS-2 controller will keep sending PumpEnfOfTransactionStatus response to PumpGetStatus request until current transaction is closed using PumpCloseTransaction request with the same transaction number as in PumpEnfOfTransactionStatus response. 2. Value of field "*TCVolume*" is non-zero only in case if: a). pump nozzle is linked to tank and tank has installed probe, measuring temperature and b). pump nozzle is linked to fuel grade with configured temperature-expansion coefficient and c). value of dispensed volume is non-zero. 3. Field "*Tag*" is optional and is present only when there was a tag identifier used to authorize the pump. 4. Fields "*TotalVolume*" and "*TotalAmount*" are optional and are present only when automatic reading of pump totals is activated for the pump. 5. Fields "*FuelGradeId*" and "*FuelGradeName*" correspond to nozzle used in transaction, which number is shown in "*Nozzle*" field, these fields are optional, they are present in response only when PTS-2 controller has pump nozzles configured to specific fuel grades. 6. Fields "*DateTimeStart*" and "*DateTime*" are present in response only in case if such values are present in the controller for the present filling. 6. Field "*NozzleUp*" shows a logical number of the nozzle presently taken up on pump (it does not relate to opened pump transaction). 7. If value of field "*OrderedType*" is "*FullTank*" then field "*OrderedDose*" is absent.

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"PumpEndOfTransactionStatus",
	"Data":{
	"Pump":1,
	"Nozzle":2,
	"Volume":5.00,
	"TCVolume":4.98,
	"Price":2.50,
	"Amount":12.50,
	"Transaction":3,
	"TotalVolume":15.99,
	"TotalAmount":27.87,
	"DateTimeStart":"2019-05-19T12:44:01",
	"DateTime":"2019-05-19T12:47:11",
	"NozzleUp":0,
	"FlowRate":27,
	"OrderedTypeAmount":"Amount",
	"OrderedDose":25.00,
	"User":"admin"
	}

```
    } ]
}
```

115. PumpOfflineStatus response

Purpose	Returns pump status when it is not connected or not responding
Data	"*Pump*" – logical number of the pump (integer, range from 1 to 120) "*State*" – state of transaction, possible values: – *"WaitingNozzleUpForAuthorization"* – pump is to be authorized on nozzle up, transaction has not started yet – *"Authorized"* – pump is authorized, transaction has not started yet – *"Filling"* – transaction is in process (filling is going) – *"EndOfTransaction"* – transaction is in process (filling has finished, transaction is still opened) – *"Finished"* – previous transaction is finished "*NozzleUp*" – logical number of the nozzle taken up on pump (decimal, range from 1 to 6, 0 means that nozzle is down) "*Nozzle*" – logical number of the nozzle taken up on pump (decimal, range from 1 to 6, 0 means that nozzle is down) "*FuelGradeId*" – identifier of fuel grade (integer, range from 1 to 20) "*FuelGradeName*" – name of fuel grade (string, up to 20 ASCII characters) "*Volume*" - currently dispensed volume (float, up to 3 digits after decimal point) "*TCVolume*" - currently dispensed temperature-compensated volume (float, up to 3 digits after decimal point) "*Price*" – product price (float, up to 3 digits after decimal point) "*Amount*" – currently dispensed product money amount (float, up to 3 digits after decimal point) "*Transaction*" – current transaction number (integer, range from 1 to 65535) "*TotalVolume*" – volume total counter value (float, up to 3 digits after decimal point) "*TotalAmount*" – money amount total counter value (float, up to 3 digits after decimal point) "*DateTimeStart*" – date and time of transaction start in format *YYYY-MM-DDThh:mm:ss* (string, 19 symbols), where: – *YYYY* – year (range from 2000 to 2099) – *MM* – month (range from 01 to 12) – *DD* – day (range from 01 to 31) – *T* – separator between date and time fields – *hh* – hours (range from 00 to 23) – *mm* – minutes (range from 00 to 59) – *ss* – seconds (range from 00 to 59) "*Tag*" – (optional) tag identifier used to authorize the pump (string, up to 32 hexadecimal symbols) "*IsSuspended*" – flag showing if the filling is suspended (Boolean, possible values: "*true*", "*false*") "*OrderedNozzle*" – *ordered* nozzle number (integer, range from 1 to 6) "*OrderedNozzles*" – array of ordered nozzle numbers (integer, up to 6 items, range from 1 to 6) "*OrderedType*" – preset type done (string, up to 8 ASCII symbols), possible values: – "*Volume*": preset done by product volume – "*Amount*": preset done by money amount

	– "*FullTank*": preset done with full tank, in some situations also meaning preset dose to be entered from pump preset keyboard "*OrderedDose*" - preset dose in volume units or in currency units (float, up to 3 digits after decimal point) "*Request*" - currently executed request on pump (string, up to 20 ASCII symbols) "*User*" – name of user, executing a command on the pump (string, up to 10 ASCII symbols) "*LastDateTimeStart*" – last date and time of transaction start in format *YYYY-MM-DDThh:mm:ss* (string, 19 symbols) "*LastDateTime*" – last date and time of transaction end in format *YYYY-MM-DDThh:mm:ss* (string, 19 symbols) "*LastNozzle*" – last transaction nozzle (integer, range from 0 to 6) "*LastFuelGradeId*" – identifier of fuel grade (integer, range from 1 to 20) "*LastFuelGradeName*" – name of fuel grade (string, up to 20 ASCII characters) "*LastTransaction*" – last transaction number (integer, range from 0 to 65535) "*LastVolume*" - last transaction volume (float, up to 3 digits after decimal point) "*LastPrice*" – last transaction product price (float, up to 3 digits after decimal point) "*LastAmount*" – last transaction money amount (float, up to 3 digits after decimal point) "*LastTotalVolume*" – last volume total counter value (float, up to 3 digits after decimal point) "*LastTotalAmount*" – last money amount total counter value (float, up to 3 digits after decimal point) "*LastFlowRate*" - value of the last filling flow rate in volume units per minute (integer, range from 0 till 9999) "*LastUser*" – name of user for last transaction (string, up to 10 ASCII symbols)
Notes	1. Fields "*NozzleUp*", "*Nozzle*", "*Volume*", "*TCVolume*", "*Amount*", "*Price*", "*FuelGradeId*", "*FuelGradeName*", "*Transaction*", "*TotalVolume*", "*TotalAmount*", "*Tag*", "*DateTimeStart*", "*DateTime*", "*IsSuspended*" are present if there was an unclosed transaction for this pump before communication was lost with it. 2. Fields "*DateTime*", "*TotalVolume*" and "*TotalAmount*" are present if the transaction was finished, but was not closed. 3. Field "*TCVolume*" is optional, its value of field "*TCVolume*" is present and is non-zero only in case if: a). pump nozzle is linked to tank and tank has installed probe, measuring temperature and b). pump nozzle is linked to fuel grade with configured temperature-expansion coefficient and c). value of dispensed volume is non-zero. 4. Field "*Tag*" is optional and is present only when there was a tag identifier used to authorize the pump. 5. Fields "*TotalVolume*", "*TotalAmount*", "*LastTotalVolume*" and "*LastTotalAmount*" are optional and are present only when automatic reading of pump totals is activated for the pump. 6. Fields "*FuelGradeId*", "*FuelGradeName*", "*LastFuelGradeId*" and "*LastFuelGradeName*" are optional, they are present in response only when PTS-2 controller has pump nozzles configured to specific fuel grades. 7. Fields "*DateTimeStart*" and "*DateTime*" are present in response only in case if such values are present in the controller for the present filling. 8. Fields "*OrderedNozzle*", "*OrderedNozzles*", "*OrderedType*" and "*OrderedDose*" are optional, they are present only if the pump was in "*Authorized*" state before

	communication was lost with it. Only one of these fields "*OrderedNozzle*" and "*OrderedNozzles*" can be present in response depending on how the pump was authorized. If value of field "*OrderedType*" is "*FullTank*" then field "*OrderedDose*" is absent.

Example	{
	"Protocol": "jsonPTS",
	"Packets": [
	{
	"Id": 1,
	"Type": "PumpOfflineStatus",
	"Data": {
	"Pump": 1,
	"State":"Authorized",
	"NozzleUp":0,
	"Nozzle":0,
	"Request":"PumpAuthorize",
	"OrderedNozzle":1,
	"OrderedType":"Amount",
	"OrderedDose":25.00,
	"User":"admin",
	"LastDateTimeStart":"2023-07-05T13:09:28",
	"LastDateTime":"2023-07-05T13:10:37",
	"LastNozzle":1,
	"LastFuelGradeId":1,
	"LastFuelGradeName":"Fuel",
	"LastTransaction":55,
	"LastVolume":42.18,
	"LastPrice":1.23,
	"LastAmount":51.88,
	"LastTotalVolume":21309.62,
	"LastTotalAmount":43890.57,
	"LastFlowRate":27,
	"LastUser":"PTS"
	}
	}
	]
	}

116. PumpAuthorize request

Purpose	Sets preset and nozzle price, allows filling for specified nozzle
Data	"*Pump*" – logical number of the pump (integer, range from 1 to 120) "*Nozzle*" – nozzle number (integer, range from 1 to 6) "*Nozzles*" – array of nozzle numbers (integer, up to 6 items, range from 1 to 6) "*FuelGradeId*" – identifier of fuel grade (integer, range from 1 to 20) "*FuelGradeIds*" – array of fuel grade identifiers (integer, up to 10 items, range from 1 to 20) "*Type*" – preset type (string, up to 8 ASCII symbols), possible values: – "*Volume*": preset by product volume – "*Amount*": preset by money amount – "*FullTank*": preset with full tank, in some situations also meaning preset dose is to be entered from pump preset keyboard "*Dose*" - preset dose in volume units or in currency units (float, up to 3 digits after decimal point) "*Price*" – price (float, up to 3 digits after decimal point) "*Transaction*" – transaction number (integer, range from 1 to 65535) "*AutoCloseTransaction*" – flag to automatically close transaction once it finishes (Boolean, possible values: "*true*", "*false*") "*Tag*" – tag identifier used to authorize the pump (string, up to 32 hexadecimal symbols) "*RequestTagInformationOnServer*" – flag to verify the tag on a remote server before authorizing the pump (Boolean, possible values: "*true*", "*false*") "*AuthorizeWithoutTag*" – flag to authorize the pump without any tag (Boolean, possible values: "*true*", "*false*") "*NoTagVerification*" – flag to authorize the pump with a specified tag, but without its verification (Boolean, possible values: "*true*", "*false*") "*NotApplyUser*" – flag not to apply the present user for the transaction (Boolean, possible values: "*true*", "*false*")
Notes	1. One of fields "*Nozzle*", "*Nozzles*", "*FuelGradeId*", "*FuelGradeIds*" should be used in request for specification of the nozzle to authorize, in case if "*FuelGradeId*" or "*FuelGradeIds*" field is used – then the PTS-2 controller should have configuration of fuel grades and configuration of nozzles to these fuel grade codes (see SetFuelGradesConfiguration and SetPumpNozzlesConfiguration requests). If several of these fields are specified in the request same time – then only one of them is applied, priority is the following: "*Nozzle*", "*Nozzles*", "*FuelGradeId*", "*FuelGradeIds*". 2. In case if all fields "*Nozzle*", "*Nozzles*", "*FuelGradeId*", "*FuelGradeIds*" are absent in the request and if the PTS-2 controller has configuration of fuel grades and configuration of nozzles to these fuel grade codes (see SetFuelGradesConfiguration and SetPumpNozzlesConfiguration requests) – then the first nozzle taken up on pump after authorization request is automatically authorized by the PTS-2 controller, field "*Price*" is ignored in this case. 3. Field "*Dose*" is optional and may be omitted in case if value of field "*Type*" equals to "*FullTank*". 4. Field "*Price*" is optional and may be omitted, in this case price is taken from fuel grades configuration, in this case the PTS-2 controller should have configuration of fuel grades set with non-zero price for nozzle to be authorized (see

	SetFuelGradesConfiguration and SetPumpNozzlesConfiguration requests). 5. Field "*Transaction*" is optional and may be omitted, it is to be applied when transaction number should be forcedly set. 6. Field "*AutoCloseTransaction*" is optional and may be omitted, in this case its value is considered to be "*false*". 7. Field "*Tag*" is optional and may be omitted, in case if it is present – then it Is used to save tag identifier for the transaction when a tag reader is connected to external control system, for application the used tag identifier value should be present in tags configuration and set as valid (see SetTagsList request). 8. Field "*RequestTagInformationOnServer*" is optional and may be omitted, in this case its value is considered to be "*false*". This flag is used to request the tag information on a remote server before authorizing a pump, to make it work the controller must be configured for requesting tags information from a remote server using SetRemoteServerConfiguration. 9. Field "*AuthorizeWithoutTag*" is optional and may be omitted, in this case its value is considered to be "*false*". This flag is used when the controller is configured to verify each tag and there is a need to make a filling without any tag. 10. Field "*NoTagVerification*" is optional and may be omitted, in this case its value is considered to be "*false*". This flag is used when it is needed to specify the tag identifier in field "*Tag*", but this tag should not be verified by the controller (for example, if case if the readers are not configured in the controller or not connected to it). 11. Field "*NotApplyUser*" is optional and may be omitted, in this case its value is considered to be "*false*". This flag is used when it is needed not to apply the present user for the transaction. When its value is set to "*true*" then the applied user for this pump is "*PTS*" (system user of the PTS-2 controller). This option is useful when later during the filling or after its end (while the transaction is opened and pump status is PumpFillingStatus or PumpEndOfTransactionStatus) any management system can lock the pump using the PumpLock request, which will apply the user used by this management system for this transaction, so previously applied user "*PTS*" will be changed to a user used by this management system.
Permission required	Any permission among listed below: – *Control*
Response	PumpAuthorizeConfirmation response

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"PumpAuthorize",
	"Data":{
	"Pump":1,
	"Nozzle":2,
	"Type":"Volume",
	"Dose":2.75,
	"Price":3.25
	}
	}]

	}

117. PumpAuthorizeConfirmation response

Purpose	Confirms pumps successful authorization and its transaction number
Data	"*Pump*" – logical number of the pump (integer, range from 1 to 120) "*Transaction*" – transaction number (integer, range from 1 to 65535)

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"PumpAuthorizeConfirmation",
	"Data":{
	"Pump":1,
	"Transaction":31
	}
	}]
	}

118. PumpGetTransactionInformation request

Purpose	Gets transaction information by its number
Data	"*Pump*" – logical number of the pump (integer, range from 1 to 120) "*Transaction*" – transaction number (integer, range from 0 to 65535)
Notes	1. PumpTransactionInformation response is returned in case if the transaction is already finished and current user is either the owner of requested transaction or has *Reports* permission (set in SetUsersConfiguration request), otherwise error is returned. 2. In case if SD flash disk is not inserted or is not working and transaction is already finished then error response is returned. 3. If field "*Transaction*" is absent or equals to 0 – then information on the last transaction is returned. 4. Operation of this request is limited to 1000 last records to avoid long processing of the request.
Permission required	Any permission among listed below: – *Control* – *Reports*
Response	PumpTransactionInformation response

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"PumpGetTransactionInformation",
	"Data":{
	"Pump":1,
	"Transaction":15
	}
	}]
	}

119. PumpTransactionInformation response

Purpose	Returns transaction information by its number
Data	"*Pump*" – logical number of the pump (integer, range from 1 to 120) "*Transaction*" – transaction number (integer, range from 1 to 65535) "*State*" – state of transaction, possible values: – *"WaitingNozzleUpForAuthorization"* – pump is to be authorized on nozzle up, transaction has not started yet – *"Authorized"* – pump is authorized, transaction has not started yet – *"Filling"* – transaction is in process (filling is going) – *"EndOfTransaction"* – transaction is in process (filling has finished, transaction is still opened) – *"Finished"* – transaction is finished – *"Not found"* – transaction can not be found: either the transaction number is wrong or transaction did not take place (no fuel was dispensed) "*DateTimeStart*" – date and time of transaction start in format *YYYY-MM-DDThh:mm:ss* (string, 19 symbols), where: – *YYYY* – year (range from 2000 to 2099) – *MM* – month (range from 01 to 12) – *DD* – day (range from 01 to 31) – *T* – separator between date and time fields – *hh* – hours (range from 00 to 23) – *mm* – minutes (range from 00 to 59) – *ss* – seconds (range from 00 to 59) "*DateTime*" – date and time of transaction end in format *YYYY-MM-DDThh:mm:ss* (string, 19 symbols) "*Nozzle*" – nozzle number (integer, range from 1 to 6) "*FuelGradeId*" – identifier of fuel grade (integer, range from 1 to 20) "*FuelGradeName*" – name of fuel grade (string, up to 20 ASCII characters) "*Volume*" – transaction volume (float, up to 3 digits after decimal point) "*TCVolume*" - currently dispensed temperature-compensated volume (float, up to 3 digits after decimal point) "*Price*" – product price (float, up to 3 digits after decimal point) "*Amount*" – transaction money amount (float, up to 3 digits after decimal point) "*TotalVolume*" – volume total counter value (float, up to 3 digits after decimal point) "*TotalAmount*" – money amount total counter value (float, up to 3 digits after decimal point) "*Tag*" – (optional) tag identifier used to authorize the pump (string, up to 32 hexadecimal symbols) "*IsSuspended*" – flag showing if the filling is suspended (Boolean, possible values: "*true*", "*false*") "*IsOffline*" – flag showing that transaction was done in offline (manual) mode of the pump (Boolean, possible values: "*true*", "*false*") "*IsPaid*" – flag showing that transaction is paid (Boolean, possible values: "*true*", "*false*") "*OrderedNozzle*" – *ordered* nozzle number (integer, range from 1 to 6) "*OrderedNozzles*" – array of ordered nozzle numbers (integer, up to 6 items, range

	from 1 to 6) "*OrderedType*" – preset type done (string, up to 8 ASCII symbols), possible values: – "*Volume*": preset done by product volume – "*Amount*": preset done by money amount – "*FullTank*": preset done with full tank, in some situations also meaning preset dose to be entered from pump preset keyboard "*OrderedDose*" - preset dose in volume units or in currency units (float, up to 3 digits after decimal point) "*UserId*" – identifier of user, who authorized the transaction (integer, range from 1 to 30)
Note	1. Fields "*Pump*", "*Transaction*" and "*State*" are always present in response, presence of other fields depend on value of "State" field. 2. If field "*Tag*" has empty value – it means that no tag identifier was used to authorize the pump. 3. Fields "*FuelGradeId*" and "*FuelGradeName*" are optional, they are present in response only when PTS-2 controller has pump nozzles configured to specific fuel grades. 4. Fields "*DateTimeStart*" and "*DateTime*" are present in response only in case if such values are present in the controller for the requested transaction. 5. Fields "*IsOffline*" and "*IsPaid*" are optional, they present in response only in case if such values are present in the controller for the requested transaction. 6. Field "*IsSuspended*" is optional and is present only when the pump is in "*Filling*" state. 7. Fields "*OrderedNozzle*" and "*OrderedNozzles*" are optional, they are present only if the pump is in "*Authorized*" state. Only one of these fields is present in response depending on how the pump was authorized. 8. Fields "*OrderedType*" and "*OrderedDose*" are optional, they are present only if the pump is in "*Authorized*", "Filling" or "*EndOfTransaction*" state. If value of field "*OrderedType*" is "*FullTank*" then field "*OrderedDose*" is absent.

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"PumpTransactionInformation",
	"Data":{
	"Pump":1,
	"Transaction":15,
	"State":"Finished",
	"DateTimeStart":"2019-05-19T12:44:01",
	"DateTime":"2019-05-19T12:45:14",
	"Nozzle":1,
	"Volume":1.15,
	"TCVolume":1.14,
	"Price":10.00,
	"Amount":11.50,
	"TotalVolume":123456.789,

```
        "TotalAmount":987654.321,
        "Tag":"1231231231231231",
        "UserId":1
      }
   }]
}
```

120. PumpStop request

Purpose	Sends stop request to pump
Data	"*Pump*" – logical number of the pump (integer, range from 1 to 120)
Permission required	Any permission among listed below: – *Control*
Response	Confirmation response message (in case of successful execution) or error response message (in case of any problem)

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"PumpStop",
	"Data":{
	"Pump":1
	}
	}]
	}

121. PumpEmergencyStop request

Purpose	Sends stop request to pump and removes a user blocking the pump
Data	"*Pump*" – logical number of the pump (integer, range from 1 to 120)
Notes	This request is sent to pump regardless the user, who blocked the pump. This request should be only applied when some pump needs to be stopped and released from a user blocking it.
Permission required	Any permission among listed below: – *Control*
Response	Confirmation response message (in case of successful execution) or error response message (in case of any problem)

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"PumpEmergencyStop",
	"Data":{
	"Pump":1
	}
	}]
	}

122. PumpSuspend request

Purpose	Suspends filling by pump. This request is used to temporarily stop filling process. Can be used in situation when contact with a Vehicle Identification Device is lost. When sent – if will pause the filling, but the PTS-2 controller will keep sending PumpFillingStatus response.
Data	"*Pump*" – logical number of the pump (integer, range from 1 to 120)
Note	Request is possible only when pump is in filling status (while PTS responses with PumpFillingStatus response on PumpGetStatus request)
Permission required	Any permission among listed below: – *Control*
Response	Confirmation response message (in case of successful execution) or error response message (in case of any problem)

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"PumpSuspend",
	"Data":{
	"Pump":1
	}
	}]
	}

123. PumpResume request

Purpose	Resumes filling by pump. This request is used to resume filling process after it was paused by PumpSuspend request. Can be used in situation when contact with a Vehicle Identification Device is again found and filling process should be resumed.
Data	"*Pump*" – logical number of the pump (integer, range from 1 to 120)
Note	Request is possible only when pump is in filling status (while PTS responses with PumpFillingStatus response on PumpGetStatus request)
Permission required	Any permission among listed below: – *Control*
Response	Confirmation response message (in case of successful execution) or error response message (in case of any problem)

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"PumpResume",
	"Data":{
	"Pump":1
	}
	}]
	}

124. PumpCloseTransaction request

Purpose	Closes transaction. Number of a transaction must correspond to the number of transaction from PumpEndOfTransactionStatus response sent by PTS.
Data	"*Pump*" – logical number of the pump (integer, range from 1 to 120) "*Transaction*" – transaction number (integer, range from 1 to 65535)
Notes	1. Request is possible only when pump has finished filling and transaction is still not closed using PumpCloseTransaction request (while PTS responses with PumpEndOfTransactionStatus response on PumpGetStatus request). 2. Transaction can be closed by user, who initiated this transaction and whose Id is sent in PumpEndOfTransactionStatus response in field "*UserId*", however at emergency it may be closed by any user using PumpEmergencyStop request.
Permission required	Any permission among listed below: – *Control*
Response	Confirmation response message (in case of successful execution) or error response message (in case of any problem)

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"PumpCloseTransaction",
	"Data":{
	"Pump":1,
	"Transaction":15
	}
	}]
	}

125. PumpGetTotals request

Purpose	Gets total counters for specified nozzle of the pump
Data	"*Pump*" – logical number of the pump (integer, range from 1 to 120) "*Nozzle*" – nozzle number (integer, range from 1 to 6) "*FuelGradeId*" – identifier for fuel grade (integer, range from 1 to 20)
Notes	1. One of fields "*Nozzle*" and "*FuelGradeId*" can be used in request for specification of the nozzle, in case if "*FuelGradeId*" field is used – then the PTS-2 controller should have configuration of fuel grades set with one of the pump nozzles configured to the fuel grade code (see SetFuelGradesConfiguration request). In case if both fields "*Nozzle*" and "*FuelGradeId*" are used in request same time – then nozzle is selected by field "*Nozzle*". 2. After this request is received, the PTS-2 controller needs time to read the values of the totals from the pump. Due to it, the PTS-2 controller responds with a confirmation or error response to this request. Later, once the totals are received by the PTS-2 controller from the pump it will response with PumpTotals response on PumpGetStatus request. So, the management system after having requested the PumpGetTotals request should be polling PumpGetStatus request periodically until the PumpTotals response is received from the PTS-2 controller, which contains the values of the pump totals. 3. Only one nozzle on a pump can be polled for totals per time. The next nozzle should be polled after a response with totals is received from a previous nozzle. At this, different pumps can be polled in parallel at the same time.
Permission required	Any permission among listed below: – *Control*
Response	Confirmation response message (in case of successful execution) or error response message (in case of any problem)

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"PumpGetTotals",
	"Data":{
	"Pump":1,
	"Nozzle":1
	}
	}]
	}

126. PumpTotals response

Purpose	Returns total counters for specified nozzle of pump
Data	"*Pump*" – logical number of the pump (integer, range from 1 to 120) "*Nozzle*" – nozzle number (integer, range from 1 to 6) "*FuelGradeId*" – identifier of fuel grade (integer, range from 1 to 20) "*FuelGradeName*" – name of fuel grade (string, up to 20 ASCII characters) "*Volume*" – volume total counter value (float, up to 3 digits after decimal point) "*Amount*" – money amount total counter value (float, up to 3 digits after decimal point) "*Transaction*" – number of last transaction, performed on pump (integer, range from 1 to 99) "*User*" – name of user, who sent request (string, up to 10 ASCII symbols)
Note	Fields *"FuelGradeId"* and *"FuelGradeName"* are optional, they are present in response only when PTS-2 controller has pump nozzles configured to specific fuel grades.

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"PumpTotals",
	"Data":{
	"Pump":1,
	"Nozzle":1,
	"Volume":123456.789,
	"Amount":987654.321,
	"Transaction":37,
	"User":"admin"
	}
	}]
	}

127. PumpGetLastSavedTotals request

Purpose	Gets last saved total counters for the pump from the controller memory
Data	"*Pump*" – logical number of the pump (integer, range from 1 to 120)
Notes	This request works only if the controller is configured to read pump totals automatically. When this request is called – then the totals are not read from the pump, but the values of totals from the memory of the controller are returned. The values of totals are automatically updated in the memory of the controller after each pump sale, after restoring communication with the pump and on the startup of the controller.
Permission required	Any permission among listed below: – *Control*
Response	Confirmation response message (in case of successful execution) or error response message (in case of any problem)

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"PumpGetLastSavedTotals",
	"Data":{
	"Pump":1
	}
	}]
	}

128. PumpLastSavedTotals response

Purpose	Returns last saved total counters for the pump from the controller memory
Data	"*Pump*" – logical number of the pump (integer, range from 1 to 120) "*Nozzles*" – array of objects for each of the pump nozzles, elements of each object: – "*Nozzle*" – identifier of nozzle (integer, range from 1 to 6) – "*FuelGradeId*" – identifier of fuel grade (integer, range from 1 to 20) – "*FuelGradeName*" – name of fuel grade (string, up to 20 ASCII characters) – "*Volume*" – volume total counter value (float, up to 3 digits after decimal point) – "*Amount*" – money amount total counter value (float, up to 3 digits after decimal point)
Note	Fields *"FuelGradeId"* and *"FuelGradeName"* are optional, they are present in response only when PTS-2 controller has pump nozzles configured to specific fuel grades.

