Remote control of PG4UWMC, developer manual

Version: 1.17
Date: January 9, 2023

Table of Contents

Index of Figures

Index of Tables

Index of Codes

1. Introduction

The documentation explains how to use r*emote control of PG4UWMC* to write an *external driver* to allow interaction between *PG4UWMC control program* with *embedded multiprogramming hardware, or handler hardware (remote system)* via set of commands.

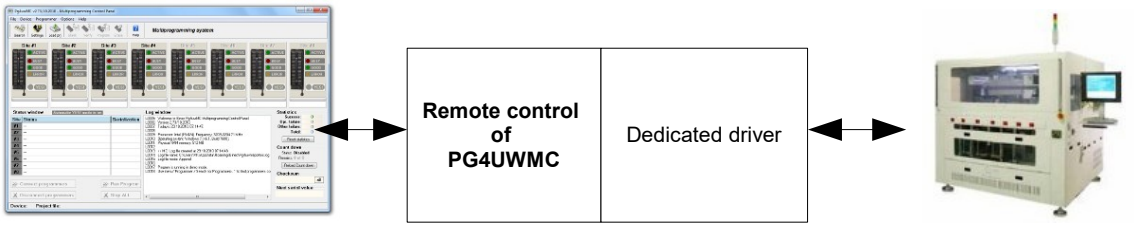

Figure 1: Automated multiprogramming system with our programmers embedded

Remote control of PG4UWMC allows to control programming process, obtain information about result of individual operations and start programming operations on *embedded programmers* using set of *published commands.*

PG4UWMC control program acts as listening/responding *server*, responding to questions from *remote application,* which acts as asking *client*. Server-client connection is realized using *named pipe*[*