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"PumpLastSavedTotals",
	"Data":{
	"Pump":1,
	"Nozzles":[{
	"Nozzle":1,
	"FuelGradeId":1,
	"FuelGradeName":"Petrol",
	"Volume":123456.789,
	"Amount":987654.321
	},{
	"Nozzle":2,
	"FuelGradeId":2,
	"FuelGradeName":"Diesel",
	"Volume":345678.901,
	"Amount":123123123.321
	}
	}
	}]
	}

129. PumpGetPrices request

Purpose	Gets prices of pump
Data	"*Pump*" – logical number of the pump (integer, range from 1 to 120)
Note	Once the prices are received by the PTS-2 controller it will response with PumpPrices response on PumpGetStatus request.
Permission required	Any permission among listed below: – *Control*
Response	Confirmation response message (in case of successful execution) or error response message (in case of any problem)

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"PumpGetPrices",
	"Data":{
	"Pump":1
	}
	}]
	}

130. PumpSetPrices request

Purpose	Sets prices for pump
Data	"*Pump*" – logical number of the pump (integer, range from 1 to 120) "*Prices*" – array of prices for pump nozzles (float, up to 3 digits after decimal point)
Note	Once the prices are received by the PTS-2 controller from pump it will response with PumpPrices response on PumpGetStatus request.
Permission required	Any permission among listed below: – *Control*
Response	Confirmation response message (in case of successful execution) or error response message (in case of any problem)

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"PumpSetPrices",
	"Data":{
	"Pump":1,
	"Prices":[1.25,1.69,1.33,1.44,1.17,2]
	}
	}]
	}

131. PumpPrices response

Purpose	Returns prices for pump
Data	"*Pump*" – logical number of the pump (integer, range from 1 to 120) "*Prices*" – array of prices for pump nozzles (float, up to 3 digits after decimal point) "*User*" – name of user, who sent request (string, up to 10 ASCII symbols)

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"PumpPrices",
	"Data":{
	"Pump":1,
	"Prices":[1.25,1.69,1.33,1.44,1.17,2],
	"User":"admin"
	}
	}]
	}

132. PumpGetDisplayData request

Purpose	Gets display data of pump
Data	"*Pump*" – logical number of the pump (integer, range from 1 to 120)
Note	Once the display data is received by the PTS-2 controller it will response with PumpDisplayData response on PumpGetStatus request.
Permission required	Any permission among listed below: – *Control*
Response	Confirmation response message (in case of successful execution) or error response message (in case of any problem)

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"PumpGetDisplayData",
	"Data":{
	"Pump":1
	}
	}]
	}

133. PumpDisplayData response

Purpose	Returns display data for pump
Data	"*Pump*" – logical number of the pump (integer, range from 1 to 120) "*LastNozzle*" – last nozzle number (integer, range from 0 to 6) "*LastFuelGradeId*" – identifier of fuel grade (integer, range from 1 to 20) "*LastFuelGradeName*" – name of fuel grade (string, up to 20 ASCII characters) "*LastTransaction*" – number of last transaction performed on pump (integer, range from 0 to 65535) "*Volume*" – displayed volume (float, up to 3 digits after decimal point) "*Amount*" – displayed money amount (float, up to 3 digits after decimal point) "*User*" – name of user, who sent request (string, up to 10 ASCII symbols)
Note	1. Fields "*LastNozzle*" and "*LastTransaction*" may have value 0 in case if it not possible to get this information from pump. 2. Fields "*LastFuelGradeId"* and "*LastFuelGradeName"* are optional, they are present in response only when PTS-2 controller has pump nozzles configured to specific fuel grades.

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"PumpDisplayData",
	"Data":{
	"Pump":1,
	"LastNozzle":1,
	"LastTransaction":37,
	"Volume":1.55,
	"Amount":12.34,
	"User":"admin"
	}
	}]
	}

134. PumpGetTag request

Purpose	Gets value of tag ID from pump nozzle reader
Data	"*Pump*" – logical number of the pump (integer, range from 1 to 120) "*Nozzle*" – nozzle number (integer, range from 0 to 6)
Note	1. Once the tag is received by the PTS-2 controller it will response with PumpTag response on PumpGetStatus request. 2. If field "*Nozzle*" has value 0 – then it means to read the tag for any nozzle on the pump.
Permission required	Any permission among listed below: – *Control*
Response	Confirmation response message (in case of successful execution) or error response message (in case of any problem)

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"PumpGetTag",
	"Data":{
	"Pump":1,
	"Nozzle":2
	}
	}]
	}

135. PumpTag response

Purpose	Returns value of tag ID from pump nozzle reader
Data	"*Pump*" – logical number of the pump (integer, range from 1 to 120) "*Nozzle*" – nozzle number (integer, range from 0 to 6) "*FuelGradeId*" – identifier of fuel grade (integer, range from 1 to 20) "*FuelGradeName*" – name of fuel grade (string, up to 20 ASCII characters) "*Tag*" – value of tag ID (string, up to 32 hexadecimal symbols) "*User*" – name of user, who sent request (string, up to 10 ASCII symbols)
Notes	1. If field "*Nozzle*" has empty value – then it means to read the tag for any nozzle on the pump. 2. If field "*Tag*" has empty value – it means that no tag identifier was read. 3. Fields *"FuelGradeId"* and *"FuelGradeName"* are optional, they are present in response only when PTS-2 controller has pump nozzles configured to specific fuel grades.

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"PumpTag",
	"Data":{
	"Pump":1,
	"Nozzle":2,
	"Tag":"1234567890ABCDEF",
	"User":"admin"
	}
	}]
	}

136. PumpGetAdditionalMeasurements request

Purpose	Gets pump additional measurements (temperature, pressure, mass, temperature-compensated volume, others)
Data	"*Pump*" – logical number of the pump (integer, range from 1 to 120)
Note	This request returns meaningful measurements data only for brands of pumps, which provide it
Permission required	Any permission among listed below: – *Control* – *Monitoring*
Response	PumpAdditionalMeasurements response

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"PumpGetAdditionalMeasurements",
	"Data":{
	"Pump":1
	}
	}]
	}

137. PumpAdditionalMeasurements response

Purpose	Returns pump additional measurements (temperature, pressure, mass, temperature-compensated volume, others)
Data	"*Pump*" – logical number of the pump (integer, range from 1 to 120) "*CurrentTemperature*" – value current product temperature in degrees Celcius (float, up to 1 digit after decimal point) "*LastTransactionTemperature*" – value of product temperature at last transaction in degrees Celcius (float, up to 1 digit after decimal point) "*LastTransactionTCVolume*" – value of product temperature compensated volume at last transaction in liters (integer) "*CurrentDensity*" – value of current product density in kg/m^3 units (float, up to 1 digit after decimal point) "*LastTransactionDensity*" – value of product density at last transaction in kg/m^3 units (float, up to 1 digit after decimal point) "*LastTransactionTCDensity*" – value of product temperature compensated density at last transaction in kg/m^3 units (float, up to 1 digit after decimal point) "*LastTransactionMass*" – value of product mass at last transaction in kg (float, up to 1 digit after decimal point) "*CurrentPressure*" – value of current product pressure in psi (integer)
Notes	This response contains only values provided by the pump, if the pump does not provide any additional measurement values – then none are returned.

Example	{
	"Protocol": "jsonPTS",
	"Packets": [
	{
	"Id": 1,
	"Type": "PumpAdditionalMeasurements",
	"Data": {
	"Pump": 1,
	"LastTransactionTemperature": 12.3
	}
	}
	]
	}

138. PumpLock request

Purpose	Locks the pump by the user.
Data	"*Pump*" – logical number of the pump (integer, range from 1 to 120)
Permission required	Any permission among listed below: – *Control*
Response	Confirmation response message (in case of successful execution) or error response message (in case of any problem)
Notes	1. This request can be useful when there are several management systems connected to the same PTS-2 controller and one of them wants to lock control over the pump, so that no other management system could control the pump during this process, for example to lock the pump while the customer is providing prepayment before authorizing the pump. In this case to unlock the previously locked pump by this request use the PumpUnlock request. This operation is possible when the pump status is PumpIdleStatus and the pump is not locked by any other management system. 2. This request can also be used to override the user used for the pump during the filling or after its end (while the transaction is opened and pump status is PumpFillingStatus or PumpEndOfTransactionStatus) in case if the presently applied user for this pump is "*PTS*" (system user of the PTS-2 controller). Any management system can lock the pump using the PumpLock request, which will apply the user used by this management system for this transaction, so previously applied user "*PTS*" will be changed to a user used by this management system. This helps to solve the situation when one management system has to authorize the pump and another management system has to close the transaction.

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"PumpLock",
	"Data":{
	"Pump":1
	}
	}]
	}

139. PumpUnlock request

Purpose	Unlocks the pump, which was previously locked using PumpLock request.
Data	"*Pump*" – logical number of the pump (integer, range from 1 to 120)
Permission required	Any permission among listed below: – *Control*
Response	Confirmation response message (in case of successful execution) or error response message (in case of any problem)
Note	1. This request will successfully execute when the present user locking the pump it the user, which calls this request. 2. In case if the transaction is opened and pump status is PumpFillingStatus or PumpEndOfTransactionStatus and also in case if the pump status is PumpIdleStatus and there is any request being executed– then after this request execution the user will become "*PTS*" (system user of the PTS-2 controller).

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"PumpUnlock",
	"Data":{
	"Pump":1
	}
	}]
	}

140. PumpSetLights request

Purpose	Sets lights for pump
Data	"*Pump*" – logical number of the pump (integer, range from 1 to 120) "*State*" – state of lights (string, up to 3 ASCII symbols), possible values: – "*On*": turns on lights – "*Off*": turns off lights
Permission required	Any permission among listed below: – *Control*
Response	Confirmation response message (in case of successful execution) or error response message (in case of any problem)

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"PumpSetLights",
	"Data":{
	"Pump":1,
	"State":"On"
	}
	}]
	}

141. PumpGetAutomaticOperation request

Purpose	Shortcut to get pump parameters for automatic pump authorization on nozzle up and also automatic closing of transactions.
Data	"*Pump*" – logical number of the pump (integer, range from 1 to 120)
Permission required	Any permission among listed below: – *Configuration* – *Control* – *Monitoring*
Response	PumpAutomaticOperation response

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"PumpGetAutomaticOperation",
	"Data":{
	"Pump":1
	}
	}]
	}

142. PumpSetAutomaticOperation request

Purpose	Shortcut to set/unset pump parameters for automatic pump authorization on nozzle up and also automatic closing of transactions.
Data	"*Pump*" – logical number of the pump (integer, range from 1 to 120) "*State*" – state of automatic pump authorization (string, up to 3 ASCII symbols), possible values: – "*On*": sets pump parameters for automatic pump authorization on nozzle up and also automatic closing of transactions – "*Off*": unsets pump parameters for automatic pump authorization on nozzle up and also automatic closing of transactions "*AutoCloseTransaction*" – specifies automatic closing of transactions, overrides the value in "*State*" field (string, up to 3 ASCII symbols), possible values: – "*On*": sets automatic closing of transactions – "*Off*": unsets automatic closing of transactions
Permission required	Any permission among listed below: – *Configuration* – *Control*
Response	Confirmation response message (in case of successful execution) or error response message (in case of any problem)
Notes	Field "*AutoCloseTransaction*" is optional and is used in for the cases when it is important to have a state of automatic closing of transactions to be different from a state of automatic pump authorization on nozzle up. When this parameter is absent – then its value is equal to value of "*State*" field.

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"PumpSetAutomaticOperation",
	"Data":{
	"Pump":1,
	"State":"On"
	}
	}]
	}

143. PumpAutomaticOperation response

Purpose	Returns state of automatic pump authorization on nozzle up and also automatic closing of transactions.
Data	"*Pump*" – logical number of the pump (integer, range from 1 to 120) "*State*" – state of automatic pump authorization (string, up to 3 ASCII symbols), possible values: – "*On*": sets pump parameters for automatic pump authorization on nozzle up and also automatic closing of transactions – "*Off*": unsets pump parameters for automatic pump authorization on nozzle up and also automatic closing of transactions "*AutoCloseTransaction*" – specifies automatic closing of transactions, overrides the value in "*State*" field (string, up to 3 ASCII symbols), possible values: – "*On*": sets automatic closing of transactions – "*Off*": unsets automatic closing of transactions
Notes	Field "*AutoCloseTransaction*" is optional and is present only when state of automatic closing of transactions differs from state of automatic pump authorization on nozzle up. When this parameter is absent – then its value is equal to value of "*State*" field.

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"PumpAutomaticOperation",
	"Data":{
	"Pump":1,
	"State":"On"
	}
	}]
	}

PROBES REQUESTS AND RESPONSES

This part describes requests and responses used for provision of monitoring over probes/tanks.

144. ProbeGetMeasurements request

Purpose	Gets probes status, list of measured values and measured values
Data	"*Probe*" – logical number of the probe (integer, range from 1 to 20)
Permission required	Any permission among listed below: – *Control* – *Monitoring*
Response	ProbeMeasurements response

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"ProbeGetMeasurements",
	"Data":{
	"Probe":1
	}
	}]
	}

145. ProbeMeasurements response

Purpose	Returns probe status and measured values
Data	"*Probe*" – logical number of the probe (integer, range from 1 to 20) "*DateTime*" – date and time of last measurement data received from probe in format *YYYY-MM-DDThh:mm:ss* (string, 19 symbols), where: – *YYYY* – year (range from 2000 to 2099) – *MM* – month (range from 01 to 12) – *DD* – day (range from 01 to 31) – *T* – separator between date and time fields – *hh* – hours (range from 00 to 23) – *mm* – minutes (range from 00 to 59) – *ss* – seconds (range from 00 to 59) "*FuelGradeId*" – identifier of fuel grade (integer, range from 1 to 20) "*FuelGradeName*" – name of fuel grade (string, up to 20 ASCII characters) "*Status*" – status of probe (string, up to 7 ASCII symbols), possible values: – "*OK*": in case of correct operation – "*Error*": in case of incorrect operation – "*Offline*": in case if there are no responses from probe "*Alarms*" – array of strings indicating probe alarms for tank, possible values: – "*CriticalHighProduct*": if product level is higher than maximum critical – "*HighProduct*": if product level is higher than maximum allowed – "*LowProduct*": if product level is lower than minimum allowed – "*CriticalLowProduct*": if product level is lower than minimum critical – "*HighWater*": in case if water level is higher than maximum allowed – "*TankLeakage*": in case if tank leakage is detected "*ProductHeight*" – value of product height in mm (float, up to 1 digit after decimal point) "*WaterHeight*" – value of water height in mm (float, up to 1 digit after decimal point) "*Temperature*" – value product temperature in degrees Celcius (float, up to 1 digit after decimal point) "*ProductVolume*" – value of product volume in liters (integer) "*ProductVolumeFloat*" – value of product volume in liters with decimals (float, up to 2 digits after decimal point) "*WaterVolume*" – value of water volume in liters (integer) "*WaterVolumeFloat*" – value of water volume in liters with decimals (float, up to 2 digits after decimal point) "*ProductUllage*" – value of product ullage in liters (integer) "*ProductUllageFloat*" – value of product ullage in liters with decimals (float, up to 2 digits after decimal point) "*ProductTCVolume*" – value of product temperature compensated volume in liters (integer) "*ProductTCVolumeFloat*" – value of product temperature compensated volume in liters with decimals (float, up to 2 digits after decimal point) "*ProductDensity*" – value of product density in kg/m^3 units (float, up to 1 digit after decimal point) "*ProductMass*" – value of product mass in kg (float, up to 1 digit after decimal point)

	"*TankFillingPercentage*" – value of tank filling percentage (integer) "*InTankDeliveryDetected*" – flag showing if there is an in-tank delivery process detected (Boolean, possible values: "*true*", "*false*") "*LastInTankDeliveryStart*" – object describing start of last in-tank delivery, contains following elements: – "*DateTime*" – date and time of last in-tank delivery start in format *YYYY-MM-DDThh:mm:ss* (string, 19 symbols), where: • *YYYY* – year (range from 2000 to 2099) • *MM* – month (range from 01 to 12) • *DD* – day (range from 01 to 31) • *T* – separator between date and time fields • *hh* – hours (range from 00 to 23) • *mm* – minutes (range from 00 to 59) • *ss* – seconds (range from 00 to 59) – "*ProductHeight*" – value of product height in mm (float, up to 1 digit after decimal point) – "*WaterHeight*" – value of water height in mm (float, up to 1 digit after decimal point) – "*Temperature*" – value of temperature in degrees Celcius (float, up to 1 digit after decimal point) – "*ProductVolume*" – value of product volume in liters (integer) – "*ProductTCVolume*" – value of product temperature compensated volume in liters (integer) – "*ProductDensity*" – value of product density in kg/m^3 units (float, up to 1 digit after decimal point) – "*ProductMass*" – value of product mass in kg (float, up to 1 digit after decimal point) "*LastInTankDeliveryEnd*" – object describing end of last in-tank delivery, contains following elements: – "*DateTime*" – date and time of last in-tank delivery end in format *YYYY-MM-DDThh:mm:ss* (string, 19 symbols), where: • *YYYY* – year (range from 2000 to 2099) • *MM* – month (range from 01 to 12) • *DD* – day (range from 01 to 31) • *T* – separator between date and time fields • *hh* – hours (range from 00 to 23) • *mm* – minutes (range from 00 to 59) • *ss* – seconds (range from 00 to 59) – "*ProductHeight*" – value of product height in mm (float, up to 1 digit after decimal point) – "*WaterHeight*" – value of water height in mm (float, up to 1 digit after decimal point) – "*Temperature*" – value of temperature in degrees Celcius (float, up to 1 digit after decimal point) – "*ProductVolume*" – value of product volume in liters (integer) – "*ProductTCVolume*" – value of product temperature compensated volume in liters (integer)

	– "*ProductDensity*" – value of product density in kg/m^3 units (float, up to 1 digit after decimal point) – "*ProductMass*" – value of product mass in kg (float, up to 1 digit after decimal point) "*LastInTankDelivery*" – object describing absolute values of last in-tank delivery, contains following elements: – "*ProductHeight*" – value of product height in mm (float, up to 1 digit after decimal point) – "*WaterHeight*" – value of water height in mm (float, up to 1 digit after decimal point) – "*Temperature*" – value of temperature in degrees Celcius (float, up to 1 digit after decimal point) – "*ProductVolume*" – value of product volume in liters (nteger) – "*ProductTCVolume*" – value of product temperature compensated volume in liters (integer) – "*ProductDensity*" – value of product density in kg/m^3 units (float, up to 1 digit after decimal point) – "*ProductMass*" – value of product mass in kg (float, up to 1 digit after decimal point) – "*PumpsDispensedVolume*" – value of product volume dispensed through pumps during delivery in liters (float, up to 3 digits after decimal point)
Notes	1. This response contains only values measured by probe and calculated by the controller, so some of the values may not be present in response. 2. In case if the probe gives out in-tank delivery information then this response can be added with "*LastInTankDeliveryStart*", "*LastInTankDeliveryEnd*" and "*LastInTankDelivery*" objects, containing in-tank delivery information. Presence of these objects and their content depends on which values are informed by ATG console or probe. 3. Array "*Alarms*" is optional and is present in response only in case if there are alarms present. 4. Field "*PumpsDispensedVolume*" is optional and is present in response only in case if: a). pump nozzles are linked to the tank and to fuel grade and b). there is a registered volume filled from tank through pumps during last in-tank delivery. 5. Field "*TankFillingPercentage*" is optional and is calculated by the controller the following way: – as ratio of values of (filled product volume + filled water volume) and (filled product volume + filled water volume + ullage), if these values are present – as ratio of values of filled product height and full tank height, if these values are present 6. Fields *"FuelGradeId"* and *"FuelGradeName"* are optional, they are present in response only when PTS-2 controller has tank configured to specific fuel grade.

Example	{
	"Protocol": "jsonPTS",
	"Packets": [
	{
	"Id": 1,
	"Type": "ProbeMeasurements",
	"Data": {
	"Probe": 1,

```
                    "DateTime": "2019-03-09T15:32:15",
                    "Status": "OK",
                    "ProductHeight": 2534.1,
                    "WaterHeight": 123.1,
                    "Temperature": 21.3,
                    "ProductVolume": 21451,
                    "ProductVolumeFloat": 21451.34,
                    "WaterVolume": 234,
                    "WaterVolumeFloat": 234.11,
                    "ProductUllage": 9276,
                    "ProductUllageFloat": 9276.17,
                    "ProductTCVolume": 9202,
                    "ProductTCVolumeFloat": 9202.91,
                    "InTankDeliveryDetected": true,
                    "LastInTankDeliveryStart": {
                        "DateTime": "2019-03-09T15:30:25",
                        "ProductHeight": 2534.1,
                        "WaterHeight": 123.1,
                        "Temperature": 21.3,
                        "ProductVolume": 2005,
                        "ProductTCVolume": 2015
                    },
                    "LastInTankDeliveryEnd": {
                        "DateTime": "2019-03-09T15:32:15",
                        "ProductHeight": 2725.1,
                        "WaterHeight": 130.4,
                        "Temperature": 21.6,
                        "ProductVolume": 2514,
                        "ProductTCVolume": 2601
                    },
                    "LastInTankDelivery": {
                        "ProductDensity": 758.9,
                        "ProductMass": 386.3,
                        "PumpsDispensedVolume": 25.3
                    }
                }
            }
        ]
}
```

146. ProbeGetTankCalibrationChartTotalRecordsNumber request

Purpose	Gets a total number of records in tank's calibration chart
Data	"*Probe*" – logical number of the probe (integer, range from 1 to 20)
Permission required	Any permission among listed below: – *Configuration* – *Control* – *Reports*
Response	ProbeTankCalibrationChartTotalRecordsNumber response

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"ProbeGetTankCalibrationChartTotalRecords
	Number",
	"Data":{
	"Probe":1
	}
	}]
	}

147. ProbeTankCalibrationChartTotalRecordsNumber response

Purpose	Returns a total number of records in tank's calibration chart
Data	"*Probe*" – logical number of the probe (integer, range from 1 to 20) "*TotalNumber*" – total number of records (integer, range from 0 and higher)

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"ProbeTankCalibrationChartTotalRecordsNumber",
	"Data":{
	"Probe":1,
	"TotalNumber":123
	}
	}]
	}

148. ProbeGetTankCalibrationChartRecordsList request

Purpose	Gets a list of records of the tank's calibration chart
Data	"*Probe*" – logical number of the probe (integer, range from 1 to 20) "*StartNumber*" – starting number of record in list to read (integer, range from 1 and higher) "*TotalNumber*" – total number of records to read (integer, range from 1 and till 100)
Permission required	Any permission among listed below: – *Configuration* – *Control* – *Reports*
Response	ProbeTankCalibrationChartRecordsList response
Note	1. Field "*StartNumber*" is optional, if it is not present in request – then its value is automatically set to 1. 2. Field "*TotalNumber*" is optional, if it is not present in request – then its value is automatically limited with 100.

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"ProbeGetTankCalibrationChartRecordsList",
	"Data":{
	"Probe":1,
	"StartNumber":1,
	"TotalNumber":100
	}
	}]
	}

149. ProbeTankCalibrationChartRecordsList response

Purpose	Returns a list of records of the tank's calibration chart
Data	"*Probe*" – logical number of the probe (integer, range from 1 to 20) "*Records*" – array of objects for configuration of pumps, elements of each object: – "*Height*" – tank height in mm (integer) – "Volume" – tank volume in volume units (integer)

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"ProbeTankCalibrationChartRecordsList",
	"Data":[{
	"Probe":1,
	"Records":[{
	"Height":10,
	"Volume":123
	},{
	"Height":20,
	"Volume":456
	},{
	"Height":30,
	"Volume":789
	}]
	}]
	}]
	}

150. ProbeSetTankCalibrationChartRecordsList request

Purpose	Sets a list of records of the tank's calibration chart
Data	"*Probe*" – logical number of the probe (integer, range from 1 to 20) "*Records*" – array of objects for configuration of pumps, elements of each object: – "*Height*" – tank height in mm (integer) – "Volume" – tank volume in volume units (integer)
Permission required	Any permission among listed below: – *Configuration*
Response	Confirmation response message (in case of successful execution) or error response message (in case of any problem)
Note	1. This request clears all previously saved records in configuration. 2. This request has a limitation of maximum 100 records to be saved. To save more tags additionally use ProbeAddTankCalibrationChartRecordsList request.

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"ProbeSetTankCalibrationChartRecordsList",
	"Data":[{
	"Probe":1,
	"Records":[{
	"Height":10,
	"Volume":123
	},{
	"Height":20,
	"Volume":456
	},{
	"Height":30,
	"Volume":789
	}]
	}]
	}]
	}

151. ProbeAddTankCalibrationChartRecordsList request

Purpose	Adds a list of records to the tank's calibration chart
Data	"*Probe*" – logical number of the probe (integer, range from 1 to 20) "*Records*" – array of objects for configuration of pumps, elements of each object: – "*Height*" – tank height in mm (integer) – "Volume" – tank volume in volume units (integer)
Permission required	Any permission among listed below: – *Configuration*
Response	Confirmation response message (in case of successful execution) or error response message (in case of any problem)
Note	1. This request has a limitation of maximum 100 records to be saved at a single request. 2. In case if any of the records is already present in configuration – then it is overwritten.

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"ProbeAddTankCalibrationChartRecordsList",
	"Data":[{
	"Probe":1,
	"Records":[{
	"Height":10,
	"Volume":123
	},{
	"Height":20,
	"Volume":456
	},{
	"Height":30,
	"Volume":789
	}]
	}]
	}]
	}

152. ProbeAddTankCalibrationChartRecordToList request

Purpose	Adds a record to the tank's calibration chart
Data	"*Probe*" – logical number of the probe (integer, range from 1 to 20) "*Height*" – tank height in mm (integer) "Volume" – tank volume in volume units (integer)
Permission required	Any permission among listed below: – *Configuration*
Response	Confirmation response message (in case of successful execution) or error response message (in case of any problem)

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"ProbeAddTankCalibrationChartRecordToList",
	"Data":[{
	"Probe":1,
	"Height":10,
	"Volume":123
	}]
	}]
	}

153. ProbeEditTankCalibrationChartRecordInList request

Purpose	Edits a record in the tank's calibration chart
Data	"*Probe*" – logical number of the probe (integer, range from 1 to 20) "*Height*" – tank height in mm (integer) "Volume" – tank volume in volume units (integer)
Permission required	Any permission among listed below: – *Configuration*
Response	Confirmation response message (in case of successful execution) or error response message (in case of any problem)

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"ProbeEditTankCalibrationChartRecordInList",
	"Data":[{
	"Probe":1,
	"Height":10,
	"Volume":123
	}]
	}]
	}

154. ProbeDeleteTankCalibrationChartRecordFromList request

Purpose	Deletes a record from the tank's calibration chart
Data	"*Probe*" – logical number of the probe (integer, range from 1 to 20) "*Height*" – tank height in mm (integer)
Permission required	Any permission among listed below: – *Configuration*
Response	Confirmation response message (in case of successful execution) or error response message (in case of any problem)

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"ProbeDeleteTankCalibrationChartRecord
	FromList",
	"Data":[{
	"Probe":1,
	"Height":10
	}]
	}]
	}

155. ProbeGetTankVolumeForHeight request

Purpose	Gets tank volume for certain height based on tank's calibration chart
Data	"*Probe*" – logical number of the probe (integer, range from 1 to 20) "*Height*" – value of height in mm (integer)
Permission required	Any permission among listed below: – *Configuration* – *Control* – *Monitoring*
Response	ProbeTankVolumeForHeight response
Note	Tank should have a valid calibration chart configured

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"ProbeGetTankVolumeForHeight",
	"Data":{
	"Probe":1,
	"Height":120
	}
	}]
	}

156. ProbeTankVolumeForHeight response

Purpose	Returns tank volume for certain height based on tank's calibration chart
Data	"*Probe*" – logical number of the probe (integer, range from 1 to 20) "*Height*" – value of height in mm (integer) "*Volume*" – value of volume in liters (integer)

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"ProbeTankVolumeForHeight",
	"Data":{
	"Probe":1,
	"Height": 120,
	"Volume":2534
	}
	}]
	}

157. ProbeGenerateTankAutomaticCalibrationChart request

Purpose	Generates a new tank automatic calibration chart based on tank interval volume table.
Data	"*Probe*" – logical number of the probe (integer, range from 1 to 20)
Permission required	Any permission among listed below: – *Configuration*
Response	Confirmation response message (in case of successful execution) or error response message (in case of any problem)

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"ProbeGenerateTankAutomaticCalibrationChart",
	"Data":{
	"Probe":1
	}
	}]
	}

158. ProbeGetTankIntervalVolumeChartTotalRecordsNumber request

Purpose	Gets a total number of records of the tank's interval volume calculated during automatic calibration process
Data	"*Probe*" – logical number of the probe (integer, range from 1 to 20)
Permission required	Any permission among listed below: – *Configuration* – *Control* – *Reports*
Response	ProbeTankIntervalVolumeChartTotalRecordsNumber response

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"ProbeGetTankIntervalVolumeChart
	TotalRecordsNumber",
	"Data":{
	"Probe":1
	}
	}]
	}

159. ProbeTankIntervalVolumeChartTotalRecordsNumber response

Purpose	Returns a total number of records of the tank's interval volume calculated during automatic calibration process
Data	"*Probe*" – logical number of the probe (integer, range from 1 to 20) "*TotalNumber*" – total number of records (integer, range from 0 and higher)

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"ProbeTankintervalVolumeChart
	TotalRecordsNumber",
	"Data":{
	"Probe":1,
	"TotalNumber":123
	}
	}]
	}

160. ProbeGetTankIntervalVolumeChartRecordsList request

Purpose	Gets a list of records of the tank's interval volume calculated during automatic calibration process
Data	"*Probe*" – logical number of the probe (integer, range from 1 to 20) "*StartNumber*" – starting number of record in list to read (integer, range from 1 and higher) "*TotalNumber*" – total number of records to read (integer, range from 1 and till 100)
Permission required	Any permission among listed below: – *Configuration* – *Control* – *Reports*
Response	ProbeTankIntervalVolumeChartRecordsList response
Note	1. Field "*StartNumber*" is optional, if it is not present in request – then its value is automatically set to 1. 2. Field "*TotalNumber*" is optional, if it is not present in request – then its value is automatically limited with 100.

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"ProbeGetTankIntervalVolumeChartRecordsList",
	"Data":{
	"Probe":1,
	"StartNumber":1,
	"TotalNumber":100
	}
	}]
	}

161. ProbeTankIntervalVolumeChartRecordsList response

Purpose	Returns a list of records of the tank's interval volume calculated during automatic calibration process
Data	"*Probe*" – logical number of the probe (integer, range from 1 to 20) "*Records*" – array of objects for configuration of pumps, elements of each object: – "*Height*" – tank interval start height in mm (integer) – "*Volume*" – tank interval volume in volume units (integer) – "*PassesNumber*" – number of passes for tank interval volume (integer)

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"ProbeTankIntervalVolumeChartRecordsList",
	"Data":[{
	"Probe":1,
	"Records":[{
	"Height":10,
	"Volume":123,
	"PassesNumber":12
	},{
	"Height":15,
	"Volume":456,
	"PassesNumber":20
	},{
	"Height":20,
	"Volume":789,
	"PassesNumber":17
	}]
	}]
	}]
	}

162. ProbeGetTankAutomaticCalibrationChartTotalRecordsNumber request

Purpose	Gets a total number of records in tank's automatic calibration chart
Data	"*Probe*" – logical number of the probe (integer, range from 1 to 20)
Permission required	Any permission among listed below: – *Configuration* – *Control* – *Reports*
Response	ProbeTankAutomaticCalibrationChartTotalRecordsNumber response

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"ProbeGetTankAutomaticCalibrationChartTotal
	RecordsNumber",
	"Data":{
	"Probe":1
	}
	}]
	}

163. ProbeTankAutomaticCalibrationChartTotalRecordsNumber response

Purpose	Returns a total number of records in tank's automatic calibration chart
Data	"*Probe*" – logical number of the probe (integer, range from 1 to 20) "*TotalNumber*" – total number of records (integer, range from 0 and higher)

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"ProbeTankAutomaticCalibrationChartTotal
	RecordsNumber",
	"Data":{
	"Probe":1,
	"TotalNumber":123
	}
	}]
	}

164. ProbeGetTankAutomaticCalibrationChartRecordsList request

Purpose	Gets a list of records of the tank's automatic calibration chart
Data	"*Probe*" – logical number of the probe (integer, range from 1 to 20) "*StartNumber*" – starting number of record in list to read (integer, range from 1 and higher) "*TotalNumber*" – total number of records to read (integer, range from 1 and till 100)
Permission required	Any permission among listed below: – *Configuration* – *Control* – *Reports*
Response	ProbeTankAutomaticCalibrationChartRecordsList response
Note	1. Field "*StartNumber*" is optional, if it is not present in request – then its value is automatically set to 1. 2. Field "*TotalNumber*" is optional, if it is not present in request – then its value is automatically limited with 100.

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"ProbeGetTankAutomaticCalibrationChart
	RecordsList",
	"Data":{
	"Probe":1,
	"StartNumber":1,
	"TotalNumber":100
	}
	}]
	}

165. ProbeTankAutomaticCalibrationChartRecordsList response

Purpose	Returns a list of records of the tank's automatic calibration chart
Data	"*Probe*" – logical number of the probe (integer, range from 1 to 20) "*Records*" – array of objects for configuration of pumps, elements of each object: – "*Height*" – tank height in mm (integer) – "*Volume*" – tank volume in volume units (integer)

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"ProbeTankAutomaticCalibrationChart
	RecordsList",
	"Data":[{
	"Probe":1,
	"Records":[{
	"Height":10,
	"Volume":123
	},{
	"Height":20,
	"Volume":456
	},{
	"Height":30,
	"Volume":789
	}]
	}]
	}]
	}

GPS RECEIVER REQUESTS AND RESPONSES

This part describes requests and responses used for getting GPS data from PTS-2 controller.

166. GetGpsData request

Purpose	Gets GPS data from GPS module
Data	Absent
Notes	It is possible to receive GPS data from PTS-2 controller when it is equipped with additional GPS module and permission for its application is allowed in parameters of PTS-2 controller
Permission required	Any permission among listed below: – *Configuration* – *Control* – *Monitoring* – *Reporting* – *Payments* – *Shifts*
Response	GpsData response

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"GetGpsData"
	}]
	}

167. GpsData response

Purpose	Returns last valid GPS data, received from GPS module
Data	"*Status*" – status of GPS data (string), possible values: '*Valid'*, '*Invalid', "Absent"* "*DateTime*" – date and time in format *YYYY-MM-DDThh:mm:ss* (string, 19 symbols), where: – *YYYY* – year (range from 2000 to 2099) – *MM* – month (range from 01 to 12) – *DD* – day (range from 01 to 31) – *T* – separator between date and time fields – *hh* – hours (range from 00 to 23) – *mm* – minutes (range from 00 to 59) – *ss* – seconds (range from 00 to 59) "*Latitude*" – latitude in format *ddmm.mmmm* (string, 9 symbols), where: – *dd* – degrees (range from 0 to 99) – *mm.mmmm* – minutes (range from 0.0000 to 99.9999) "*NorthSouthIndicator*" – north/south indicator (string), possible values: '*North'* and '*South*' "*Longitude*" – latitude in format *dddmm.mmmm* (string, 10 symbols), where: – *ddd* – degrees (range from 0 to 999) – *mm.mmmm* – minutes (range from 0.0000 to 99.9999) "*EastWestIndicator*" – east/west indicator (string), possible values: '*East'* and '*West*' "*SpeedOverGround*" – speed over ground in kilometers per hour (float, up to 2 digits after decimal point) "*CourseOverGround*" – course over ground in degrees (float, up to 2 digits after decimal point) "*Mode*" – operation mode (string), possible values: '*Autonomous*' and '*DGPS*'
Notes	1. In case if GPS module is not present then only a field *Status* with value *Absent* is present in response stating absence of data. 2. In case if GPS data is not valid (impossible to receive valid data from GPS module or from satellites) then only a field *Status* with value *Invalid* is present in response stating invalid data, other fields present in response relate to last previous valid GPS data. 3. UTC offset configured in PTS-2 controller using SetDateTime request is already taken into account in *DateTime* field.

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"GpsData",
	"Data":{
	"Status":"Valid",
	"DateTime":"2020-12-29T10:35:41",
	"Latitude":"3113.3156",
	"NorthSouthIndicator":"North",
	"Longitude":"12121.2686",

```
        "EastWestIndicator":"East",
        "SpeedOverGround":55.73,
        "CourseOverGround":193.93,
        "Mode":"Autonomous"
      }
   }]
}
```

PRICE BOARDS REQUESTS AND RESPONSES

This part describes requests and responses used for communication with price boards.

168. PriceBoardGetStatus request

Purpose	Gets status of price board
Data	"*PriceBoard*" – logical number of the price board (integer, range from 1 to 5)
Permission required	Any permission among listed below: – *Configuration* – *Control* – *Monitoring* – *Reporting*
Response	PriceBoardStatus response

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"PriceBoardGetStatus",
	"Data":{
	"PriceBoard":1
	}
	}]
	}

169. PriceBoardStatus response

Purpose	Returns status of price board
Data	"*PriceBoard*" – logical number of the price board (integer, range from 1 to 5) "*Online*" – state of reader communication (Boolean, possible values: "*true*", "*false*") "*Error*" – presence of error (Boolean, possible values: "*true*", "*false*")

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"PriceBoardGetStatus",
	"Data":{
	"PriceBoard":1,
	"Online":true,
	"Error":false
	}
	}]
	}

READERS REQUESTS AND RESPONSES

This part describes requests and responses used for communication with readers.

170. ReaderGetTag request

Purpose	Gets value of tag ID from reader
Data	"*Reader*" – logical number of the reader (integer, range from 1 to 120)
Permission required	Any permission among listed below: – *Configuration* – *Control* – *Monitoring* – *Reporting*
Response	ReaderTag response

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"ReaderGetTag",
	"Data":{
	"Reader":1
	}
	}]
	}

171. ReaderTag response

Purpose	Returns value of tag ID from reader
Data	"*Reader*" – logical number of the reader (integer, range from 1 to 120) "*Tag*" – value of tag ID (string, up to 32 hexadecimal symbols) "*Online*" – state of reader communication (Boolean, possible values: "*true*", "*false*") "*Error*" – presence of error (Boolean, possible values: "*true*", "*false*")
Note	If field "*Tag*" has empty value – it means that no tag identifier was read.

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"ReaderTag",
	"Data":{
	"Reader":1,
	"Tag":"1234567890ABCDEF",
	"Online":true,
	"Error":false
	}
	}]
	}

172. ReaderGetStatus request

Purpose	Gets status of reader
Data	"*Reader*" – logical number of the reader (integer, range from 1 to 120)
Permission required	Any permission among listed below: – *Configuration* – *Control* – *Monitoring* – *Reporting*
Response	ReaderStatus response

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"ReaderGetStatus",
	"Data":{
	"Reader":1
	}
	}]
	}

173. ReaderStatus response

Purpose	Returns status of reader
Data	"*Reader*" – logical number of the reader (integer, range from 1 to 120) "*Online*" – state of reader communication (Boolean, possible values: "*true*", "*false*") "*Error*" – presence of error (Boolean, possible values: "*true*", "*false*")

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"ReaderStatus",
	"Data":{
	"Reader":1,
	"Online":true,
	"Error":false
	}
	}]
	}

PAYMENTS REQUESTS AND RESPONSES

This part describes requests and responses used for payments.

174. PaymentCreateTransaction request

Purpose	Creates a payment transaction
Data	"*PumpTransactions*" – array of objects for each paid pump transaction: – *"Pump"* – logical number of the pump (integer, range from 1 to 120) – *"Transaction"* – number of transaction (integer, range from 1 to 65535) – "*DateTime*" – date and time of transaction end in format *YYYY-MM-DDThh:mm:ss* (string, 19 symbols), where: • *YYYY* – year (range from 2000 to 2099) • *MM* – month (range from 01 to 12) • *DD* – day (range from 01 to 31) • *T* – separator between date and time fields • *hh* – hours (range from 00 to 23) • *mm* – minutes (range from 00 to 59) • *ss* – seconds (range from 00 to 59) – *"Payments"* – array of objects for payments for the transaction, each element contains • *"Amount"* – payment sum (float, up to 3 digits after decimal point) • *"PaymentFormId"* – identifier of payment form (integer, range from 0 to 10)
Permission required	Any permission among listed below: – *Payments*
Notes	1. It is possible to pay only for pump transactions, which are stored inside the controller. 2. In case if there are several payments done for the same pump transaction – then their total payment sum must equal to the pump transaction amount and each of the payments must have a separate payment form.
Response	PaymentCreateTransactionConfirmation response

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"PaymentCreateTransaction",
	"Data":{
	"PumpTransactions":[{
	"Pump":1,
	"Transaction":1234,
	"DateTime":"2025-07-07T12:34:56",
	"Payments":[{
	"Amount": 12.34,
	"PaymentFormId":1

```
          },{
            "Amount": 56.78,
            "PaymentFormId":3
          }]
        },{
          "Pump":2,
          "Transaction":5678,
          "DateTime":"2025-07-07T12:47:37",
          "Payments":[{
            "Amount": 1.23,
            "PaymentFormId":2
          }]
        }]
      }
    }]
}
```

175. PaymentCreateTransactionConfirmation response

Purpose	Confirms successful payment and its transaction number
Data	"*PaymentTransaction*" – payment transaction number (integer, range from 1 to 999999999)

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"PaymentCreateTransactionConfirmation",
	"Data":{
	"PaymentTransaction":31
	}
	}]
	}

176. PaymentGetTransactionInformation request

Purpose	Requests payment transaction information
Data	"*PaymentTransaction*" – number of payment transaction (integer, range from 1 to 999999999)
Permission required	Any permission among listed below: – *Payments*
Note	If field "*PaymentTransaction*" is absent or its value equals 0 then information on the last performed is returned.
Response	PaymentTransactionInformation response

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"PaymentGetTransactionInformation",
	"Data":{
	"PaymentTransaction":31
	}
	}]
	}

177. PaymentTransactionInformation response

Purpose	Returns a payment transaction information
Data	"*PaymentTransaction*" – number of payment transaction (integer, range from 1 to 999999999) "*Transactions*" – array of objects for each paid pump transaction: – "*PaymentTransactionDateTime*" – date and time of payment transaction in format *YYYY-MM-DDThh:mm:ss* (string, 19 symbols), where: • *YYYY* – year (range from 2000 to 2099) • *MM* – month (range from 01 to 12) • *DD* – day (range from 01 to 31) • *T* – separator between date and time fields • *hh* – hours (range from 00 to 23) • *mm* – minutes (range from 00 to 59) • *ss* – seconds (range from 00 to 59) – "*Pump*" – logical number of the pump (integer, range from 1 to 120) – "*PumpTransaction*" – number of transaction (integer, range from 1 to 65535) – "*PumpTransactionDateTime*" – date and time of transaction end in format *YYYY-MM-DDThh:mm:ss* (string, 19 symbols), where: • *YYYY* – year (range from 2000 to 2099) • *MM* – month (range from 01 to 12) • *DD* – day (range from 01 to 31) • *T* – separator between date and time fields • *hh* – hours (range from 00 to 23) • *mm* – minutes (range from 00 to 59) • *ss* – seconds (range from 00 to 59) – *"Payments"* – array of payments for the transaction, each element contains • *"Amount"* – payment amount (float, up to 3 digits after decimal point) • "*IsRefunded*" - flag *showing* that the transaction was refunded (Boolean, possible values: "*true*", "*false*") • *"PaymentFormId*" – identifier of payment form (integer, range from 0 to 10) • "*PaymentFormName*" – name of payment form (string, up to 10 ASCII characters)
Permission required	Any permission among listed below: – *Payments*

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"PaymentTransactionInformation",
	"Data":{
	"PaymentTransaction":31,
	"Transactions":[{
	"PaymentTransactionDateTime":"2025-07-21T12:34:56",

```
          "Pump":1,
          "PumpTransaction":1234,
          "PumpTransactionDateTime":"2025-07-07T12:34:56",
          "Payments":[{
            "Amount": 12.34,
            "PaymentFormId":1,
            "PaymentFormName":"Cash",
            "IsRefunded":false
          },{
            "Amount": 56.78,
            "PaymentFormId":3,
            "PaymentFormName":"Waybill",
            "IsRefunded":false
          }]
        },{
          "PaymentTransactionDateTime":"2025-07-
21T12:34:56",
          "Pump":2,
          "PumpTransaction":5678,
          "PumpTransactionDateTime":"2025-07-07T12:47:37",
          "Payments":[{
            "Amount": 1.23,
            "PaymentFormId":2,
            "PaymentFormName":"Bank card",
            "IsRefunded":false
          }]
        }]
      }
    }]
}
```

178. PaymentRefundTransaction request

Purpose	Refunds a payment transaction
Data	"*PaymentTransaction*" – number of payment transaction (integer, range from 1 to 999999999)
Permission required	Any permission among listed below: – *Payments*
Response	Confirmation response message (in case of successful execution) or error response message (in case of any problem)

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"PaymentRefundTransaction",
	"Data":{
	"PaymentTransaction":31
	}
	}]
	}

SHIFT MANAGEMENT REQUESTS AND RESPONSES

This part describes requests and responses used for shifts management.

179. ShiftOpen request

Purpose	Opens a new working shift
Data	Absent
Permission required	Any permission among listed below: – *Shifts*
Response	ShiftOpenConfirmation response

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"ShiftOpen"
	}]
	}

180. ShiftOpenConfirmation response

Purpose	Confirms successful opening of the working shift and its sequence number
Data	"*ShiftNumber*" – working shift sequence number (integer, range from 1 to 999999999)

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"ShiftOpenConfirmation",
	"Data":{
	"Number":31
	}
	}]
	}

181. ShiftClose request

Purpose	Closes presently opened working shift
Data	Absent
Permission required	Any permission among listed below: – *Shifts*
Response	Confirmation response message (in case of successful execution) or error response message (in case of any problem)

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"ShiftClose"
	}]
	}

182. ShiftGetInformation request

Purpose	Requests information about a working shift
Data	"*Number*" – working shift sequence number (integer, range from 1 to 999999999)
Permission required	Any permission among listed below: – *Configuration* – *Control* – *Monitoring* – *Reports* – *Shifts*
Note	If field "*Number*" is absent or its value equals 0 then information on the presently opened or the last closest working shift is returned.
Response	ShiftInformation response

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"ShiftGetInformation",
	"Data":{
	"Number":456
	}
	}]
	}

183. ShiftInformation response

Purpose	Returns information about a working shift
Data	"*Enabled*" – flag showing if leading of working shift is enabled (Boolean, possible values: "*true*", "*false*") "*Number*" – working shift sequence number (integer, range from 1 to 999999999) "*IsOpened*" - flag showing if the working shift is presently opened (Boolean, possible values: "*true*", "*false*") "*DateTimeStart*" – date and time of the working shift start in format *YYYY-MM-DDThh:mm:ss* (string, 19 symbols), where: – *YYYY* – year (range from 2000 to 2099) – *MM* – month (range from 01 to 12) – *DD* – day (range from 01 to 31) – *T* – separator between date and time fields – *hh* – hours (range from 00 to 23) – *mm* – minutes (range from 00 to 59) – *ss* – seconds (range from 00 to 59) "*DateTimeEnd*" – date and time of the working shift end in format *YYYY-MM-DDThh:mm:ss* (string, 19 symbols), where: – *YYYY* – year (range from 2000 to 2099) – *MM* – month (range from 01 to 12) – *DD* – day (range from 01 to 31) – *T* – separator between date and time fields – *hh* – hours (range from 00 to 23) – *mm* – minutes (range from 00 to 59) – *ss* – seconds (range from 00 to 59) "*UserId*" – identifier of user, who opened the working shift (integer, range from 1 to 30)
Note	1. If field "*IsEnabled*" has value *false* then other fields are absent. 2. If field "*IsOpened*" has value *true* then field "*DateTimeEnd*" is absent.

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"ShiftInformation",
	"Data":{
	"Enabled":true,
	"Number":456,
	"IsOpened":true,
	"DateTimeStart":"2025-07-07T12:34:56",
	"UserId":3
	}
	}]
	}

REPORTS REQUESTS AND RESPONSES

This part describes requests and responses used for getting information for reports on pumps and probes/tanks.

184. ReportGetPumpTransactions request

Purpose	Gets list of transactions performed by pumps
Data	"*Pump*" – logical number of the pump (integer, range from 0 to 120), value 0 means all pumps "*Transaction*" – optional field for transaction number (integer, range from 0 to 65535) "*IsPaid*" – optional flag showing that transaction payment status (Boolean, possible values: "*true*", "*false*") "*DateTimeStart*" – date and time of transactions start in format *YYYY-MM-DDThh:mm:ss* (string, 19 symbols), where: – *YYYY* – year (range from 2000 to 2099) – *MM* – month (range from 01 to 12) – *DD* – day (range from 01 to 31) – *T* – separator between date and time fields – *hh* – hours (range from 00 to 23) – *mm* – minutes (range from 00 to 59) – *ss* – seconds (range from 00 to 59) "*DateTimeEnd*" – date and time of transactions end in format *YYYY-MM-DDThh:mm:ss* (string, 19 symbols), where: – *YYYY* – year (range from 2000 to 2099) – *MM* – month (range from 01 to 12) – *DD* – day (range from 01 to 31) – *T* – separator between date and time fields – *hh* – hours (range from 00 to 23) – *mm* – minutes (range from 00 to 59) – *ss* – seconds (range from 00 to 59)
Notes	1. When sending this request there should not be any other requests in a packet 2. In case if value of field "*Pump*" is 0 – then data on all pumps is returned 3. Field "*Transaction*" is optional. If field "*Transaction*" is absent or equals to 0 – then information on the all transactions is returned. 4. Field "*IsPaid*" is optional.
Permission required	Any permission among listed below: – *Reports*
Response	ReportPumpTransactions response

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"ReportGetPumpTransactions",

```
    "Data":{
      "Pump":1,
      "DateTimeStart":"2019-05-19T12:45:14",
      "DateTimeEnd":"2019-05-19T13:45:14"
    }
  }]
}
```

185. ReportPumpTransactions response

Purpose	Returns list of transactions performed by pumps
Data	Array of objects for each pump transaction: – "*DateTimeStart*" – date and time of transaction start in format *YYYY-MM-DDThh:mm:ss* (string, 19 symbols), where: • *YYYY* – year (range from 2000 to 2099) • *MM* – month (range from 01 to 12) • *DD* – day (range from 01 to 31) • *T* – separator between date and time fields • *hh* – hours (range from 00 to 23) • *mm* – minutes (range from 00 to 59) • *ss* – seconds (range from 00 to 59) – "*DateTime*" – date and time of transaction end in format *YYYY-MM-DDThh:mm:ss* (string, 19 symbols) – "*Pump*" – logical number of the pump (integer, range from 1 to 120) – "*Nozzle*" – nozzle number (integer, range from 1 to 6) – "*FuelGradeId*" – identifier of fuel grade (integer, range from 1 to 20) – "*FuelGradeName*" – name of fuel grade (string, up to 20 ASCII characters) – "*Transaction*" – number of transaction (integer, range from 1 to 65535) – "*Volume*" – transaction volume (float, up to 3 digits after decimal point) – "*TCVolume*" – transaction temperature-compensated volume (float, up to 3 digits after decimal point) – "*Price*" – product price (float, up to 3 digits after decimal point) – "*Amount*" – transaction money amount (float, up to 3 digits after decimal point) – "*TotalVolume*" – volume total counter value (float, up to 3 digits after decimal point) – "*TotalAmount*" – money amount total counter value (float, up to 3 digits after decimal point) – "*Tag*" – value of tag ID used to authorize the pump (string, up to 32 hexadecimal symbols) – "*FlowRate*" – value of the filling flow rate in volume units per minute (integer, range from 0 till 9999) – "*IsOffline*" – flag showing that transaction was done in offline (manual) mode of the pump (Boolean, possible values: "*true*", "*false*") – "*IsPaid*" – flag showing that transaction is paid (Boolean, possible values: "*true*", "*false*") – "*UserId*" – identifier of user, who authorized the transaction (integer, range from 1 to 30)
Notes	1. Value of field "*TCVolume*" is non-zero only in case if: a). pump nozzle is linked to tank and tank has installed probe, measuring temperature and b). pump nozzle is linked to fuel grade with configured temperature-expansion coefficient and c). value of dispensed volume is non-zero. 2. If field "*Tag*" has empty value – it means that no tag identifier was used for pump authorization. 3. Fields *"FuelGradeId"* and *"FuelGradeName"* are optional, they are present in response only when PTS-2 controller has pump nozzles configured to specific fuel grades.

	4. Fields "*IsOffline"* and "*IsPaid"* are optional, they present in response only in case if such values are present in the controller for the requested transaction.

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"ReportPumpTransactions",
	"Data":[{
	"DateTimeStart":"2019-05-19T12:44:01",
	"DateTime":"2019-05-19T12:45:14",
	"Pump":1,
	"Nozzle":1,
	"Transaction":15,
	"Volume":1.15,
	"TCVolume":1.14,
	"Price":10.00,
	"Amount":11.50,
	"TotalVolume":1231.15,
	"TotalAmount":45611.50,
	"FlowRate":25,
	"Tag":"0123456789ABCDEF",
	"UserId":1
	}, {
	"DateTimeStart":"2019-05-19T12:46:12",
	"DateTime":"2019-05-19T12:47:15",
	"Pump":1,
	"Nozzle":2,
	"Transaction":16,
	"Volume":3.00,
	"TCVolume":2.98,
	"Price":10.05,
	"Amount":30.15,
	"TotalVolume":1231.15,
	"TotalAmount":45611.50,
	"FlowRate":21,
	"Tag":"0000000000000000",
	"UserId":2
	}]
	}]
	}

186. ReportGetTankMeasurements request

Purpose	Gets list of measurements saved at changes of product height
Data	"*Tank*" – logical number of the tank (integer, range from 1 to 20), value 0 means all tanks "*DateTimeStart*" – date and time of measurements start in format *YYYY-MM-DDThh:mm:ss* (string, 19 symbols), where: – *YYYY* – year (range from 2000 to 2099) – *MM* – month (range from 01 to 12) – *DD* – day (range from 01 to 31) – *T* – separator between date and time fields – *hh* – hours (range from 00 to 23) – *mm* – minutes (range from 00 to 59) – *ss* – seconds (range from 00 to 59) "*DateTimeEnd*" – date and time of measurements end in format *YYYY-MM-DDThh:mm:ss* (string, 19 symbols), where: – *YYYY* – year (range from 2000 to 2099) – *MM* – month (range from 01 to 12) – *DD* – day (range from 01 to 31) – *T* – separator between date and time fields – *hh* – hours (range from 00 to 23) – *mm* – minutes (range from 00 to 59) – *ss* – seconds (range from 00 to 59)
Notes	When sending this request there should not be any other requests in a packet
Permission required	Any permission among listed below: – *Reports*
Response	ReportTankMeasurements response

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"ReportGetTankMeasurements",
	"Data":{
	"Tank":1,
	"DateTimeStart":"2019-05-19T12:45:14",
	"DateTimeEnd":"2019-05-19T13:45:14"
	}
	}]
	}

187. ReportTankMeasurements response

Purpose	Returns list of measurements saved at changes of product height
Data	Array of objects for each tank measurement: – "*DateTime*" – date and time of measurement in format *YYYY-MM-DDThh:mm:ss* (string, 19 symbols), where: • *YYYY* – year (range from 2000 to 2099) • *MM* – month (range from 01 to 12) • *DD* – day (range from 01 to 31) • *T* – separator between date and time fields • *hh* – hours (range from 00 to 23) • *mm* – minutes (range from 00 to 59) • *ss* – seconds (range from 00 to 59) – "*Tank*" – logical number of the tank (integer, range from 1 to 20) – "*FuelGradeId*" – identifier of fuel grade (integer, range from 1 to 20) – "*FuelGradeName*" – name of fuel grade (string, up to 20 ASCII characters) – "*Status*" – status of probe (string, up to 5 ASCII symbols), possible values: • "*OK*": in case of correct operation • "*Error*": in case of incorrect operation – "*Alarms*" – array of strings indicating probe alarms for tank, possible values: • "*CriticalHighProduct*": if product level is higher than maximum critical • "*HighProduct*": if product level is higher than maximum allowed • "*LowProduct*": if product level is lower than minimum allowed • "*CriticalLowProduct*": if product level is lower than minimum critical • "*HighWater*": in case if water level is higher than maximum allowed • "*TankLeakage*": in case if tank leakage is detected – "*ProductHeight*" – value of product height in mm (float, up to 1 digit after decimal point) – "*WaterHeight*" – value of water height in mm (float, up to 1 digit after decimal point) – "*Temperature*" – value product temperature in degrees Celcius (float, up to 1 digit after decimal point) – "*ProductVolume*" – value of product volume in liters (integer) – "*WaterVolume*" – value of water volume in liters (integer) – "*ProductUllage*" – value of product ullage in liters (integer) – "*ProductTCVolume*" – value of product temperature compensated volume in liters (integer) – "*ProductDensity*" – value of product density in kg/m^3 units (float, up to 1 digit after decimal point) – "*ProductMass*" – value of product mass in kg (float, up to 1 digit after decimal point) – "*TankFillingPercentage*" – valuf of tank filling percentage (integer)
Notes	1. This response contains only values measured by probe and calculated by the controller, so some of the values may not be present in response. 2. Array "*Alarms*" is optional and is present in response only in case if there are alarms present.

	3. Field "*TankFillingPercentage*" is optional and is calculated by the controller the following way: – as ratio of values of (filled product volume + filled water volume) and (filled product volume + filled water volume + ullage), if these values are present – as ratio of values of filled product height and full tank height, if these values are present 4. Fields "*FuelGradeId*" and "*FuelGradeName*" are optional, they are present in response only when PTS-2 controller has tank configured to specific fuel grade.

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"ReportTankMeasurements",
	"Data":[{
	"DateTime":"2019-05-19T12:45:14",
	"Tank":1,
	"Status":"OK",
	"Alarms":[
	"HighProduct"
	],
	"ProductHeight":2534.1,
	"WaterHeight":123.1,
	"Temperature":21.3,
	"ProductVolume":200,
	"WaterVolume":15,
	"ProductUllage":850,
	"ProductTCVolume":195,
	"ProductDensity":0.76,
	"ProductMass":190
	}]
	}]
	}

188. ReportGetInTankDeliveries request

Purpose	Gets list of in-tank deliveries saved
Data	"*Tank*" – logical number of the tank (integer, range from 1 to 20), value 0 means all tanks "*DateTimeStart*" – date and time of in-tank deliveries start in format *YYYY-MM-DDThh:mm:ss* (string, 19 symbols), where: – *YYYY* – year (range from 2000 to 2099) – *MM* – month (range from 01 to 12) – *DD* – day (range from 01 to 31) – *T* – separator between date and time fields – *hh* – hours (range from 00 to 23) – *mm* – minutes (range from 00 to 59) – *ss* – seconds (range from 00 to 59) "*DateTimeEnd*" – date and time of in-tank deliveries end in format *YYYY-MM-DDThh:mm:ss* (string, 19 symbols), where: – *YYYY* – year (range from 2000 to 2099) – *MM* – month (range from 01 to 12) – *DD* – day (range from 01 to 31) – *T* – separator between date and time fields – *hh* – hours (range from 00 to 23) – *mm* – minutes (range from 00 to 59) – *ss* – seconds (range from 00 to 59)
Notes	When sending this request there should not be any other requests in a packet
Permission required	Any permission among listed below: – *Reports*
Response	ReportInTankDeliveries response

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"ReportGetInTankDeliveries",
	"Data":{
	"Tank":1,
	"DateTimeStart":"2019-05-19T12:45:14",
	"DateTimeEnd":"2019-05-19T13:45:14"
	}
	}]
	}

189. ReportInTankDeliveries response

Purpose	Returns list of in-tank deliveries saved
Data	Array of objects for each in-tank delivery: – "*Tank*" – logical number of the tank (integer, range from 1 to 20), value 0 means all tanks – "*FuelGradeId*" – identifier of fuel grade (integer, range from 1 to 20) – "*FuelGradeName*" – name of fuel grade (string, up to 20 ASCII characters) – "*StartValues*" – object describing values on start of the in-tank delivery, contains following elements: • "*DateTime*" – date and time of last in-tank delivery start in format *YYYY-MM-DDThh:mm:ss* (string, 19 symbols), where: ▪ *YYYY* – year (range from 2000 to 2099) ▪ *MM* – month (range from 01 to 12) ▪ *DD* – day (range from 01 to 31) ▪ *T* – separator between date and time fields ▪ *hh* – hours (range from 00 to 23) ▪ *mm* – minutes (range from 00 to 59) ▪ *ss* – seconds (range from 00 to 59) • "*ProductHeight*" – value of product height in mm (float, up to 1 digit after decimal point) • "*WaterHeight*" – value of water height in mm (float, up to 1 digit after decimal point) • "*Temperature"* – value of temperature in degrees Celcius (float, up to 1 digit after decimal point) • *"ProductVolume*" – value of product volume in liters (integer) • "*ProductTCVolume*" – value of product temperature compensated volume in liters (integer) • "*ProductDensity*" – value of product density in kg/m^3 units (float, up to 1 digit after decimal point) • "*ProductMass*" – value of product mass in kg (float, up to 1 digit after decimal point) – "*EndValues*" – object describing values on end of the in-tank delivery, contains following elements: • "*DateTime*" – date and time of last in-tank delivery end in format *YYYY-MM-DDThh:mm:ss* (string, 19 symbols), where: ▪ *YYYY* – year (range from 2000 to 2099) ▪ *MM* – month (range from 01 to 12) ▪ *DD* – day (range from 01 to 31) ▪ *T* – separator between date and time fields ▪ *hh* – hours (range from 00 to 23) ▪ *mm* – minutes (range from 00 to 59) ▪ *ss* – seconds (range from 00 to 59) • *"ProductHeight"* – value of product height in mm (float, up to 1 digit after decimal point) • *"WaterHeight"* – value of water height in mm (float, up to 1 digit after decimal point)

	• *"Temperature"* – value of temperature in degrees Celcius (float, up to 1 digit after decimal point) • *"ProductVolume"* – value of product volume in liters (integer) • *"ProductTCVolume"* – value of product temperature compensated volume in liters (integer) • *"ProductDensity"* – value of product density in kg/m3 units (float, up to 1 digit after decimal point) • *"ProductMass*" – value of product mass in kg (float, up to 1 digit after decimal point) – "*AbsoluteValues*" – object describing absolute values of the in-tank delivery, contains following elements: • "*ProductHeight"* – value of product height in mm (float, up to 1 digit after decimal point) • *"WaterHeight*" – value of water height in mm (float, up to 1 digit after decimal point) • "*Temperature"* – value of temperature in degrees Celcius (float, up to 1 digit after decimal point) • *"ProductVolume"* – value of product volume in liters (integer) • *"ProductTCVolume"* – value of product temperature compensated volume in liters (integer) • *"ProductDensity"* – value of product density in kg/m3 units (float, up to 1 digit after decimal point) • *"ProductMass"* – value of product mass in kg (float, up to 1 digit after decimal point) • *"PumpsDispensedVolume*" – value of product volume dispensed through pumps during delivery in liters (float, up to 3 digits after decimal point)
Note	Fields "*FuelGradeId"* and "*FuelGradeName"* are optional, they are present in response only when PTS-2 controller has tank configured to specific fuel grade.

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"ReportInTankDeliveries",
	"Data":[{
	"Tank":1,
	"StartValues": {
	"DateTime": "2019-03-09T15:30:25",
	"ProductHeight": 2534.1,
	"WaterHeight": 123.1,
	"Temperature": 21.3,
	"ProductVolume": 2005,
	"ProductTCVolume": 2015
	},
	"EndValues": {
	"DateTime": "2019-03-09T15:32:15",
	"ProductHeight": 2725.1,

```
          "WaterHeight": 130.4,
          "Temperature": 21.6,
          "ProductVolume": 2514,
          "ProductTCVolume": 2601
        },
        "AbsoluteValues": {
          "ProductDensity": 758.9,
          "ProductMass": 386.3,
          "PumpsDispensedVolume": 25.3
        }
      }]
    }]
}
```

190. ReportGetGpsRecords request

Purpose	Gets list of GPS records saved
Data	"*DateTimeStart*" – date and time of GPS records start in format *YYYY-MM-DDThh:mm:ss* (string, 19 symbols), where: – *YYYY* – year (range from 2000 to 2099) – *MM* – month (range from 01 to 12) – *DD* – day (range from 01 to 31) – *T* – separator between date and time fields – *hh* – hours (range from 00 to 23) – *mm* – minutes (range from 00 to 59) – *ss* – seconds (range from 00 to 59) "*DateTimeEnd*" – date and time of GPS records end in format *YYYY-MM-DDThh:mm:ss* (string, 19 symbols), where: – *YYYY* – year (range from 2000 to 2099) – *MM* – month (range from 01 to 12) – *DD* – day (range from 01 to 31) – *T* – separator between date and time fields – *hh* – hours (range from 00 to 23) – *mm* – minutes (range from 00 to 59) – *ss* – seconds (range from 00 to 59)
Notes	When sending this request there should not be any other requests in a packet
Permission required	Any permission among listed below: – *Reports*
Response	ReportGpsRecords response

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"ReportGetGpsRecords",
	"Data":{
	"DateTimeStart":"2019-05-19T12:45:14",
	"DateTimeEnd":"2019-05-19T13:45:14"
	}
	}]
	}

191. ReportGpsRecords response

Purpose	Returns list of GPS records saved
Data	Array of objects for each GPS record: – "*Status*" – status of GPS data (string), possible values: '*Valid*' and '*Invalid*' – "*DateTime*" – date and time in format *YYYY-MM-DDThh:mm:ss* (string, 19 symbols), where: • *YYYY* – year (range from 2000 to 2099) • *MM* – month (range from 01 to 12) • *DD* – day (range from 01 to 31) • *T* – separator between date and time fields • *hh* – hours (range from 00 to 23) • *mm* – minutes (range from 00 to 59) • *ss* – seconds (range from 00 to 59) – "*Latitude*" – latitude in format *ddmm.mmmm* (string, 9 symbols), where: • *dd* – degrees (range from 0 to 99) • *mm.mmmm* – minutes (range from 0.0000 to 99.9999) – "*NorthSouthIndicator*" – north/south indicator (string), possible values: '*North*' and '*South*' – "*Longitude*" – latitude in format *dddmm.mmmm* (string, 10 symbols), where: • *ddd* – degrees (range from 0 to 999) • *mm.mmmm* – minutes (range from 0.0000 to 99.9999) – "*EastWestIndicator*" – east/west indicator (string), possible values: '*East*' and '*West*' – "*SpeedOverGround*" – speed over ground in kilometers per hour (float, up to 2 digits after decimal point) – "*CourseOverGround*" – course over ground in degrees (float, up to 2 digits after decimal point) – "*Mode*" – operation mode (string), possible values: '*Autonomous*' and '*DGPS*'

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"ReportGpsRecords",
	"Data":[{
	"DateTime":"2020-12-29T10:35:41",
	"Latitude":"3113.3156",
	"NorthSouthIndicator":"North",
	"Longitude":"12121.2686",
	"EastWestIndicator":"East",
	"SpeedOverGround":55.73,
	"CourseOverGround":193.93,
	"Mode":"Autonomous"
	}]
	}]

	}

192. ReportGetAlertRecords request

Purpose	Gets list of alert records saved
Data	"*DateTimeStart*" – date and time of alert records start in format *YYYY-MM-DDThh:mm:ss* (string, 19 symbols), where: – *YYYY* – year (range from 2000 to 2099) – *MM* – month (range from 01 to 12) – *DD* – day (range from 01 to 31) – *T* – separator between date and time fields – *hh* – hours (range from 00 to 23) – *mm* – minutes (range from 00 to 59) – *ss* – seconds (range from 00 to 59) "*DateTimeEnd*" – date and time of alert records end in format *YYYY-MM-DDThh:mm:ss* (string, 19 symbols), where: – *YYYY* – year (range from 2000 to 2099) – *MM* – month (range from 01 to 12) – *DD* – day (range from 01 to 31) – *T* – separator between date and time fields – *hh* – hours (range from 00 to 23) – *mm* – minutes (range from 00 to 59) – *ss* – seconds (range from 00 to 59)
Notes	When sending this request there should not be any other requests in a packet
Permission required	Any permission among listed below: – *Reports*
Response	ReportAlertRecords response

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"ReportGetAlertRecords",
	"Data":{
	"DateTimeStart":"2019-05-19T12:45:14",
	"DateTimeEnd":"2019-05-19T13:45:14"
	}
	}]
	}

193. ReportAlertRecords response

Purpose	Returns list of alert records saved
Data	Array of objects for each alert record: – "*DateTime*" – date and time in format *YYYY-MM-DDThh:mm:ss* (string, 19 symbols), where: • *YYYY* – year (range from 2000 to 2099) • *MM* – month (range from 01 to 12) • *DD* – day (range from 01 to 31) • *T* – separator between date and time fields • *hh* – hours (range from 00 to 23) • *mm* – minutes (range from 00 to 59) • *ss* – seconds (range from 00 to 59) – "*DeviceType*" – strings indicating device type, possible values: • "*PTS*": if alert is related to the PTS-2 controller • "*Pump*": if alert is related to pump • "*Probe*": if alert is related to probe • "*PriceBoard*": if alert is related to price board • "*Reader*": if alert is related to reader – "*DeviceNumber*" – number of device, to which the alert relate (integer, range from 1 to 120) – "*State*" – strings indicating alert state, possible values: • "*Started*" – in case if the alert is started and has duration • "*Finished*" – in case if the alert is finished and had duration • *"Detected"* – in case if the alert is detected and does not have duration – "*Code*" –alert code (integer, range from 1 to 99999)
Note	Values of alert codes is given in section ALERT CODES

Example	{
	"Protocol": "jsonPTS",
	"Packets": [{
	"Id": 1,
	"Type": "ReportAlertRecords",
	"Data": [{
	"DateTime": "2022-09-24T21:46:47",
	"DeviceType": "PTS",
	"DeviceNumber": 0,
	"State": "Started",
	"Code": 1
	}, {
	"DateTime": "2022-09-24T21:46:47",
	"DeviceType": "PTS",
	"DeviceNumber": 0,
	"State": "Finished",
	"Code": 1

```
      }, {
        "DateTime": "2022-09-24T21:50:21",
        "DeviceType": "Probe",
        "DeviceNumber": 2,
        "State": "Started",
        "Code": 1
      }, {
        "DateTime": "2022-09-24T21:50:26",
        "DeviceType": "Probe",
        "DeviceNumber": 2,
        "State": "Finished",
        "Code": 1
      }]
    }]
}
```

194. ReportGetPayments request

Purpose	Gets list of performed payments
Data	"*DateTimeStart*" – date and time of transactions start in format *YYYY-MM-DDThh:mm:ss* (string, 19 symbols), where: – *YYYY* – year (range from 2000 to 2099) – *MM* – month (range from 01 to 12) – *DD* – day (range from 01 to 31) – *T* – separator between date and time fields – *hh* – hours (range from 00 to 23) – *mm* – minutes (range from 00 to 59) – *ss* – seconds (range from 00 to 59) "*DateTimeEnd*" – date and time of transactions end in format *YYYY-MM-DDThh:mm:ss* (string, 19 symbols), where: – *YYYY* – year (range from 2000 to 2099) – *MM* – month (range from 01 to 12) – *DD* – day (range from 01 to 31) – *T* – separator between date and time fields – *hh* – hours (range from 00 to 23) – *mm* – minutes (range from 00 to 59) – *ss* – seconds (range from 00 to 59)
Note	When sending this request there should not be any other requests in a packet
Permission required	Any permission among listed below: – *Reports*
Response	ReportPayments response

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"ReportGetPayments",
	"Data":{
	"DateTimeStart":"2025-07-20T12:45:14",
	"DateTimeEnd":"2025-07-21T13:45:14"
	}
	}]
	}

195. ReportPayments response

Purpose	Returns list of performed payments
Data	Array of objects for each payment transaction: – "*PaymentTransactionDateTime*" – date and time of payment transaction in format *YYYY-MM-DDThh:mm:ss* (string, 19 symbols), where: • *YYYY* – year (range from 2000 to 2099) • *MM* – month (range from 01 to 12) • *DD* – day (range from 01 to 31) • *T* – separator between date and time fields • *hh* – hours (range from 00 to 23) • *mm* – minutes (range from 00 to 59) • *ss* – seconds (range from 00 to 59) – "*PaymentTransaction*" – number of payment transaction (integer, range from 1 to 999999999) – "*Pump*" – logical number of the pump (integer, range from 1 to 120) – "*PumpTransaction*" – number of pump transaction (integer, range from 1 to 65535) – "*PumpTransactionDateTime*" – date and time of pump transaction end in format *YYYY-MM-DDThh:mm:ss* (string, 19 symbols), where: • *YYYY* – year (range from 2000 to 2099) • *MM* – month (range from 01 to 12) • *DD* – day (range from 01 to 31) • *T* – separator between date and time fields • *hh* – hours (range from 00 to 23) • *mm* – minutes (range from 00 to 59) • *ss* – seconds (range from 00 to 59) – *"Amount"* – payment amount (float, up to 3 digits after decimal point) – *"PaymentFormId"* – identifier of payment form (integer, range from 0 to 10) – "*PaymentFormName*" – name of payment form (string, up to 10 ASCII characters) – "*IsRefunded*" - flag showing that the transaction was refunded (Boolean, possible values: "*true*", "*false*")

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"ReportPayments",
	"Data":[{
	"PaymentTransactionDateTime":"2025-07-21T12:34:56",
	"PaymentTransaction":15,
	"Pump":1,
	"PumpTransaction":155,
	"PumpTransactionDateTime":"2025-07-21T11:22:33",
	"Amount":1.14,
	"PaymentFormId":1,

```
        "PaymentFormName":"Cash",
        "IsRefunded":false
      },{
        "DateTime":"2025-07-21T12:34:56",
        "PaymentTransaction":15,
        "Pump":1,
        "PumpTransaction":155,
        "PumpTransactionDateTime":"2025-07-21T11:22:33",
        "Amount":2.34,
        "PaymentFormId":2,
        "PaymentFormName":"Bank card",
        "IsRefunded":false
      }]
    }]
}
```

196. ReportGetShifts request

Purpose	Gets list of closed working shifts
Data	"*DateTimeStart*" – date and time of the working shifts start in format *YYYY-MM-DDThh:mm:ss* (string, 19 symbols), where: – *YYYY* – year (range from 2000 to 2099) – *MM* – month (range from 01 to 12) – *DD* – day (range from 01 to 31) – *T* – separator between date and time fields – *hh* – hours (range from 00 to 23) – *mm* – minutes (range from 00 to 59) – *ss* – seconds (range from 00 to 59) "*DateTimeEnd*" – date and time of the working shifts end in format *YYYY-MM-DDThh:mm:ss* (string, 19 symbols), where: – *YYYY* – year (range from 2000 to 2099) – *MM* – month (range from 01 to 12) – *DD* – day (range from 01 to 31) – *T* – separator between date and time fields – *hh* – hours (range from 00 to 23) – *mm* – minutes (range from 00 to 59) – *ss* – seconds (range from 00 to 59)
Note	When sending this request there should not be any other requests in a packet
Permission required	Any permission among listed below: – *Reports*
Response	ReportShifts response

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"ReportGetShifts",
	"Data":{
	"DateTimeStart":"2025-07-20T12:45:14",
	"DateTimeEnd":"2025-07-21T13:45:14"
	}
	}]
	}

197. ReportShifts response

Purpose	Returns list of closed working shifts
Data	Array of objects for each payment transaction: – "*Number*" – working shift sequence number (integer, range from 1 to 999999999) – "*DateTimeStart*" – date and time of the working shift start in format *YYYY-MM-DDThh:mm:ss* (string, 19 symbols), where: • *YYYY* – year (range from 2000 to 2099) • *MM* – month (range from 01 to 12) • *DD* – day (range from 01 to 31) • *T* – separator between date and time fields • *hh* – hours (range from 00 to 23) • *mm* – minutes (range from 00 to 59) • *ss* – seconds (range from 00 to 59) – "*DateTimeEnd*" – date and time of the working shift end in format *YYYY-MM-DDThh:mm:ss* (string, 19 symbols), where: • *YYYY* – year (range from 2000 to 2099) • *MM* – month (range from 01 to 12) • *DD* – day (range from 01 to 31) • *T* – separator between date and time fields • *hh* – hours (range from 00 to 23) • *mm* – minutes (range from 00 to 59) • *ss* – seconds (range from 00 to 59) – "*UserId*" – identifier of user, who opened the working shift (integer, range from 1 to 30)
Note	If field "*DateTimeEnd*" is absent – then the shift is opened and not closed yet.

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"ReportShifts",
	"Data":[{
	"DateTimeStart":"2025-07-21T12:34:56",
	"DateTimeEnd":"2025-07-22T12:00:00",
	"Number":152,
	"UserId":1
	},{
	"DateTimeStart":"2025-07-22T12:34:56",
	"DateTimeEnd":"2025-07-23T12:00:00",
	"Number":153,
	"UserId":1
	}]
	}]

	}

SELF-DIAGNOSTICS REQUESTS AND RESPONSES

This part describes requests and responses used for provision of self-diagnostics of PTS-2 controller.

198. MakeDiagnostics request

Purpose	Make peripherals self-diagnostics
Data	Absent
Notes	1. While this request is being executed the PTS-2 controller stops communication with pumps and probes 2. For diagnostics of pump ports they should be interconnected with other: – all pump ports lines A should be interconnected with each other – all pump ports lines B should be interconnected with each other 3. For diagnostics of probe ports each of them should have line TxD connected to line RxD
Permission required	Any permission among listed below: – *Configuration* – *Control* – *Monitoring*
Response	Diagnostics response

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"MakeDiagnostics"
	}]
	}

199. Diagnostics response

Purpose	Returns results of peripherals self-diagnostics
Data	"*PumpPort1*" – states of RS-485 ports, which receive a request from this port: value "*OK*" or "*Error*" (each element is string, up to 5 ASCII symbols) "*PumpPort2*" – states of RS-485 ports, which receive a request from this port: value "*OK*" or "*Error*" (each element is string, up to 5 ASCII symbols) "*PumpPort3*" – states of RS-485 ports, which receive a request from this port: value "*OK*" or "*Error*" (each element is string, up to 5 ASCII symbols) "*PumpPort4*" – states of RS-485 ports, which receive a request from this port: value "*OK*" or "*Error*" (each element is string, up to 5 ASCII symbols) "*DispPort485*" – states of RS-485 ports, which receive a request from this port: value "*OK*" or "*Error*" (string, up to 5 ASCII symbols) "*LogPort485*" – states of RS-485 ports, which receive a request from this port: value "*OK*" or "*Error*" (string, up to 5 ASCII symbols) "*UserPort485*" – states of RS-485 ports, which receive a request from this port: value "*OK*" or "*Error*" (string, up to 5 ASCII symbols) "*DispPort232*" – state of DISP (RS-232) port: value "*OK*" or "*Error*" (string, up to 5 ASCII symbols) "*LogPort232*" – state of LOG (RS-232) port: value "*OK*" or "*Error*" (string, up to 5 ASCII symbols) "*UserPort232*" – state of USER (RS-232) port: value "*OK*" or "*Error*" (string, up to 5 ASCII symbols) "*PcPort232*" – state of PC (RS-232) port: value "*OK*" or "*Error*" (string, up to 5 ASCII symbols) "*ExtPort*" – state of EXT port: value "*OK*" or "*Error*" (string, up to 5 ASCII symbols) "*DIP*" – array of each switch states on 4-position DIP-switch: value "*On*" or "Off" (each is string, up to 3 ASCII symbols) "*SD*" – state of SD flash disk: value "*OK*" or "*Error*" (string, up to 5 ASCII symbols) "*DateTime*" – date and time of RTC (realtime clock) in format *YYYY-MM-DDThh:mm:ss* (string, 19 symbols), where: – *YYYY* – year (range from 2000 to 2099) – *MM* – month (range from 01 to 12) – *DD* – day (range from 01 to 31) – *T* – separator between date and time fields – *hh* – hours (range from 00 to 23) – *mm* – minutes (range from 00 to 59) – *ss* – seconds (range from 00 to 59)

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"Diagnostics",
	"Data":{
	"PumpPort1":{
	"PumpPort2":"OK",
	"PumpPort3":"OK",

```
        "PumpPort4":"OK",
        "DispPort485":"Error",
        "LogPort485":"Error",
        "UserPort485":"Error"
      },
      "PumpPort2":{
        "PumpPort1":"OK",
        "PumpPort3":"OK",
        "PumpPort4":"OK",
        "DispPort485":"Error",
        "LogPort485":"Error",
        "UserPort485":"Error"
      },
      "PumpPort3":{
        "PumpPort1":"OK",
        "PumpPort2":"OK",
        "PumpPort4":"OK",
        "DispPort485":"Error",
        "LogPort485":"Error",
        "UserPort485":"Error"
      },
      "PumpPort4":{
        "PumpPort1":"OK",
        "PumpPort2":"OK",
        "PumpPort3":"OK",
        "DispPort485":"Error",
        "LogPort485":"Error",
        "UserPort485":"Error"
      },
      "DispPort485":{
        "PumpPort1":"Error",
        "PumpPort2":"Error",
        "PumpPort3":"Error",
        "PumpPort4":"Error",
        "LogPort485":"Error",
        "UserPort485":"Error"
      },
      "LogPort485":{
        "PumpPort1":"Error",
        "PumpPort2":"Error",
        "PumpPort3":"Error",
        "PumpPort4":"Error",
        "DispPort485":"Error",
```

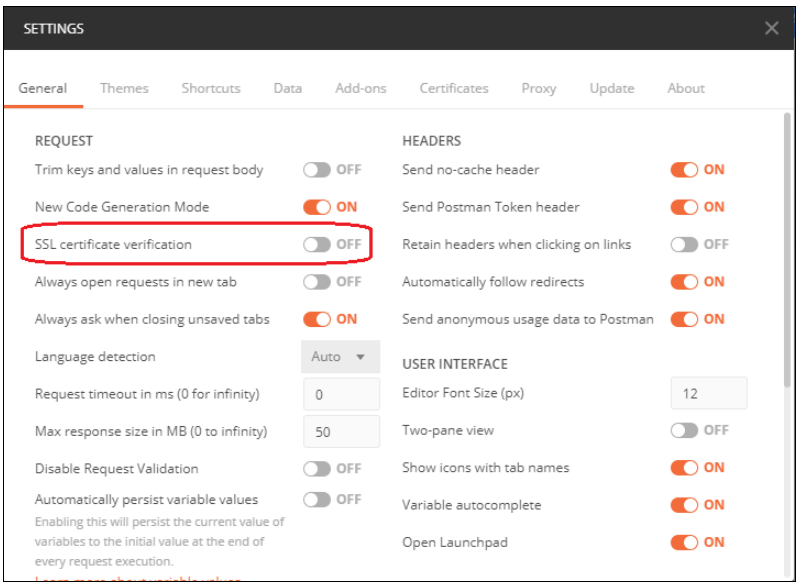
```
        "UserPort485":"Error"
      },
      "UserPort485":{
        "PumpPort1":"Error",
        "PumpPort2":"Error",
        "PumpPort3":"Error",
        "PumpPort4":"Error",
        "DispPort485":"Error",
        "LogPort485":"Error"
      },
      "DispPort232":"OK",
      "LogPort232":"OK",
      "UserPort232":"OK",
      "PcPort232":"OK",
      "ExtPort":"OK",
      "DIP":[
        "On",
        "Off",
        "Off",
        "Off"
      ],
      "SD":"OK",
      "DateTime":"2020-12-29T08:05:41"
    }
  }]
}
```

TESTING USING POSTMAN UTILITY

Postman utility provided by Postman, Inc (https://www.postman.com/) is a great tool to test communication with PTS-2 controller. Here is a short guide on how to configure it for operation.

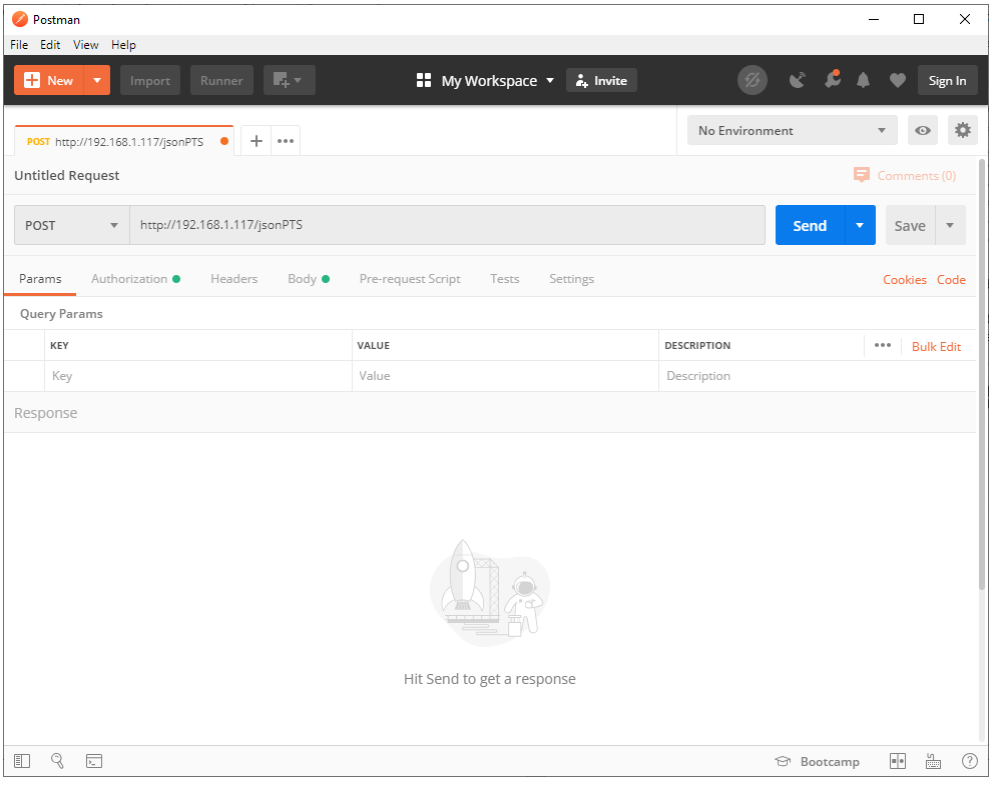
1. Download Postman utility from official web-site: https://www.postman.com/downloads/

2. In case if you are using communication over HTTPS (configuration DIP-switch 1 is set to OFF position) – then you need to disable SSL certificate verification in settings of Postman, this is required cause PTS-2 controller is using a self-signed SSL certificate (for HTTP communication this setting is not important):



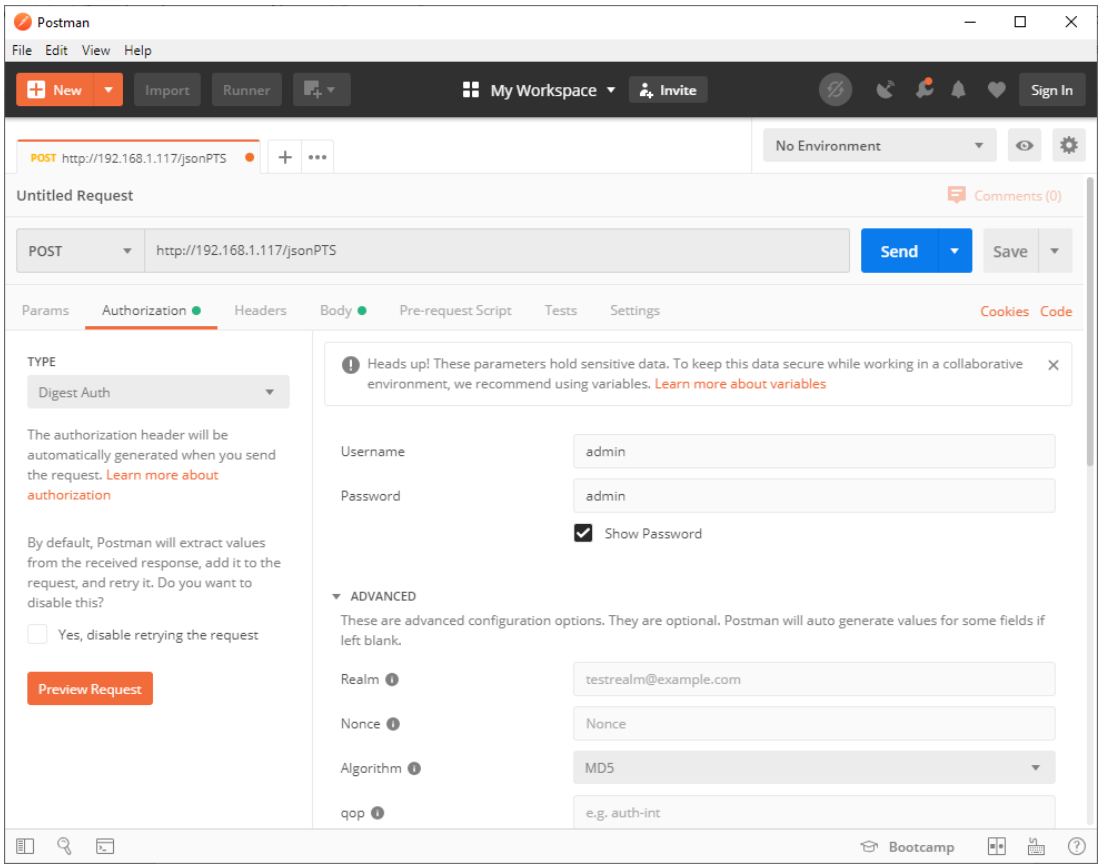
3. In Postman you need to select type of request to POST and input the address, which is

   - http://192.168.1.117/jsonPTS - in case of communication is set to use HTTP
   - https://192.168.1.117/jsonPTS - in case of communication is set to use HTTPS

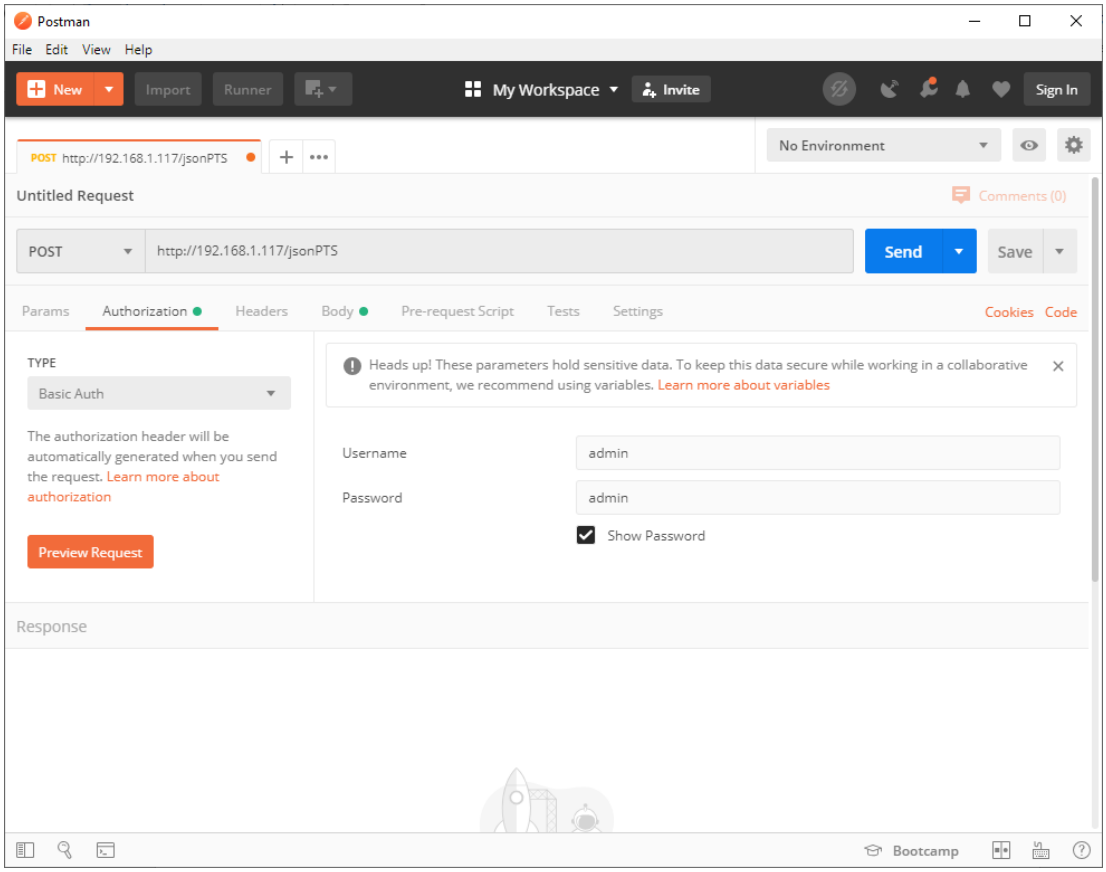


4. On tab Authorization we need to set authorization type configured using configuration DIP-switch 2:

   - in case if DIP-switch 2 is set to OFF – we need to select value Digest Auth and enter login and password:



   - in case if DIP-switch 2 is set to ON – we need to select value Basic Auth and enter login and password:

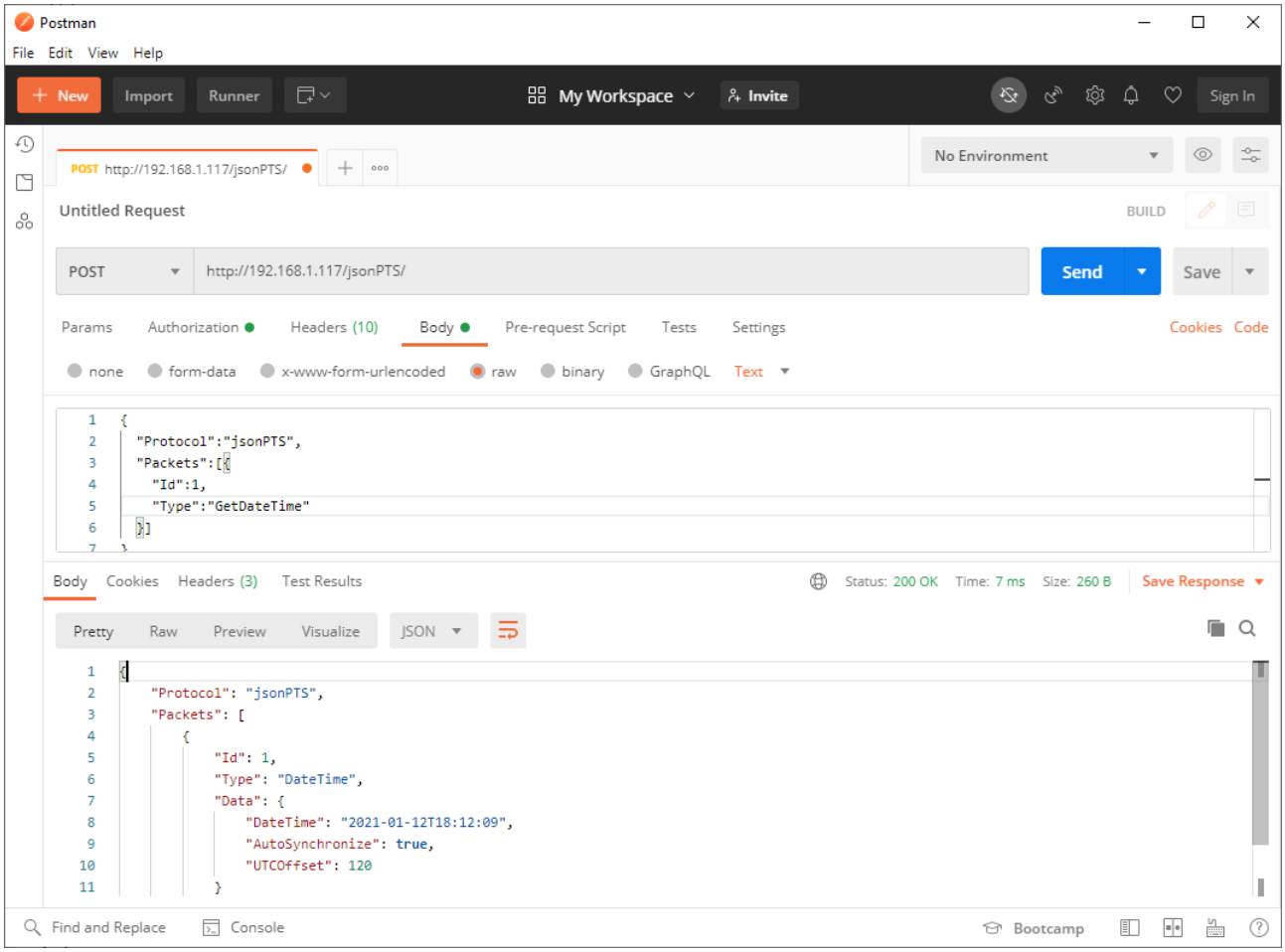


5. On tab Headers we need to leave all the fields empty (Postman will automatically fill some of them on execution of the request).

6. On tab Body you need to switch to raw data type and insert a request, for example GetDateTime request:

```
{
  "Protocol":"jsonPTS",
  "Packets":[{
    "Id":1,
    "Type":"GetDateTime"
  }]
}
```

7. At clicking Send button the request should be sent to PTS-2 controller and you should receive a response from it:



COMMUNICATION TO A REMOTE SERVER USING WEBSOCKET PROTOCOL

When there is a need to make a remote server to communicate to the PTS-2 controller for any general purposes (for example getting/setting configuration, control/monitoring of pumps/tanks/readers/price-boards, generating reports, or any other possible) - then communication of the PTS-2 controller and the remote server can done using the WebSocket protocol (according to RFC6455) when the PTS-2 controller connects to the remote server as a client and establishes a constant communication, so that the remote server can get/set any information from/to the PTS-2 controller any time.

All communication using the WebSocket protocol is done in accordance with the RFC6455 document, which can be found here: *https://datatracker.ietf.org/doc/html/rfc6455*. Application data for the WebSocket protocol is formed from requests and responses described in previous sections this document, responses from the PTS-2 controller can be fragmented.

The WebSocket protocol has two parts: a handshake and the data transfer. For establishing a connection, the PTS-2 controller and the remote server perform a handshake and after that start the data transfer.

Each packet sent to server contains a unique identifier of PTS-2 controller in field "*PtsId*", which is a string (up to 24 hexadecimal digits), so that the remote server could easily identify it and distinguish from other PTS-2 controllers, which might also communicate with this remote server. Also, the same value is present in a HTTP header named "*X-Pts-Id"*. Also, the same value can be also received from PTS-2 controller using GetUniqueIdentifier request.

In each of the messages the value of the PTS-2 controller firmware date/time is sent in a header named "*X-Pts-Firmware-Version-DateTime*".

In each of the messages the value of the PTS-2 controller configuration identifier is sent in a header named "*X-Pts-Configuration-Identifier*", which is used to point if configuration was changed anyhow: given identifier is assigned a new unique value at any change of configuration, thus an application may know if the configuration was changed based on value of this identifier.

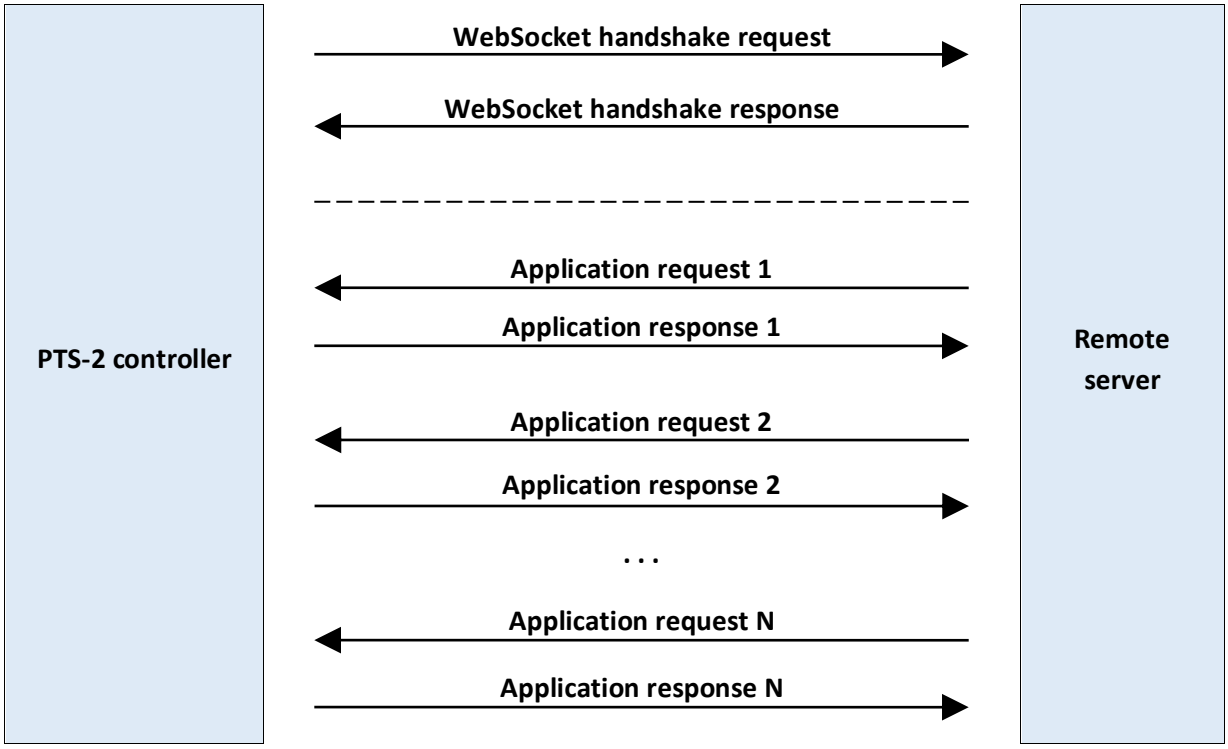
The handshake from the PTS-2 controller looks as follows:

```
GET /ptsWebSocket HTTP/1.1
Host: 192.168.1.246:54098
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Sec-WebSocket-Version: 13
Origin: http://192.168.1.117
X-Pts-Id: 0041001C524E500420323442
X-Pts-Firmware-Version-DateTime: 2021-11-12T08:05:41
X-Pts-Configuration-Identifier: b99cafbb
```

The handshake from the remote server looks as follows:

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
Sec-WebSocket-Protocol: chat
```

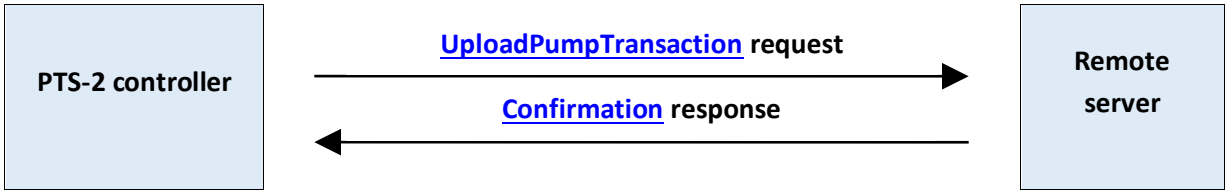
Once the client and server have both sent their handshakes, and if the handshake was successful, then the data transfer part starts. At this the remote server is to send requests described in this document and the PTS-2 controller is to respond to them.



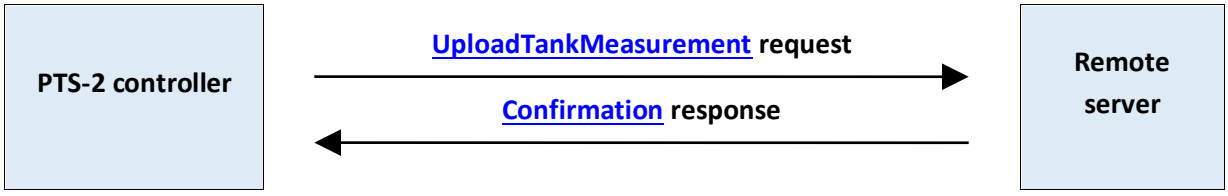
PTS-2 controller regularly polls the remote server if the connection is still present using a Ping request, to which the remote server should respond with a Pong response. In case if the remote server does not respond to any of Ping requests – then the PTS-2 controller immediately closes the connection and reconnects to the remote server.

PTS-2 controller can be configured to asynchronously send following data to the remote server automatically without a need to make any request:

1. Uploading a performed pump transaction to a remote server – in this case PTS-2 controller is using UploadPumpTransaction request:

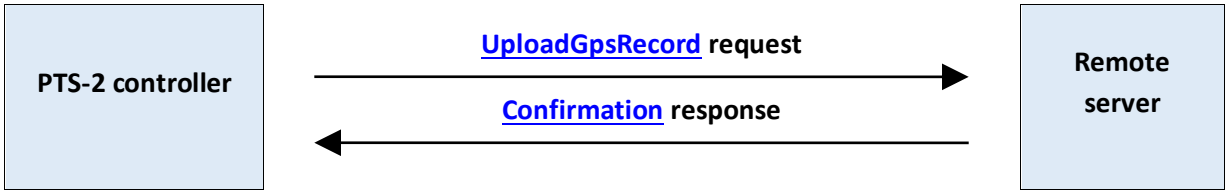


2. Uploading a measured tank data to a remote server – in this case PTS-2 controller is using UploadTankMeasurement request:

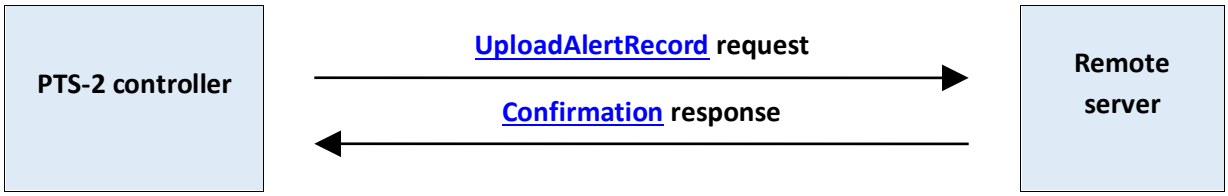


3. Uploading a registered in-tank delivery data to a remote server – in this case PTS-2 controller is using UploadInTankDelivery request:

4. Uploading a registered GPS record to a remote server – in this case PTS-2 controller is using UploadGpsRecord request:

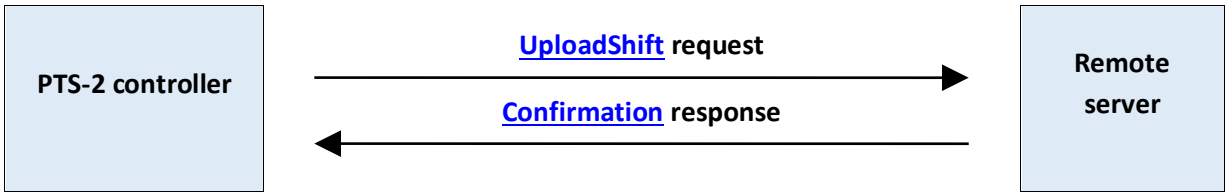


5. Uploading a registered alert record to a remote server – in this case PTS-2 controller is using UploadAlertRecord request:

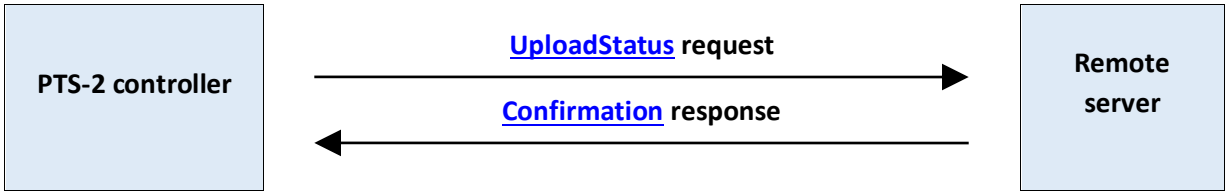


6. Uploading a registered payment record to a remote server – in this case PTS-2 controller is using UploadPayment request:

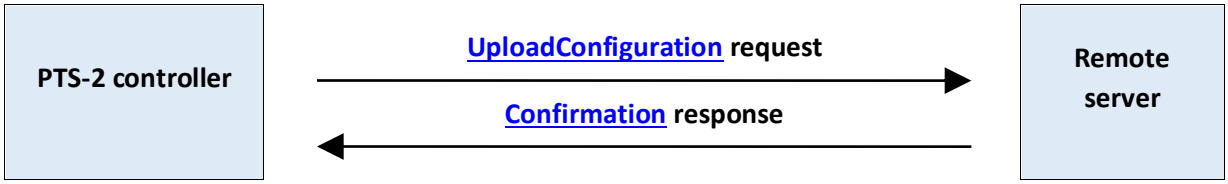
7. Uploading a registered working shift record to a remote server – in this case PTS-2 controller is using UploadShift request:



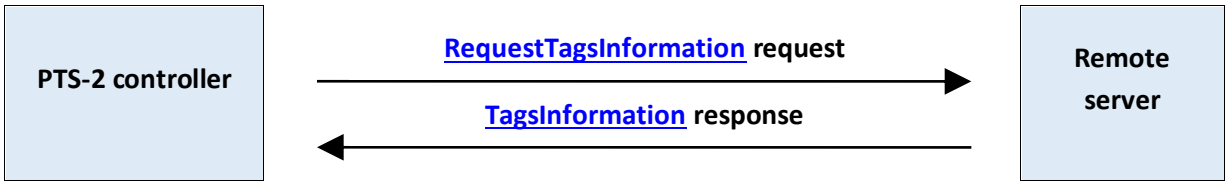
8. Uploading a realtime status of all connected forecourt equipment and status of the PTS-2 controller itself for realtime online monitoring – in this case PTS-2 controller is using UploadStatus request:



9. Uploading configuration to a remote server – in this case PTS-2 controller is using UploadConfiguration request:



10. Requesting tags accounts' balance from a remote server – in this case PTS-2 controller is using RequestTagsInformation request:



UploadStatus request is used for periodical sending by the PTS-2 controller to the remote server for check of communication state and realtime online monitoring of the controller. It provides the remote server with online realtime data about the PTS-2 controller and peripheral equipment statuses (pumps, probes, price boards, readers, GPS receiver, others), so that the remote server can have online information on everything going with the PTS-2 controller.

UploadConfiguration request is sent to the remote server with present complete configuration of the PTS-2 controller. It is sent if there is a configuration change in the PTS-2 controller – then the PTS-2 controller automatically uploads a new configuration to the server once.

RequestTagsInformation request is sent to the remote server to validate the tags and to check their accounts' balance remains. It can be used for the purpose of application of the tags for payment for the pump transactions.

Each packet sent by the controller to server contains a unique ID (range from 1 to 65535). Response of the server should be in format of confirmation or error response message and should contain the same packet ID as received in the packet from the controller.

In case of UploadPumpTransaction, UploadTankMeasurement, UploadInTankDelivery, UploadGpsRecord, UploadAlertRecord, UploadPayment, UploadShift and UploadConfiguration requests the PTS-2 controller will be sending same message to the remote server until it is accepted by the remote server, which means that the remote server should respond to it with a confirmation message.

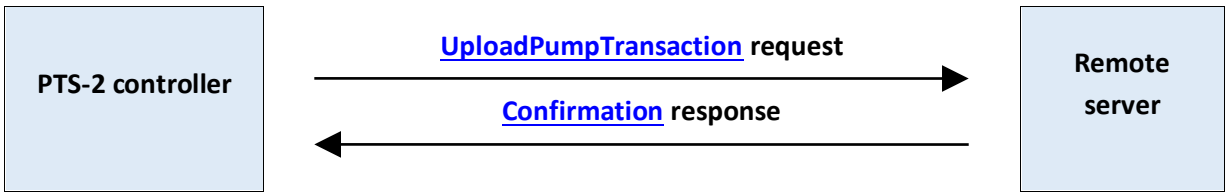
If option "*UseWebSocketsCommunication*" is enabled in SetRemoteServerConfiguration request – then the PTS-2 controller is listening for incoming requests from the remote server, so the remote server can send any request to the PTS-2 controller any time and the PTS-2 controller will respond.

COMMUNICATION TO A REMOTE SERVER USING HTTP REQUESTS

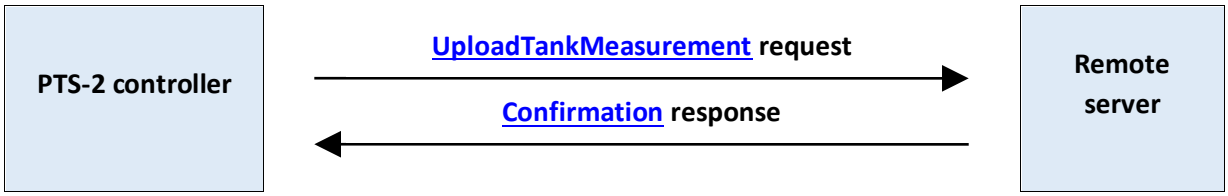
Communication of the PTS-2 controller to a remote server can done using simple HTTP requests, which the PTS-2 controller as a client sends to the remote server. It allows a simple approach to get to get all important information from the PTS-2 controller to the remote server.

There are several types of requests used by the PTS-2 controller for data upload to a remote server:

1. Uploading a performed pump transaction to a remote server – in this case PTS-2 controller is using UploadPumpTransaction request:

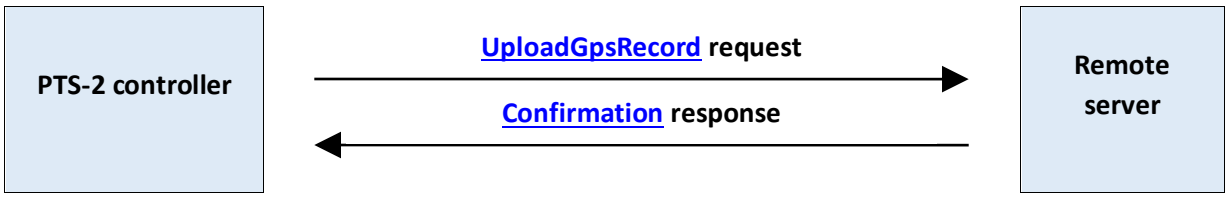


2. Uploading a measured tank data to a remote server – in this case PTS-2 controller is using UploadTankMeasurement request:

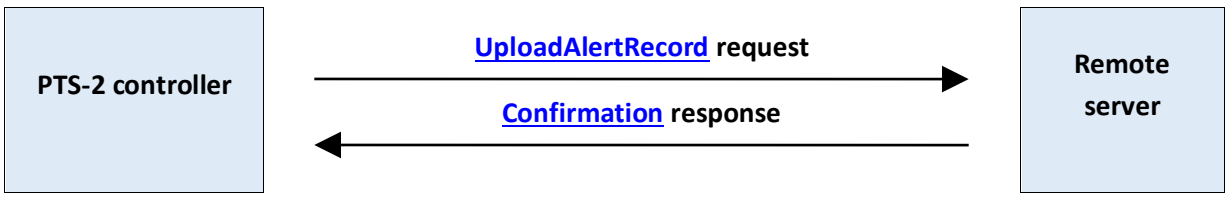


3. Uploading a registered in-tank delivery data to a remote server – in this case PTS-2 controller is using UploadInTankDelivery request:

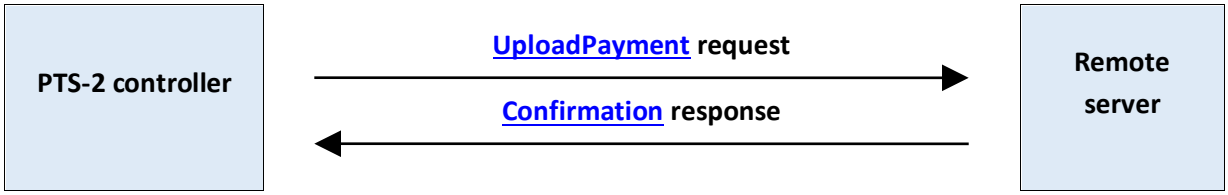
4. Uploading a registered GPS record to a remote server – in this case PTS-2 controller is using UploadGpsRecord request:



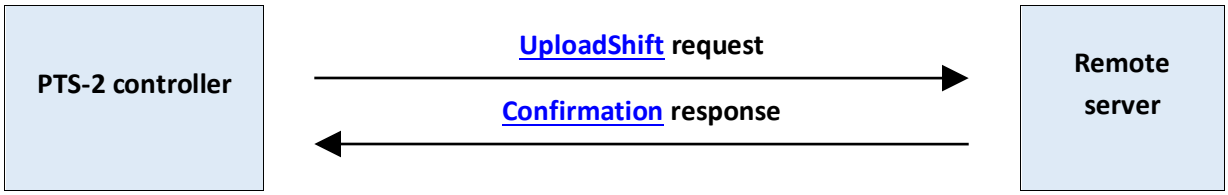
5. Uploading a registered alert record to a remote server – in this case PTS-2 controller is using UploadAlertRecord request:



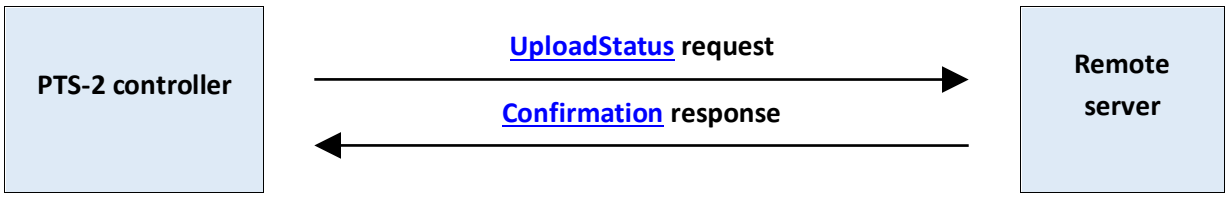
6. Uploading a registered payment record to a remote server – in this case PTS-2 controller is using UploadPayment request:



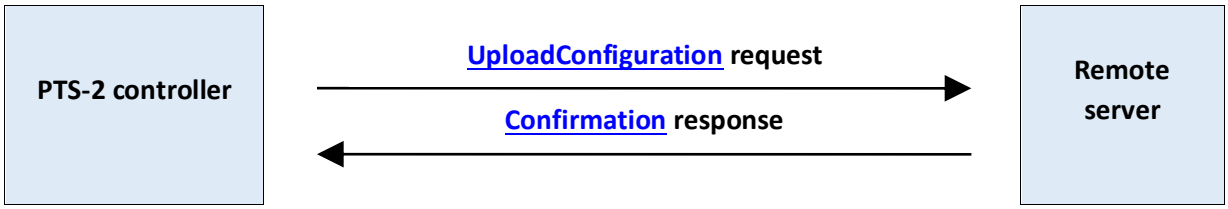
7. Uploading a registered working shift record to a remote server – in this case PTS-2 controller is using UploadShift request:



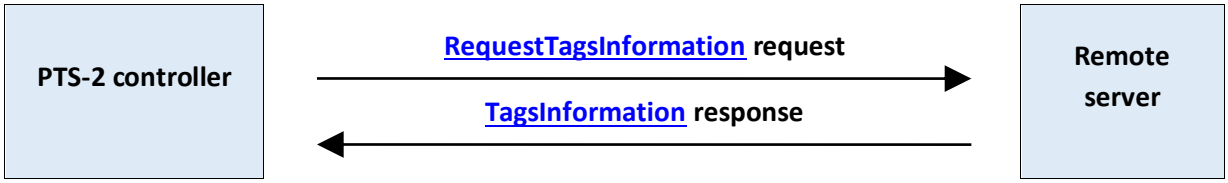
8. Uploading a realtime status of all connected forecourt equipment and status of the PTS-2 controller itself for realtime online monitoring – in this case PTS-2 controller is using UploadStatus request:



9. Uploading configuration to a remote server – in this case PTS-2 controller is using UploadConfiguration request:



10. Requesting tags accounts' balance from a remote server – in this case PTS-2 controller is using RequestTagsInformation request:



Each packet sent by the controller to server contains a unique ID (range from 1 to 65535). Response of the server should be in format of confirmation or error response message and should contain the same packet ID as received in the packet from the controller.

In case of UploadPumpTransaction, UploadTankMeasurement, UploadInTankDelivery, UploadGpsRecord, UploadAlertRecord, UploadPayment, UploadShift and UploadConfiguration requests the PTS-2 controller

will be sending same message to the remote server until it is accepted by the remote server, which means that the remote server should respond to it with a confirmation message.

UploadStatus request is used for periodical sending by the PTS-2 controller to the remote server for check of communication state and realtime online monitoring of the controller, which gives several solutions:

1. It indicates to the PTS-2 controller that the remote server is alive
2. It indicates to the remote server and PTS-2 controller is alive
3. It provides the remote server with online realtime data about the PTS-2 controller and forecourt equipment statuses (pumps, probes, price boards, readers, GPS receiver, others), so that the remote server can have online information on everything going with the PTS-2 controller

When "*UploadStatus*" option is enabled in configuration of remote server (using SetRemoteServerConfiguration request) – then the PTS-2 controller at absence of any other records to upload to the remote server keeps sending the UploadStatus request (time period between sending the request is configurable).

UploadStatus request can be initiated by the server by adding an additional optional field "*GetStatus*" with a Boolean value set to *true* to a confirmation response.

UploadConfiguration request is sent to the remote server with present complete configuration of the PTS-2 controller. It allows the remote server to know the present configuration of the PTS-2 controller and keep track of any changes in its configuration.

UploadConfiguration request, in case if it is enabled in configuration of the PTS-2 controller, is sent automatically to the remote server in case if there is any change in configuration of the PTS-2 controller. Also, UploadConfiguration request can be initiated by the server by adding an additional optional field "*GetConfiguration*" with a Boolean value set to *true* to a confirmation response.

RequestTagsInformation request is sent to the remote server to validate the tags and to check their accounts' balance remains. It can be used for the purpose of application of the tags for payment for the pump transactions.

In addition to abovementioned requests the remote server in confirmation response can add any request it wants to execute, check the Confirmation response with a single data setting request section for more information.

Each packet sent to server contains a unique identifier of PTS-2 controller in field "*PtsId*", which is a string (up to 24 hexadecimal digits), so that the remote server could easily identify it and distinguish from other PTS-2 controllers, which might also communicate with this remote server. Also, the same value is present in a HTTP header named "*X-Pts-Id"*. Also, the same value can be also received from PTS-2 controller using GetUniqueIdentifier request.

In each of the messages the value of the PTS-2 controller firmware date/time is sent in a header named "*X-Pts-Firmware-Version-DateTime*".

In each of the messages the value of the PTS-2 controller configuration identifier is sent in a header named "*X-Pts-Configuration-Identifier*", which is used to point if configuration was changed anyhow: given identifier is assigned a new unique value at any change of configuration, thus an application may know if the configuration was changed based on value of this identifier.

In case if value "*SecretKey*" has some value in configuration of remote server (using SetRemoteServerConfiguration request) – then an upload request is added with a message signature using HMAC SHA-256 mechanism in a HTTP header named "*X-Data-Signature"* in base64 encoding, which allows to ensure message authentication and integrity and check whether the message was sent by the PTS-2 controller. In this case a response from the server also should have the same HTTP header named "*X-Data-Signature"* to ensure authentication.

Example of request from the PTS-2 controller to the remote server (login is *admin*, password is *admin*, value of "*SecretKey*" is present in configuration of remote server and equals *secretKey*, so HTTP header named "*X-Data-Signature"* is present):

```
POST /jsonPTS HTTP/1.1\
Host: 192.168.1.246:54098
Authorization: Basic YWRtaW46YWRtaW4=
Content-Length: 101
Content-Type: application/json; charset=utf-8
X-Pts-Id: 0041001C524E500420323442
X-Pts-Firmware-Version-DateTime: 2021-11-12T08:05:41
X-Pts-Configuration-Identifier: b99cafbb
X-Data-Signature: 6UTdxAYVf9x2stuAfTWsfgKVo3BgL1uDe1t5eCmp7R0=
Connection: close

{"Protocol":"jsonPTS","PtsId":"0041001C524E500420323442","Packets":[
{"Id":5068,"Type":"UploadStatus"}]}
```

Response from the server should obligatorily contain HTTP code 200 or 201 and a Content-Length header.

Example of response from the remote server (value of "*SecretKey*" is present in configuration of remote server and equals *secretKey*, so HTTP header named "*X-Data-Signature"* is present):

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
Content-Length: 76
X-Data-Signature: zAZRWkBwRDLEfl0vsI7T6cpou/CAhUI7tlRa8vkVoVI=

{"Protocol":"jsonPTS","Packets":[{"Id":15999,"Error":false,"Message"
:"OK"}]}
```

REQUESTS TO A REMOTE SERVER AND RESPONSES FROM A REMOTE SERVER

200. UploadPumpTransaction request

Purpose	Uploads pump transaction to a remote server
Data	"*DateTimeStart*" – date and time of transaction start in format *YYYY-MM-DDThh:mm:ss* (string, 19 symbols), where: – *YYYY* – year (range from 2000 to 2099) – *MM* – month (range from 01 to 12) – *DD* – day (range from 01 to 31) – *T* – separator between date and time fields – *hh* – hours (range from 00 to 23) – *mm* – minutes (range from 00 to 59) – *ss* – seconds (range from 00 to 59) "*DateTime*" – date and time of transaction end in format *YYYY-MM-DDThh:mm:ss* (string, 19 symbols) "*Pump*" – logical number of the pump (integer, range from 1 to 120) "*Nozzle*" – nozzle number (integer, range from 1 to 6) "*Tank*" – logical number of the tank (integer, range from 1 to 20) "*FuelGradeId*" – identifier of fuel grade (integer, range from 1 to 20) "*FuelGradeName*" – name of fuel grade (string, up to 20 ASCII characters) "*Transaction*" – number of transaction (integer, range from 1 to 65535) "*Volume*" – transaction volume (float, up to 3 digits after decimal point) "*TCVolume*" – transaction temperature-compensated volume (float, up to 3 digits after decimal point) "*Price*" – product price (float, up to 3 digits after decimal point) "*Amount*" – transaction money amount (float, up to 3 digits after decimal point) "*TotalVolume*" – volume total counter value (float, up to 3 digits after decimal point) "*TotalAmount*" – money amount total counter value (float, up to 3 digits after decimal point) "*Tag*" – value of tag ID used to authorize the pump (string, up to 32 hexadecimal symbols) "*UserId*" – identifier of user, who authorized the transaction (integer, range from 1 to 30) "*IsOffline*" – flag showing that transaction was done in offline (manual) mode of the pump (Boolean, possible values: "*true*", "*false*") "*IsPaid*" – flag showing that transaction is paid (Boolean, possible values: "*true*", "*false*") "*ConfigurationId*" – unique configuration identifier (string, up to 8 hexadecimal digits)
Notes	1. Fields "*FuelGradeId*" and "*FuelGradeName*" are optional, they are present in response only when PTS-2 controller has pump nozzles configured to specific fuel grades. 2. Field "*ConfigurationId*" contains configuration unique identifier, which is used to point if configuration was changed anyhow: given identifier is assigned a new unique value at any change of configuration, thus an application may know if the configuration was changed based on value of this identifier. 3. If field "*Tag*" has empty value – it means that no tag identifier was used for pump

	authorization. 4. Field "Tank" is optional, it is present only if the pump nozzle is linked to tank in the configuration of the PTS-2 controller. 5. Fields "*IsOffline*" and "*IsPaid*" are optional, they present in response only in case if such values are present in the controller for the requested transaction.

Example	{
	"Protocol":"jsonPTS",
	"PtsId":"003A003A3435510938393730",
	"Packets":[{
	"Id":123,
	"Type":"UploadPumpTransaction",
	"Data":{
	"DateTimeStart":"2019-05-19T12:44:01",
	"DateTime":"2019-05-19T12:45:14",
	"Pump":1,
	"Nozzle":1,
	"FuelGradeId":1,
	"FuelGradeName":"Petrol",
	"Tank":3,
	"Transaction":15,
	"Volume":1.15,
	"TCVolume":1.07,
	"Price":10.00,
	"Amount":11.50,
	"TotalVolume":1231.15,
	"TotalAmount":45611.50,
	"Tag":"0123456789ABCDEF",
	"UserId":1,
	"ConfigurationId":"1234ABCD"
	}
	}]
	}

Response	Possible responses:
	– confirmation response message in case of successful reception
	– error response message in case remote server faces a problem with this message reception, in this case PTS-2 controller should send this message again to a remote server
	
	Example of response in case of successful operation:
	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":123,
	"Type":"UploadPumpTransaction",
	"Message":"OK"

```
    } ]
}
```

201. UploadTankMeasurement request

Purpose	Uploads tank measurement to a remote server
Data	"*DateTime*" – date and time of measurement in format *YYYY-MM-DDThh:mm:ss* (string, 19 symbols), where: – *YYYY* – year (range from 2000 to 2099) – *MM* – month (range from 01 to 12) – *DD* – day (range from 01 to 31) – *T* – separator between date and time fields – *hh* – hours (range from 00 to 23) – *mm* – minutes (range from 00 to 59) – *ss* – seconds (range from 00 to 59) "*Tank*" – logical number of the tank (integer, range from 1 to 20) "*FuelGradeId*" – identifier of fuel grade (integer, range from 1 to 20) "*FuelGradeName*" – name of fuel grade (string, up to 20 ASCII characters) "*Status*" – status of probe (string, up to 5 ASCII symbols), possible values: – "*OK*": in case of correct operation – "*Error*": in case of incorrect operation "*Alarms*" – array of strings indicating probe alarms for tank, possible values: – "*CriticalHighProduct*": if product level is higher than maximum critical – "*HighProduct*": if product level is higher than maximum allowed – "*LowProduct*": if product level is lower than minimum allowed – "*CriticalLowProduct*": if product level is lower than minimum critical – "*HighWater*": in case if water level is higher than maximum allowed – "*TankLeakage*": in case if tank leakage is detected "*ProductHeight*" – value of product height in mm (float, up to 1 digit after decimal point) "*WaterHeight*" – value of water height in mm (float, up to 1 digit after decimal point) "*Temperature*" – value product temperature in degrees Celcius (float, up to 1 digit after decimal point) "*ProductVolume*" – value of product volume in liters (integer) "*WaterVolume*" – value of water volume in liters (integer) "*ProductUllage*" – value of product ullage in liters (integer) "*ProductTCVolume*" – value of product temperature compensated volume in liters (integer) "*ProductDensity*" – value of product density in kg/m^3 units (float, up to 1 digit after decimal point) "*ProductMass*" – value of product mass in kg (float, up to 1 digit after decimal point) "*TankFillingPercentage*" – value of tank filling percentage (integer) "*ConfigurationId*" – unique configuration identifier (string, up to 8 hexadecimal digits)
Notes	1. This request contains only values measured by probe and calculated by the controller, so some of the values may not be present in this request. 2. Array "*Alarms*" is optional and is present in response only in case if there are alarms present. 3. Field "*TankFillingPercentage*" is calculated by the controller the following way: – as ratio of values of (filled product volume + filled water volume) and (filled

product volume + filled water volume + ullage), if these values are present
- as ratio of values of filled product height and full tank height, if these values are present

4. Field "*ConfigurationId*" contains configuration unique identifier, which is used to point if configuration was changed anyhow: given identifier is assigned a new unique value at any change of configuration, thus an application may know if the configuration was changed based on value of this identifier.
5. Fields "*FuelGradeId*" and "*FuelGradeName*" are optional, they are present in response only when PTS-2 controller has tank configured to specific fuel grade.

Example

```
{
  "Protocol":"jsonPTS",
  "PtsId":"003A003A3435510938393730",
  "Packets":[{
    "Id":124,
    "Type":"UploadTankMeasurement",
    "Data":{
      "DateTime":"2019-05-19T12:45:14",
      "Tank":1,
      "Status":"OK",
      "Alarms":[
        "HighProduct"
      ],
      "ProductHeight":2534.1,
      "WaterHeight":123.1,
      "Temperature":21.3,
      "ProductVolume":200,
      "WaterVolume":15,
      "ProductUllage":850,
      "ProductTCVolume":195,
      "ProductDensity":0.76,
      "ProductMass":190,
      "ConfigurationId":"1234ABCD"
    }
  }]
}
```

Response

Possible responses:
- confirmation response message in case of successful reception
- error response message in case remote server faces a problem with this message reception, in this case PTS-2 controller should send this message again to a remote server

Example of response in case of successful operation:

```
{
  "Protocol":"jsonPTS",
  "Packets":[{
```

```
    "Id":124,
    "Type":"UploadTankMeasurement",
    "Message":"OK"
  }]
}
```

202. UploadInTankDelivery request

Purpose	Uploads in-tank delivery to a remote server
Data	"*Tank*" – logical number of the tank (integer, range from 1 to 20), value 0 means all tanks "*FuelGradeId*" – identifier of fuel grade (integer, range from 1 to 20) "*FuelGradeName*" – name of fuel grade (string, up to 20 ASCII characters) "*StartValues*" – object describing values on start of the in-tank delivery, contains following elements: – "*DateTime*" – date and time of last in-tank delivery start in format *YYYY-MM-DDThh:mm:ss* (string, 19 symbols), where: • *YYYY* – year (range from 2000 to 2099) • *MM – month (range from 01 to 12)* • *DD – day (range from 01 to 31)* • *T – separator between date and time fields* • *hh – hours (range from 00 to 23)* • *mm – minutes (range from 00 to 59)* • *ss – seconds (range from 00 to 59)* – "*ProductHeight*" – value of product height in mm (float, up to 1 digit after decimal point) – "WaterHeight" – value of water height in mm (float, up to 1 digit after decimal point) – "*Temperature*" – value of temperature in degrees Celcius (float, up to 1 digit after decimal point) – "*ProductVolume*" – value of product volume in liters (integer) – "*ProductTCVolume*" – value of product temperature compensated volume in liters (integer) – "*ProductDensity*" – value of product density in kg/m3 units (float, up to 1 digit after decimal point) – "*ProductMass*" – value of product mass in kg (float, up to 1 digit after decimal point) "*EndValues*" – object describing values on end of the in-tank delivery, contains following elements: – "*DateTime*" – date and time of last in-tank delivery end in format *YYYY-MM-DDThh:mm:ss* (string, 19 symbols), where: ▪ *YYYY* – year (range from 2000 to 2099) ▪ *MM* – month (range from 01 to 12) ▪ *DD* – day (range from 01 to 31) ▪ *T* – separator between date and time fields ▪ *hh* – hours (range from 00 to 23) ▪ *mm* – minutes (range from 00 to 59) ▪ *ss* – seconds (range from 00 to 59) – *"ProductHeight"* – value of product height in mm (float, up to 1 digit after decimal point) – *"WaterHeight"* – value of water height in mm (float, up to 1 digit after decimal point) – *"Temperature"* – value of temperature in degrees Celcius (float, up to 1 digit after decimal point)

	– *"ProductVolume"* – value of product volume in liters (integer) – *"ProductTCVolume"* – value of product temperature compensated volume in liters *(integer)* – *"ProductDensity"* – value of product density in kg/m3 units (float, up to 1 digit after decimal point) – *"ProductMass*" – value of product mass in kg (float, up to 1 digit after decimal point) "*AbsoluteValues*" – object describing absolute values of the in-tank delivery, contains following elements: – "*ProductHeight"* – value of product height in mm (float, up to 1 digit after decimal point) – *"WaterHeight"* – value of water height in mm (float, up to 1 digit after decimal point) – *"Temperature"* – value of temperature in degrees Celcius (float, up to 1 digit after decimal point) – *"ProductVolume"* – value of product volume in liters (integer) – *"ProductTCVolume"* – value of product temperature compensated volume in liters (integer) – *"ProductDensity"* – value of product density in kg/m3 units (float, up to 1 digit after decimal point) – *"ProductMass"* – value of product mass in kg (float, up to 1 digit after decimal point) – *"PumpsDispensedVolume*" – value of product volume dispensed through pumps during delivery in liters (float, up to 3 digits after decimal point) "*ConfigurationId*" – unique configuration identifier (string, up to 8 hexadecimal digits)
Notes	1. This request contains only values measured by probe and calculated by the controller, so some of the values may not be present in this request. 2. Field "*ConfigurationId*" contains configuration unique identifier, which is used to point if configuration was changed anyhow: given identifier is assigned a new unique value at any change of configuration, thus an application may know if the configuration was changed based on value of this identifier. 3. Fields *"FuelGradeId"* and *"FuelGradeName"* are optional, they are present in response only when PTS-2 controller has tank configured to specific fuel grade.

Example	{
	"Protocol":"jsonPTS",
	"PtsId":"003A003A3435510938393730",
	"Packets":[{
	"Id":124,
	"Type":"UploadInTankDelivery",
	"Data":{
	"Tank":1,
	"StartValues": {
	"DateTime": "2019-03-09T15:30:25",
	"ProductHeight": 2534.1,
	"WaterHeight": 123.1,
	"Temperature": 21.3,
	"ProductVolume": 2005,

```
          "ProductTCVolume": 2015
        },
        "EndValues": {
          "DateTime": "2019-03-09T15:32:15",
          "ProductHeight": 2725.1,
          "WaterHeight": 130.4,
          "Temperature": 21.6,
          "ProductVolume": 2514,
          "ProductTCVolume": 2601
        },
        "AbsoluteValues": {
          "ProductDensity": 758.9,
          "ProductMass": 386.3,
          "PumpsDispensedVolume": 25.3
        },
        "ConfigurationId":"1234ABCD"
      }
    }]
}
```


Response

Possible responses:

- confirmation response message in case of successful reception
- error response message in case remote server faces a problem with this message reception, in this case PTS-2 controller should send this message again to a remote server

Example of response in case of successful operation:


```
{
  "Protocol":"jsonPTS",
  "Packets":[{
    "Id":124,
    "Type":"UploadInTankDelivery",
    "Message":"OK"
  }]
}
```

203. UploadGpsRecord request

Purpose	Uploads GPS record to a remote server
Data	"*DateTime*" – date and time of GPS record in format *YYYY-MM-DDThh:mm:ss* (string, 19 symbols), where: – *YYYY* – year (range from 2000 to 2099) – *MM* – month (range from 01 to 12) – *DD* – day (range from 01 to 31) – *T* – separator between date and time fields – *hh* – hours (range from 00 to 23) – *mm* – minutes (range from 00 to 59) – *ss* – seconds (range from 00 to 59) "*Latitude*" – latitude in format *ddmm.mmmm* (string, 9 symbols), where: – *dd* – degrees (range from 0 to 99) – *mm.mmmm* – minutes (range from 0.0000 to 99.9999) "*NorthSouthIndicator*" – north/south indicator (string), possible values: '*N*' (for North) and '*S*' (for South) "*Longitude*" – latitude in format *dddmm.mmmm* (string, 10 symbols), where: – *ddd* – degrees (range from 0 to 999) – *mm.mmmm* – minutes (range from 0.0000 to 99.9999) "*EastWestIndicator*" – east/west indicator (string), possible values: '*E*' (for East) and '*W*' (for West) "*SpeedOverGround*" – speed over ground in kilometers per hour (float, up to 2 digits after decimal point) "*CourseOverGround*" – course over ground in degrees (float, up to 2 digits after decimal point) "*Mode*" – operation mode (string), possible values: '*A*' (for Autonomous) and '*D*' (for DGPS) "*ConfigurationId*" – unique configuration identifier (string, up to 8 hexadecimal digits)
Notes	Field "*ConfigurationId*" contains configuration unique identifier, which is used to point if configuration was changed anyhow: given identifier is assigned a new unique value at any change of configuration, thus an application may know if the configuration was changed based on value of this identifier.

Example	{
	"Protocol":"jsonPTS",
	"PtsId":"003A003A3435510938393730",
	"Packets":[{
	"Id":125,
	"Type": "UploadGpsRecord",
	"Data":{
	"DateTime":"2021-05-19T12:45:14",
	"Latitude":"3113.3156",
	"NorthSouthIndicator":"N",
	"Longitude":"12121.2686",
	"EastWestIndicator":"E",
	"SpeedOverGround":55.73,

<table>
<tr><td></td><td>

```
        "CourseOverGround":193.93,
        "Mode":"A",
        "ConfigurationId":"1234ABCD"
      }
   }]
}
```

</td></tr>
<tr><td>Response</td><td>
Possible responses:

confirmation response message in case of successful reception
error response message in case remote server faces a problem with this message reception, in this case PTS-2 controller should send this message again to a remote server

Example of response in case of successful operation:

```
{
  "Protocol":"jsonPTS",
  "Packets":[{
    "Id":125,
    "Type":"UploadGpsRecord",
    "Message":"OK"
  }]
}
```

</td></tr>
</table>

204. UploadAlertRecord request

Purpose	Uploads alert record to a remote server
Data	"*DateTime*" – date and time in format *YYYY-MM-DDThh:mm:ss* (string, 19 symbols), where: – *YYYY* – year (range from 2000 to 2099) – *MM* – month (range from 01 to 12) – *DD* – day (range from 01 to 31) – *T* – separator between date and time fields – *hh* – hours (range from 00 to 23) – *mm* – minutes (range from 00 to 59) – *ss* – seconds (range from 00 to 59) "*DeviceType*" – strings indicating device type, possible values: – "*PTS*": if alert is related to the PTS-2 controller – "*Pump*": if alert is related to pump – "*Probe*": if alert is related to probe – "*PriceBoard*": if alert is related to price board – "*Reader*": if alert is related to reader "*DeviceNumber*" – number of device, to which the alert relate (integer, range from 1 to 120) "*State*" – strings indicating alert state, possible values: – "*Started*" – in case if the alert is started and has duration – "*Finished*" – in case if the alert is finished and had duration – *"Detected"* – in case if the alert is detected and does not have duration "*Code*" –alert code (integer, range from 1 to 99999) "*ConfigurationId*" – unique configuration identifier (string, up to 8 hexadecimal digits)
Notes	1. Values of alert codes is given in section ALERT CODES 2. Field "*ConfigurationId*" contains configuration unique identifier, which is used to point if configuration was changed anyhow: given identifier is assigned a new unique value at any change of configuration, thus an application may know if the configuration was changed based on value of this identifier.

Example	{
	"Protocol": "jsonPTS",
	"PtsId": "0054002E524E500420323442",
	"Packets": [{
	"Id": 63609,
	"Type": "UploadAlertRecord",
	"Data": {
	"DateTime": "2022-09-24T21:46:47",
	"DeviceType": "PTS",
	"DeviceNumber": 0,
	"State": "Started",
	"Code": 1,
	"ConfigurationId": "c8ebb368"
	}

	<pre>}]
	}</pre>

Response	Possible responses:
	– confirmation response message in case of successful reception
	– error response message in case remote server faces a problem with this message reception, in this case PTS-2 controller should send this message again to a remote server
	
	Example of response in case of successful operation:
	<pre>{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":63609,
	"Type":"UploadAlertRecord",
	"Message":"OK"
	}]
	}</pre>

205. UploadPayment request

Purpose	Uploads a payment record to a remote server
Data	"*PaymentTransactionDateTime*" – date and time of payment transaction in format *YYYY-MM-DDThh:mm:ss* (string, 19 symbols), where: – *YYYY* – year (range from 2000 to 2099) – *MM* – month (range from 01 to 12) – *DD* – day (range from 01 to 31) – *T* – separator between date and time fields – hh – hours (range from 00 to 23) – *mm* – minutes (range from 00 to 59) – *ss* – seconds (range from 00 to 59) "*PaymentTransaction*" – number of payment transaction (integer, range from 1 to 999999999) "*Pump*" – logical number of the pump (integer, range from 1 to 120) "*PumpTransaction*" – number of pump transaction (integer, range from 1 to 65535) "*PumpTransactionDateTime*" – date and time of pump transaction end in format *YYYY-MM-DDThh:mm:ss* (string, 19 symbols), where: – *YYYY* – year (range from 2000 to 2099) – *MM* – month (range from 01 to 12) – *DD* – day (range from 01 to 31) – *T* – separator between date and time fields – hh – hours (range from 00 to 23) – *mm* – minutes (range from 00 to 59) – *ss* – seconds (range from 00 to 59) *"Amount"* – payment amount (float, up to 3 digits after decimal point) *"PaymentFormId"* – identifier of payment form (integer, range from 0 to 10) "*PaymentFormName*" – name of payment form (string, up to 10 ASCII characters) "*IsRefunded*" - flag showing that the transaction was refunded (Boolean, possible values: "*true*", "*false*") "*ConfigurationId*" – unique configuration identifier (string, up to 8 hexadecimal digits)
Note	Field "*ConfigurationId*" contains configuration unique identifier, which is used to point if configuration was changed anyhow: given identifier is assigned a new unique value at any change of configuration, thus an application may know if the configuration was changed based on value of this identifier.

Example	{
	"Protocol":"jsonPTS",
	"PtsId": "0054002E524E500420323442",
	"Packets":[{
	"Id":12345,
	"Type":"UploadPayment",
	"Data":{
	"PaymentTransactionDateTime":"2025-07-21T12:34:56",
	"PaymentTransaction":15,
	"Pump":1,
	"PumpTransaction":155,

```
        "PumpTransactionDateTime":"2025-07-21T11:22:33",
        "Amount":1.14,
        "PaymentFormId":1,
        "PaymentFormName":"Cash",
        "IsRefunded":false,
        "ConfigurationId": "c8ebb368"
      }
    }]
}
```


Response

Possible responses:

- confirmation response message in case of successful reception
- error response message in case remote server faces a problem with this message reception, in this case PTS-2 controller should send this message again to a remote server

Example of response in case of successful operation:


```
{
  "Protocol":"jsonPTS",
  "Packets":[{
    "Id":12345,
    "Type":"UploadPayment",
    "Message":"OK"
  }]
}
```

206. UploadShift request

Purpose	Uploads a working shift record to a remote server
Data	"*Number*" – working shift sequence number (integer, range from 1 to 999999999) "*DateTimeStart*" – date and time of transaction start in format *YYYY-MM-DDThh:mm:ss* (string, 19 symbols), where: – *YYYY* – year (range from 2000 to 2099) – *MM* – month (range from 01 to 12) – *DD* – day (range from 01 to 31) – *T* – separator between date and time fields – hh – hours (range from 00 to 23) – *mm* – minutes (range from 00 to 59) – *ss* – seconds (range from 00 to 59) "*DateTimeEnd*" – date and time of transaction end in format *YYYY-MM-DDThh:mm:ss* (string, 19 symbols), where: – *YYYY* – year (range from 2000 to 2099) – *MM* – month (range from 01 to 12) – *DD* – day (range from 01 to 31) – *T* – separator between date and time fields – hh – hours (range from 00 to 23) – *mm* – minutes (range from 00 to 59) – *ss* – seconds (range from 00 to 59) "*UserId*" – identifier of user, who opened the working shift (integer, range from 1 to 30)
Note	1. If field "*DateTimeEnd*" is absent – then the shift is opened and not closed yet. 2. Field "*ConfigurationId*" contains configuration unique identifier, which is used to point if configuration was changed anyhow: given identifier is assigned a new unique value at any change of configuration, thus an application may know if the configuration was changed based on value of this identifier.

Example	{
	"Protocol":"jsonPTS",
	"PtsId": "0054002E524E500420323442",
	"Packets":[{
	"Id":1379,
	"Type":"UploadShift",
	"Data":{
	"DateTimeStart":"2025-07-21T12:34:56",
	"DateTimeEnd":"2025-07-22T12:00:00",
	"Number":152,
	"UserId":1,
	"ConfigurationId": "c8ebb368"
	}
	}]
	}

Response	Possible responses:

	– confirmation response message in case of successful reception
	– error response message in case remote server faces a problem with this message reception, in this case PTS-2 controller should send this message again to a remote server
	
	Example of response in case of successful operation:
	`{`
	`  "Protocol":"jsonPTS",`
	`  "Packets":[{`
	`  "Id":1379,`
	`  "Type":"UploadShift",`
	`  "Message":"OK"`
	`  }]`
	`}`

207. UploadConfiguration request

Purpose	Sends the PTS-2 controller configuration to a remote server. Using it the remote server can know the present configuration of the PTS-2 controller.
Data	"*ConfigurationId*" – unique configuration identifier (string, up to 8 hexadecimal digits) "*Configuration*" - array of objects for each response packets used for configuration in the PTS-2 controller provided by this communication protocol description. The response packets come without "*Id*" fields.
Notes	1. If the "*UploadConfiguration*" field is enabled in configuration of the remote server settings using SetRemoteServerConfiguration request and there is a configuration change in the PTS-2 controller – then the PTS-2 controller automatically uploads a new configuration to the server once. 2. UploadConfiguration request can be initiated by the server by adding an additional optional field "*GetConfiguration*" with a Boolean value set to *true* to a confirmation response. Example of confirmation response for UploadStatus request with request for configuration: `{ "Protocol":"jsonPTS", "Packets":[{ "Id":127, "Type":"UploadStatus", "Message":"OK", "GetConfiguration":true }] }` 3. The "*UploadConfiguration*" request is sent without a signature in "*X-Data-Signature"* header even if "*SecretKey*" has some value in configuration of remote server. 4. Field "*ConfigurationId*" contains configuration unique identifier, which is used to point if configuration was changed anyhow: given identifier is assigned a new unique value at any change of configuration, thus an application may know if the configuration was changed based on value of this identifier.
Example	`{ "Protocol":"jsonPTS", "PtsId":"003A003A3435510938393730", "Packets":[{ "Id":126, "Type":"UploadConfiguration", "ConfigurationId": "3751ade7", "Configuration":[{"Type":"PtsNetworkSettings", "Data":{"IpAddress":[192,168,1,117], "NetMask":[255,255,255,0], "Gateway":[192,168,1,13], "HttpPort":80, "HttpsPort":443, "Dns1":[8,8,8,8], "Dns2":[8,8,4,4]}}, {"Type":"PumpsConfiguration", "Data":{"Ports":[{"Id":1, "Protocol":5, "BaudRate":3}], "Pumps":[{"Id":1, "Port":1, "Address":1},`

<table>
<tr><td></td><td>

```
        {"Id":2, "Port":1, "Address":2}]}},
        ... (other packets)
      ]
   }]
}
```

Real-life example of UploadConfiguration request can be found here:
https://www.technotrade.ua/Uploads/Files/PTS-2/UploadConfiguration_example.json
</td></tr>
<tr><td>Response</td><td>

Possible responses:

- confirmation response message in case of successful reception
- error response message in case remote server faces a problem with this message reception, in this case PTS-2 controller should send this message again to a remote server

Example of response in case of successful operation:

```
{
  "Protocol":"jsonPTS",
  "Packets":[{
    "Id":126,
    "Type":"UploadConfiguration",
    "Message": "OK"
  }]
}
```
</td></tr>
</table>

208. UploadStatus request

Purpose	Sends status message for check of communication with the server and realtime monitoring of the controller and forecourt equipment.
Data	"*ConfigurationId*" – unique configuration identifier (string, up to 8 hexadecimal digits) "*DateTime*" – present date and time in the controller in format *YYYY-MM-DDThh:mm:ss* (string, 19 symbols), where: – *YYYY* – year (range from 2000 to 2099) – *MM* – month (range from 01 to 12) – *DD* – day (range from 01 to 31) – *T* – separator between date and time fields – *hh* – hours (range from 00 to 23) – *mm* – minutes (range from 00 to 59) – *ss* – seconds (range from 00 to 59) "*FirmwareDateTime*" – firmware release date and time in format *YYYY-MM-DDThh:mm:ss* (string, 19 symbols), where: – *YYYY* – year (range from 2000 to 2099) – *MM* – month (range from 01 to 12) – *DD* – day (range from 01 to 31) – *T* – separator between date and time fields – *hh* – hours (range from 00 to 23) – *mm* – minutes (range from 00 to 59) – *ss* – seconds (range from 00 to 59) "*StartupSeconds*" – number of seconds after the controller started operation (integer) "*BatteryVoltage*" – value of battery voltage in millivolts (mV) DC (integer) "*CpuTemperature*" – value of CPU temperature in degrees Celsius (°C) (integer) "*PtsPowerDownDetected*" – flag showing if controller power down is detected (Boolean, possible values: "*true*", "*false*") "*SdMounted*" – flag showing if controller has detected and mounted the SD flash disk (Boolean, possible values: "*true*", "*false*") "*ShiftInformation*" – object describing presently running working shift with the following elements: – "*Enabled*" – flag showing if leading of working shift is enabled (Boolean, possible values: "*true*", "*false*") – "*Number*" – working shift sequence number (integer, range from 1 to 999999999) – "*IsOpened*" - flag showing if the working shift is presently opened (Boolean, possible values: "*true*", "*false*") – "*DateTimeStart*" – date and time of the working shift start in format *YYYY-MM-DDThh:mm:ss* (string, 19 symbols), where: • *YYYY* – year (range from 2000 to 2099) • *MM* – month (range from 01 to 12) • *DD* – day (range from 01 to 31) • *T* – separator between date and time fields • *hh* – hours (range from 00 to 23) • *mm* – minutes (range from 00 to 59) • *ss* – seconds (range from 00 to 59)

- "*DateTimeEnd*" – date and time of the working shift end in format *YYYY-MM-DDThh:mm:ss* (string, 19 symbols), where:
  - *YYYY* – year (range from 2000 to 2099)
  - *MM* – month (range from 01 to 12)
  - *DD* – day (range from 01 to 31)
  - *T* – separator between date and time fields
  - *hh* – hours (range from 00 to 23)
  - *mm* – minutes (range from 00 to 59)
  - *ss* – seconds (range from 00 to 59)
- "*UserId*" – identifier of user, who opened the working shift (integer, range from 1 to 30)

"*Pumps*" – object for pumps with the following elements:

- "*IdleStatus*" – object for pumps having filling status with the following elements:
  - "*Ids*" – array of integers, which mean pumps' identifiers having filling status (each element is integer, range from 1 to 120)
  - "*NozzlesUp*" – array of integers, which mean pumps' nozzle numbers taken up on pumps (each element is integer, range from 1 to 6)
  - "*LastNozzles*" – array of integers, which mean last pumps' transaction nozzles (each element is integer, range from 1 to 6)
  - "*LastTransactions*" – array of integers, which mean last pumps' transaction numbers (each element is integer, range from 1 to 65535)
  - "*LastVolumes*" – array of floats, which mean last pumps' transaction dispensed volumes (each element is float, up to 3 digits after decimal point)
  - "*LastAmounts*" – array of floats, which mean last pumps' transaction dispensed amounts (each element is float, up to 3 digits after decimal point)
  - "*LastPrices*" – array of floats, which mean last pumps' transaction product price (each element is float, up to 3 digits after decimal point)
  - "*LastFlowRates*" - value of the last filling flow rate in volume units per minute (integer, range from 0 till 9999)
  - "*LastTotalVolumes*" – array of floats, which mean last pumps' volume total counters (each element is float, up to 3 digits after decimal point)
  - "*LastTotalAmounts*" – array of floats, which mean last pumps' amount total counters (each element is float, up to 3 digits after decimal point)
  - "*Tags*" – array of strings, which mean tag identifiers currently brought to pump reader (string, up to 32 hexadecimal symbols)
  - "*Requests*" – array of strings, which mean presently executed requests on pumps (each element is string, up to 20 ASCII symbols)
- "*FillingStatus*" – object for pumps having filling status with the following elements:
  - "*Ids*" – array of integers, which mean pumps' identifiers having filling status (each element is integer, range from 1 to 120)
  - "*Nozzles*" – array of integers, which mean pumps' active nozzles number (each element is integer, range from 1 to 6)
  - "*FuelGradeIds*" – array of integers, which mean pumps' active nozzles' identifier of fuel grade (each element is integer, range from 1 to 20)
  - "*FuelGradeNames*" – array of strings, which mean names of pumps' active nozzles' fuel grade (each element is string, up to 20 ASCII characters)
  - "*Transactions*" – array of integers, which mean pumps' active transaction

numbers (each element is integer, range from 1 to 65535)
- "*Volumes*" – array of floats, which mean pumps' dispensed volumes (each element is float, up to 3 digits after decimal point)
- "*Amounts*" – array of floats, which mean pumps' dispensed amounts (each element is float, up to 3 digits after decimal point)
- "*Prices*" – array of floats, which mean pumps' active product price (each element is float, up to 3 digits after decimal point)
- "*FlowRates*" - value of the filling flow rate in volume units per minute (integer, range from 0 till 9999)
- "*Tags*" – array of strings, which mean tag identifiers used to authorize the pump (string, up to 32 hexadecimal symbols)

– "*EndOfTransactionStatus*" – object for pumps having end of transaction status with the following elements:
- "*Ids*" – array of integers, which mean pumps' identifiers having filling status (each element is integer, range from 1 to 120)
- "*Nozzles*" – array of integers, which mean pumps' active nozzles number (each element is integer, range from 1 to 6)
- "*FuelGradeIds*" – array of integers, which mean pumps' active nozzles' identifier of fuel grade (each element is integer, range from 1 to 20)
- "*FuelGradeNames*" – array of strings, which mean names of pumps' active nozzles' fuel grade (each element is string, up to 20 ASCII characters)
- "*Transactions*" – array of integers, which mean pumps' active transaction numbers (each element is integer, range from 1 to 65535)
- "*Volumes*" – array of floats, which mean pumps' dispensed volumes (each element is float, up to 3 digits after decimal point)
- "*Amounts*" – array of floats, which mean pumps' dispensed amounts (each element is float, up to 3 digits after decimal point)
- "*Prices*" – array of floats, which mean pumps' active product price (each element is float, up to 3 digits after decimal point)
- "*FlowRates*" - value of the filling flow rate in volume units per minute (integer, range from 0 till 9999)
- "*Tags*" – array of strings, which mean tag identifiers used to authorize the pump (string, up to 32 hexadecimal symbols)

– "*OfflineStatus*" – object for pumps having offline status with the following elements:
- "*Ids*" – array of integers, which mean pumps' identifiers having offline status (each element is integer, range from 1 to 120)

– "*Users*" – array of strings, which mean users, currently executing a request on pumps (each element is string, up to 10 ASCII symbols)

"*Probes*" – object for probes with the following elements:

– "*OnlineStatus*" – object for probes having online status with the following elements:
- "*Ids*" – array of integers, which mean probes' identifiers having online status (each element is integer, range from 1 to 20)
- "*CriticalHighProductAlarms*" – array of integers, which mean probes' identifiers with critical high product alarms for tank (each element is integer, presence of alarm is 1, absence of alarm is 0)
- "*HighProductAlarms*" – array of integers, which mean probes' identifiers with high product alarms for tank (each element is integer, presence of alarm is 1,

absence of alarm is 0)

- "*LowProductAlarms*" – array of integers, which mean probe's identifiers with low product alarms for tank (each element is integer, presence of alarm is 1, absence of alarm is 0)
- "*CriticalLowProductAlarms*" – array of integers, which mean probe's identifiers with critical low product alarms for tank (each element is integer, presence of alarm is 1, absence of alarm is 0)
- "*HighWaterAlarms*" – array of integers, which mean probe's identifiers with high water alarms for tank (each element is integer, presence of alarm is 1, absence of alarm is 0)
- "*TankLeakageAlarms*" – array of integers, which mean probe's identifiers with leakage alarms for tank (each element is integer, presence of alarm is 1, absence of alarm is 0)
- "*Errors*" – array of integers, which mean probe's identifiers with error for probes (each element is integer, presence of alarm is 1, absence of alarm is 0)
- "*Measurements*" – array of arrays each containing measurements with following elements (absent measurements are filled with null values):
  - probe number (integer, range from 1 to 20)
  - value of probe's product height in mm (each element is float, up to 1 digit after decimal point) or null (if no value)
  - value of probe's water height in mm (each element is float, up to 1 digit after decimal point) or null (if no value)
  - value of probe's temperature in degrees Celcius (each element is float, up to 1 digit after decimal point) or null (if no value)
  - value of probe's product volume in liters (each element is integer) or null (if no value)
  - value of probe's water volume in liters (each element is integer) or null (if no value)
  - value of probe's product ullage in liters (each element is integer) or null (if no value)
  - value of probe's product temperature compensated volume in liters (each element is integer) or null (if no value)
  - value of probe's product density in kg/m^3 units (each element is float, up to 1 digit after decimal point) or null (if no value)
  - value of probe product mass in kg (each element is float, up to 1 digit after decimal point) or null (if no value)
  - value of tank filling percentage (each element is integer) or null (if no value)

– "*OfflineStatus*" – object for probes having offline status with the following elements:

- "*Ids*" – array of integers, which mean probes' identifiers having online status (each element is integer, range from 1 to 20)

"*PriceBoards*" – object for price boards with the following elements:

– "*OnlineStatus*" – object for price boards having online status with the following elements:

- "*Ids*" – array of integers, which mean price boards' identifiers having online status (each element is integer, range from 1 to 5)
- "*Errors*" – array of integers, which mean price boards' identifiers having error status (each element is integer, range from 1 to 5)

- "*OfflineStatus*" – object for price boards having online status with the following elements:
  - "*Ids*" – array of integers, which mean price boards' identifiers having online status (each element is integer, range from 1 to 5)

"*Readers*" – object for readers with the following elements:

- "*OnlineStatus*" – object for readers having online status with the following elements:
  - "*Ids*" – array of integers, which mean readers' identifiers having online status (each element is integer, range from 1 to 120)
  - "*Tags*" – array of strings, which mean readers' tags' IDs (each element is string, up to 32 hexadecimal symbols)
  - "*Errors*" – array of integers, which mean readers' identifiers having error status (each element is integer, range from 1 to 5)
- "*OfflineStatus*" – object for readers having online status with the following elements:
  - "*Ids*" – array of integers, which mean readers' identifiers having online status (each element is integer, range from 1 to 120)

"*Gps*" – object with online GPS data with the following elements (field present if GPS module is present and enabled in the controller):

- "*Status*" – status of GPS data (string), possible values: '*Valid'*, '*Invalid', "Absent"*
- "*DateTime*" – date and time in format *YYYY-MM-DDThh:mm:ss* (string, 19 symbols), where:
  - *YYYY* – year (range from 2000 to 2099)
  - *MM* – month (range from 01 to 12)
  - *DD* – day (range from 01 to 31)
  - *T* – separator between date and time fields
  - *hh* – hours (range from 00 to 23)
  - *mm* – minutes (range from 00 to 59)
  - *ss* – seconds (range from 00 to 59)
- "*Latitude*" – latitude in format *ddmm.mmmm* (string, 9 symbols), where:
  - *dd* – degrees (range from 0 to 99)
  - *mm.mmmm* – minutes (range from 0.0000 to 99.9999)
- "*NorthSouthIndicator*" – north/south indicator (string), possible values: '*North'* and '*South'*
- "*Longitude*" – latitude in format *dddmm.mmmm* (string, 10 symbols), where:
  - *ddd* – degrees (range from 0 to 999)
  - *mm.mmmm* – minutes (range from 0.0000 to 99.9999)
- "*EastWestIndicator*" – east/west indicator (string), possible values: '*East*' and '*West'*
- "*SpeedOverGround*" – speed over ground in kilometers per hour (float, up to 2 digits after decimal point)
- "*CourseOverGround*" – course over ground in degrees (float, up to 2 digits after decimal point)
- "*Mode*" – operation mode (string), possible values: '*Autonomous*' and '*DGPS'*

"*FuelGrades*" – array of objects for configuration of fuel grades, elements of each object:

- "*Id*" – identifier of fuel grade (integer, range from 1 to 10)

	– "*Name*" – name of fuel grade (string, up to 20 ASCII characters) – "*Price*" – price of fuel grade (float, up to 9 digits with dot as a decimal separator, value without a decimal separator is considered to have no decimal digits) – "*ExpansionCoefficient*" – thermal coefficient of expansion at 15 °C for fuel grade (float, up to 5 digits after decimal point) – "*BlendTank1Id*" – optional parameter needed only for blended fuel grade, means identifier of first tank for blended fuel grade (integer, value range from 0 to 20, value 0 means no tank configured) – "*BlendTank1Percentage*" – optional parameter needed only for blended fuel grade, means blend percentage from first tank for blended fuel grade (integer, value range from 1 to 99) – "*BlendTank2Id*" – optional parameter needed only for blended fuel grade, means identifier of second tank for blended fuel grade (integer, value range from 0 to 20, value 0 means no tank configured)

Note	1. Some of the fields may be absent in this message meaning that there is error or no data for filling them.
	2. Some fields may have null value indicating absence of value.
	3. UploadStatus request can be initiated by the server by adding an additional optional field "*GetStatus*" with a Boolean value set to *true* to a confirmation response. Example of confirmation response for UploadPumpTransaction request with request for status:
	`{`
	`"Protocol":"jsonPTS",`
	`"Packets":[{`
	`"Id":127,`
	`"Type":"UploadPumpTransaction",`
	`"Message":"OK",`
	`"GetStatus":true`
	`}]`
	`}`

Example	`{`
	`"Protocol": "jsonPTS",`
	`"PtsId": "0054002E524E500420323442",`
	`"Packets": [{`
	`"Id": 21444,`
	`"Type": "UploadStatus",`
	`"Data": {`
	`"ConfigurationId": "747a3fe2",`
	`"DateTime": "2022-09-20T19:30:08",`
	`"FirmwareDateTime": "2022-09-15T14:25:16",`
	`"PtsStartupSeconds": 59,`
	`"BatteryVoltage": 3023,`
	`"CpuTemperature":37,`
	`"PowerDownDetected": false,`
	`"Pumps": {`
	`"IdleStatus": {`
	`"Ids": [1, 3],`
	`"NozzlesUp": [1, 0],`
	`"LastNozzles": [0, 0],`
	`"LastTransactions": [0, 0],`

```
            "LastVolumes": [0.00, 0.00],
            "LastAmounts": [0.00, 0.00],
            "LastPrices": [1.11, 1.11],
            "Requests": ["", ""]
          },
          "FillingStatus": {
            "Ids": [2],
            "Nozzles": [2],
            "Transactions": [2996],
            "Volumes": [43.80],
            "Amounts": [45.99],
            "Prices": [1.05]
          },
          "EndOfTransactionStatus": {},
          "OfflineStatus": {
            "Ids": [4]
          },
          "Users": ["","","admin",""]
        },
        "Probes": {
          "OnlineStatus": {
            "Ids": [1, 2],
            "Errors": [2],
            "CriticalHighProductAlarms": [],
            "HighProductAlarms": [],
            "LowProductAlarms": [],
            "CriticalLowProductAlarms": [],
            "HighWaterAlarms": [],
            "TankLeakageAlarms": [],
            "Measurements": [
              [1, 1000.0, 10.0, 20.0, 20000, 100, 15000,
19900, null, null, 20],
              [2, 2520.0, 50.0, 25.3, 25500, 500, 1000,
25500, 759.0, 25500.0, 37]
            ]
          },
          "OfflineStatus": {
            "Ids": [3]
          }
        },
        "PriceBoards": {
            "OnlineStatus": {},
            "OfflineStatus": {
            "Ids": [1]
          }
        },
        "Readers": {
          "OnlineStatus": {
            "Ids": [1, 2, 3, 4],
            "Tags": ["12345", "", "", "0400527ec4"]
          },
```

```
        "OfflineStatus": {}
      },
      "Gps": {
        "Status": "Absent"
      },
      "FuelGrades":[{
        "Id":1,
        "Name":"Petrol",
        "Price":27.50,
        "ExpansionCoefficient":0.00110
      },{
        "Id":2,
        "Name":"Diesel",
        "Price":23.99,
        "ExpansionCoefficient":0.00082
      }]
    }
  }]
}
```


Real-life example of UploadStatus request can be found here: https://www.technotrade.ua/Uploads/Files/PTS-2/UploadStatus_example.json

Response

Possible responses:

- confirmation response message in case of successful reception
- error response message in case remote server faces a problem with this message reception, in this case PTS-2 controller should send this message again to a remote server

Example of response in case of successful operation:


```
{
  "Protocol":"jsonPTS",
  "Packets":[{
    "Id":21444,
    "Type":"UploadStatus",
    "Message":"OK"
  }]
}
```

209. Confirmation response with a single data setting request

Purpose	Remote server can add a single setting request in confirmation response to allow the data to be get/set. For the request type a field named "Set*RequestType*" is used, data is sent in "*Data*" field.
Notes	1. Inside each of the "*Packets*" elements keep the field "Id" to be the first and the field "*Data*" to be the last. 2. Execution of a setting request may require restarting of the PTS-2 controller, so this should be accounted and on the side of the remote server. 3. During the time while the PTS-2 controller is busy executing the setting request the PTS-2 controller may respond with HTTP status code 503 Service Unavailable for any requests coming to as to a web-server.

Example	Example of confirmation response for UploadStatus request in case of successful operation with SetFuelGradesConfiguration request to update prices for fuel grades:
	`{`
	`"Protocol":"jsonPTS",`
	`"Packets":[{`
	`"Id":127,`
	`"Type":"UploadStatus",`
	`"Message":"OK",`
	`"SetRequestType":"SetFuelGradesConfiguration",`
	`"Data":{`
	`"FuelGrades":[{`
	`"Id":1,`
	`"Name":"Petrol",`
	`"Price":27.50,`
	`"ExpansionCoefficient":0.00110`
	`},{`
	`"Id":2,`
	`"Name":"Diesel",`
	`"Price":23.99,`
	`"ExpansionCoefficient":0.00082`
	`}]`
	`}`
	`}]`
	`}`

Response	After executing the request the PTS-2 controller will connect to the server and send data as described in the description of the sent request.

210. RequestTagsInformation request

Purpose	Requests information about tags from server
Data	"*Tags*" – array of object with the following elements: – "*Tag*" – tag identifier (string, up to 32 hexadecimal symbols) – "*Pump*" – logical number of the pump (integer, range from 1 to 120) – "*Nozzle*" – number of selected nozzle (integer, range from 1 to 6) – "*FuelGradeId*" – identifier of selected fuel grade (integer, range from 1 to 20) – "*FuelGradeName*" – name of selected fuel grade (string, up to 20 ASCII characters) – "*Price*" – price for the fuel grade (float, up to 3 digits after decimal point)
Notes	Fields "*FuelGradeId*" and "*FuelGradeName*" are optional, they are present in request only when PTS-2 controller has tank configured to specific fuel grade.
Response	TagsInformation response

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Type":"RequestTagsInformation",
	"Data":{
	"Tags":[{
	"Tag":"1234567890ABCDEF",
	"Pump":1,
	"Nozzle":2,
	"FuelGradeId":2,
	"FuelGradeName":"Diesel",
	"Price":2.34
	},{
	"Tag":"1122334455667788",
	"Pump":2,
	"Nozzle":1,
	"FuelGradeId":1,
	"FuelGradeName":"Petrol",
	"Price":1.23
	}]
	}
	}]
	}

211. TagsInformation response

Purpose	Returns information about tags from server
Data	"*Tags*" – array of object with the following elements: – "*Tag*" – tag identifier (string, up to 32 hexadecimal symbols) – "*Valid*" – flag indicating if the tag is valid (Boolean, possible values: "*true*", "*false*") – "*FuelGradeIds*" – array of identifiers of fuel grade allowed for this tag (each element is integer, range from 1 to 20) – "*AccountType*" – type of units stored on account of the tag (string, up to 8 ASCII symbols), possible values: • "*Volume*": for volumetric units (for example prepaid liters) • "*Amount*": for money units (for example prepaid money) – "*Balance*" – quantity of units remaining on account of the tag (float, up to 3 digits after decimal point) – "*Price*" – price to be used for filling (float, up to 3 digits after decimal point) – "*OwnerName*" – name of tag owner (string, up to 20 ASCII characters) – "*CompanyName*" – name of tag owner company (string, up to 20 ASCII characters) – "*DepartmentName*" – company department of tag owner (string, up to 20 ASCII characters) – "*VehicleNumber*" – vehicle number of tag holder (string, up to 20 ASCII characters) – "*RegistrationNumber*" – registration number of the tag owner (string, up to 20 ASCII characters) – "*Phone*" – phone number of tag owner (string, up to 20 ASCII characters) – "*AdditionalInfo*" – additional tag information (string, up to 50 ASCII characters)
Notes	1. There can be many packets inside the same response received from server. 2. Field "*FuelGradeIds*" is optional and is used to show which fuel grades are allowed for the tag, if this field is absent – it means that all fuel grades are allowed. 3. Field "*Price*" is optional and may be omitted, in this case price is taken from fuel grades configuration, in this case the PTS-2 controller should have configuration of fuel grades set with non-zero price for nozzle to be authorized (see SetFuelGradesConfiguration and SetPumpNozzlesConfiguration requests). 4. Fields "*AccountType*" and "*Balance*" are optional and may be omitted. If any of these fields is missing then authorization of the pump will be done without any limit. If both fields are present then authorization of the pumps is done with a limitation set in these fields. 5. Fields "*OwnerName*", "*CompanyName*", "*DepartmentName*", "*VehicleNumber*", "*RegistrationNumber*", "*Phone*" and "*AdditionalInfo*" are optional.

Example	{
	"Protocol":"jsonPTS",
	"Packets":[{
	"Id":1,
	"Message":"OK",
	"Type":"TagsInformation",
	"Data":{

```
      "Tags":[{
        "Tag":"1234567890ABCDEF",
        "Valid":true,
        "FuelGradeIds":[1,2],
        "AccountType":"Amount",
        "Balance":200,
        "Price":3.25,
        "OwnerName":"John Smith",
        "CompanyName":"Technotrade LLC",
        "DepartmentName":"IT",
        "VehicleNumber":"AA1234OE",
        "RegistrationNumber":"ABCDEFGH12345",
        "Phone":"+380971234567",
        "AdditionalInfo":""
      },{
        "Tag":"1122334455667788",
        "Valid":true,
        "FuelGradeIds":[3],
        "AccountType":"Volume",
        "Balance":100,
        "Price":2.75,
        "OwnerName":"Anastasia Gray",
        "CompanyName":"Technotrade LLC",
        "DepartmentName":"Accountant",
        "VehicleNumber":"AA5677KK",
        "RegistrationNumber":"12345678AAAA",
        "Phone":"+380971234567",
        "AdditionalInfo":""
      }]
    }
  }]
}
```

ERROR MESSAGES

List of possible error messages and their descriptions:

CODE	ERROR MESSAGE	DESCRIPTION
1	JSONPTS_ERROR_NOT_FOUND	Request not found
2	INVALID_JSON_REQUEST	Invalid JSON request
3	NO_DATA	No data
4	NO_DATA_FOR_RESPONSE	No data for response
5	JSON_REQUEST_IS_TOO_LONG	JSON request is too long
6	JSONPTS_ERROR_NO_PERMISSIONS	Access forbidden
7	JSONPTS_ERROR_NO_SD_FOUND	No SD found
8	JSONPTS_ERROR_POWER_DOWN_DETECTED	Power down detected
9	JSONPTS_ERROR_SD_NOT_MOUNTED	SD is not mounted
10	JSONPTS_ERROR_FILE_UPLOAD_PROCESS_RUNNING	File upload process running
11	JSONPTS_ERROR_SD_ERROR	SD error
12	JSONPTS_ERROR_NO_CALIBRATION_CHART_FOUND_ERROR	No calibration chart found
13	JSONPTS_ERROR_COULD_NOT_CHECK_FILE	Could not check file
14	JSONPTS_ERROR_COULD_NOT_DELETE_FILE	Could not delete file
15	JSONPTS_ERROR_INCORRECT_PUMPS_CONFIGURATION	Incorrect pumps configuration
16	JSONPTS_ERROR_INCORRECT_PROBES_CONFIGURATION	Incorrect probes configuration
17	JSONPTS_ERROR_COULD_NOT_GET_DATETIME	Could not get datetime
18	JSONPTS_ERROR_COULD_NOT_GET_PUMP_NUMBER	Could not get pump number
19	JSONPTS_ERROR_COULD_NOT_GET_PUMP_TRANSACTION_NUMBER	Could not get pump transaction number
20	JSONPTS_ERROR_PUMP_NUMBER_OUT_OF_RANGE	Pump number is out of range
21	JSONPTS_ERROR_PUMP_TRANSACTION_NUMBER_OUT_OF_RANGE	Pump transaction number is out of range
22	JSONPTS_ERROR_PUMP_TRANSACTION_NOT_FOUND	Pump transaction not found. Error message contains additional fields: – "*Pump*" – logical number of the pump (integer, range from 1 to 120) – "*Transaction*" – number of transaction (integer, range from 1 to 65535)
23	JSONPTS_ERROR_TANK_NUMBER_OUT_OF_RANGE	Tank number is out of range
24	JSONPTS_ERROR_COULD_NOT_GET_TANK_NUMBER	Could not get tank number
25	JSONPTS_ERROR_COULD_NOT_GET_TRANSACTION_NUMBER	Could not get transaction number
26	JSONPTS_ERROR_TRANSACTION_NUMBER_OUT_OF_RANGE	Transaction number is out of range
27	JSONPTS_ERROR_TRANSACTION_NUMBER_NOT_MATCH	Transaction number does not match
28	JSONPTS_ERROR_TRANSACTION_NUMBER_ALREADY_EXIST	Transaction number already exist
29	JSONPTS_ERROR_COULD_NOT_GET_NOZZLE_NUMBER	Could not get nozzle number
30	JSONPTS_ERROR_COULD_NOT_GET_FUEL_GRADE_ID	Could not get fuel grade Id
31	JSONPTS_ERROR_COULD_NOT_GET_NOZZLE_NUMBER_FUEL_GRADE_ID	Could not get nozzle number and fuel grade Id
32	JSONPTS_ERROR_COULD_NOT_GET_NOZZLE_NUMBER_FROM_GRADE_ID	Could not get nozzle number from fuel grade Id
33	JSONPTS_ERROR_NOZZLE_NUMBER_OUT_OF_RANGE	Nozzle number is out of range
34	JSONPTS_ERROR_COULD_NOT_GET_TYPE	Could not get type
35	JSONPTS_ERROR_COULD_NOT_GET_NAME	Could not get name
36	JSONPTS_ERROR_TYPE_OUT_OF_RANGE	Type is out of range
37	JSONPTS_ERROR_COULD_NOT_GET_DOSE_VALUE	Could not get dose value
38	JSONPTS_ERROR_COULD_NOT_GET_PRICE_VALUE	Could not get price value
39	JSONPTS_ERROR_DUPLICATED_AUTHORIZATION_REQUEST	Duplicated authorization request
40	JSONPTS_ERROR_PUMP_BUSY_OTHER_USER	Pump is busy by other user. Error message contains additional fields: – "*Pump*" – logical number of the pump (integer, range from 1 to 120) – "*User*" – name of user, to whom current filling belongs (string, up to 10 ASCII symbols)
41	JSONPTS_ERROR_PUMP_BUSY_OTHER_REQUEST_EXECUTED	Pump is busy with other request being executed. Error message contains additional fields: – "*Pump*" – logical number of the pump (integer, range from 1 to 120) – "*Request*" – name of request being currently executed (string, up to 20 ASCII symbols)
42	JSONPTS_ERROR_PUMP_BUSY_FILLING	Pump is busy with filling.

		Error message contains additional field: – "*Pump*" – logical number of the pump (integer, range from 1 to 120)
43	**JSONPTS_ERROR_PUMP_NOT_IN_FILLING_PROCESS**	Pump is not in filling process
44	**JSONPTS_ERROR_COULD_NOT_GET_STATE_VALUE**	Could not get state value
45	**JSONPTS_ERROR_STATE_VALUE_OUT_OF_RANGE**	State value is out of range
46	**JSONPTS_ERROR_USER_NOT_MATCH**	User does not match
47	**JSONPTS_ERROR_DATE_OUT_OF_RANGE**	Date is out of range
48	**JSONPTS_ERROR_TIME_OUT_OF_RANGE**	Time is out of range
49	**JSONPTS_ERROR_COULD_NOT_GET_HEIGHT**	Could not get height value
50	**JSONPTS_ERROR_COULD_NOT_GET_PROBE_NUMBER**	Could not get probe number
51	**JSONPTS_ERROR_PROBE_NUMBER_OUT_OF_RANGE**	Probe number is out of range
52	**JSONPTS_ERROR_PUMP_IS_NOT_CONFIGURED**	Pump is not configured
53	**JSONPTS_ERROR_RESTORE_CONFIGURATION_FAILED**	Restore of configuration failed
54	**JSONPTS_ERROR_CONFIGURATION_FILE_NOT_FOUND**	Configuration file not found
55	**JSONPTS_ERROR_CALIBRATION_CHART_NOT_CONFIGURED**	Calibration chart for tank is not configured
56	**JSONPTS_ERROR_TANK_NOT_CONFIGURED**	Tank is not configured
57	**JSONPTS_ERROR_PUMP_GRADE_NOT_CORRESPOND_TANK**	Pump fuel grade does not correspond to the tank fuel grade. Error message contains additional field: – "*Pump*" – logical number of the pump (integer, range from 1 to 120)
58	**JSONPTS_ERROR_INCORRECT_TANKS_CONFIGURATION**	Incorrect tanks configuration
59	**JSONPTS_ERROR_INCORRECT_PRICE_BOARDS_CONFIGURATION**	Incorrect price boards configuration
60	**JSONPTS_ERROR_INCORRECT_READERS_CONFIGURATION**	Incorrect readers configuration
61	**JSONPTS_ERROR_COULD_NOT_GET_TAG**	Could not get tag
62	**JSONPTS_ERROR_COULD_NOT_GET_READER_NUMBER**	Could not get reader number
63	**JSONPTS_ERROR_READER_NUMBER_OUT_OF_RANGE**	Reader number is out of range
64	**JSONPTS_ERROR_READER_NOT_CONFIGURED**	Reader is not configured
65	**JSONPTS_ERROR_TANK_PRODUCT_HEIGHT_IS_CRITICAL_LOW**	Product height in tank is critical low
66	**JSONPTS_ERROR_COULD_NOT_GET_PRICE_BOARD_NUMBER**	Could not get price board number
67	**JSONPTS_ERROR_PRICE_BOARD_NUMBER_OUT_OF_RANGE**	Price board number is out of range
68	**JSONPTS_ERROR_PRICE_BOARD_NOT_CONFIGURED**	Price board is not configured
69	**JSONPTS_ERROR_PUMP_STATUS_NOT_END_OF_TRANSACTION**	Pump is not in the end of transaction status
70	**JSONPTS_ERROR_TAG_IS_NOT_VALID**	Tag is not valid
71	**JSONPTS_ERROR_DATETIME_INCORRECT_FORMAT**	DateTime has incorrect format
72	**JSONPTS_ERROR_NO_FREE_USERS_TO_ADD**	No free non-configured users to add
73	**JSONPTS_ERROR_USER_LOGIN_USED**	User login is already used
74	**JSONPTS_ERROR_USER_ID_OUT_OF_RANGE**	User ID is out of range
75	**JSONPTS_ERROR_CAN_NOT_DELETE_LAST_USER**	Can not delete last user from user's configuration
76	**JSONPTS_ERROR_TAG_ALREADY_EXIST**	Specified tag already exists
77	**JSONPTS_ERROR_TAG_NOT_EXIST**	Specified tag does not exist
78	**JSONPTS_ERROR_REPEATED_AUTHORIZE_PUMP_PRICE_DIFFERENT**	Different price sent in repeated authorization request
79	**JSONPTS_ERROR_CAN_NOT_USE_DATA_UPLOAD_WEBSOCKETS_SAME_TIME**	Can not use upload data and WebSocket same time
80	**JSONPTS_ERROR_NO_USER_WITH_CONFIG_PERMISSION_CONFIGURED**	No user with configuration permission is configured
81	**JSONPTS_ERROR_TOO_BIG_TAGS_TOTAL_NUMBER_REQUESTED**	Too big tags total number requested
82	**JSONPTS_ERROR_COULD_NOT_DELETE_EXISTING_TAG**	Could not delete existing tag
83	**JSONPTS_ERROR_NOT_ENOUGH_DATA_IN_REQUEST_FOR_EXECUTION**	Not enough data in request for execution
84	**JSONPTS_ERROR_NOT_ENOUGH_DATA_FOR_TANK_AUTO_CALIB_CHART**	Not enough data for creation of tank automatic calibration chart
85	**JSONPTS_ERROR_TANK_AUTO_CALIB_PROBE_NOT_CONFIG**	Probe is not configured for the tank
86	**JSONPTS_ERROR_TANK_AUTO_CALIB_TANK_NOT_CONFIG**	Tank is not configured
87	**JSONPTS_ERROR_TANK_AUTO_CALIB_PROBE_NO_TEMPERATURE**	Probe of the tank does not measure temperature
88	**JSONPTS_ERROR_TANK_AUTO_CALIB_GRADE_NOT_CONFIG**	Fuel grade is not configured for the tank
89	**JSONPTS_ERROR_TANK_AUTO_CALIB_TEMP_COEF_NOT_CONFIG**	Temperature-expansion coefficient is not configured for fuel grade used in tank
90	**JSONPTS_ERROR_TANK_AUTO_CALIB_NOZZLES_NOT_CONFIG**	No pump nozzles are configured for tank
91	**JSONPTS_ERROR_PUMP_IS_NOT_LOCKED_BY_USER**	Pump is not locked by the user
92	**JSONPTS_ERROR_INVALID_IP_ADDRESS_VALUE**	Invalid IP-address value
93	**JSONPTS_ERROR_INVALID_PORT_VALUE**	Invalid port value
94	**JSONPTS_ERROR_RECORD_WITH_HEIGHT_ALREADY_EXIST**	Record with specified height already exists
95	**JSONPTS_ERROR_RECORD_WITH_HEIGHT_NOT_EXIST**	Record with specified height does not exist
96	**JSONPTS_ERROR_OPERATION_ERROR**	Operation error

97	**JSONPTS_ERROR_PAYMENT_FORM_NUMBER_OUT_OF_RANGE**	Payment form number is out of range Error message contains additional fields: "*PaymentFormId*" – identifier of the payment form (integer, range from 1 to 10)
98	**JSONPTS_ERROR_PAYMENT_FORM_NUMBER_NOT_CONFIGURED**	Payment form number is not configured Error message contains additional fields: "*PaymentFormId*" – identifier of the payment form (integer, range from 1 to 10)
99	**JSONPTS_ERROR_AUTOREAD_PUMPS_TOTALS_NOT_CONFIGURED**	Automatic readout of totals is not configured for pump
100	**JSONPTS_ERROR_INCORRECT_FUEL_GRADES_PRICES_CONFIGURATION**	Incorrect fuel grades prices configuration
101	**JSONPTS_ERROR_DATA_IS_NOT_READY**	Data is not ready, try again later
102	**JSONPTS_ERROR_PUMP_TRANSACTION_ALREADY_PAID**	Pump transaction is already paid. Error message contains additional fields: – "*Pump*" – logical number of the pump (integer, range from 1 to 120) – "*Transaction*" – number of transaction (integer, range from 1 to 65535)
103	**JSONPTS_ERROR_PUMP_TRANSACTION_AMOUNT_NOT_MATCH_PAYMENT**	Payment amount does not match the pump transaction amount. Error message contains additional fields: – "*Pump*" – logical number of the pump (integer, range from 1 to 120) – "*Transaction*" – number of transaction (integer, range from 1 to 65535)
104	**JSONPTS_ERROR_PAYMENT_ZERO_AMOUNT**	Payment is for zero amount
105	**JSONPTS_ERROR_COULD_NOT_GET_PAYMENT_TRANSACTION_NUMBER**	Could not get payment transaction number
106	**JSONPTS_ERROR_PAYMENT_TRANSACTION_NUMBER_OUT_OF_RANGE**	Payment transaction number is out of range
107	**JSONPTS_ERROR_PAYMENT_TRANSACTION_NOT_FOUND**	Payment transaction is not found
108	**JSONPTS_ERROR_COULD_NOT_REFUND_PAYMENT**	Could not refund payment
109	**JSONPTS_ERROR_PAYMENT_ALREADY_REFUNDED**	Payment transaction is already refunded
110	**JSONPTS_ERROR_SHIFT_IS_ALREADY_OPENED**	Working shift is already opened
111	**JSONPTS_ERROR_SHIFT_IS_NOT_OPENED**	Working shift is not opened
112	**JSONPTS_ERROR_SHIFT_IS_OPENED_BY_ANOTHER_USER**	Working shift is opened by another user. Error message contains additional fields: – "*User*" – name of the user, who opened the shift (string, up to 10 ASCII symbols)
113	**JSONPTS_ERROR_LEADING_OF_WORKING_SHIFTS_DISABLED**	Leading of working shifts is not enabled
114	**JSONPTS_ERROR_COULD_NOT_GET_SHIFT_INFORMATION**	Could not get working shift information
115	**JSONPTS_ERROR_DATETIME_NOT_CONFIGURED**	Date/time are not configured

ALERT CODES

Values of field "***Code***" depend on field "***DeviceType***" and may have the following values:

DeviceType	Code	Description
PTS	**1**	Low battery voltage detected
	2	High CPU temperature detected
	3	Power down detected
	4	Restart detected
Pump	**1**	Offline state detected
	20	Overfilling detected
	21	Overfilling detected for nozzle 1
	22	Overfilling detected for nozzle 2
	23	Overfilling detected for nozzle 3
	24	Overfilling detected for nozzle 4
	25	Overfilling detected for nozzle 5
	26	Overfilling detected for nozzle 6
	30	Filling in offline mode detected
	31	Filling in offline mode detected for nozzle 1
	32	Filling in offline mode detected for nozzle 2
	33	Filling in offline mode detected for nozzle 3
	34	Filling in offline mode detected for nozzle 4
	35	Filling in offline mode detected for nozzle 5
	36	Filling in offline mode detected for nozzle 6
Probe	**1**	Offline state detected
	2	Error detected
	3	Critical high product level detected
	4	High product level detected
	5	Low product level detected
	6	Critical low product level detected
	7	High water level detected
	8	Tank leakage detected
PriceBoard	**1**	Offline state detected
	2	Error detected
Reader	**1**	Offline state detected
	2	Error detected
	3	Low battery detected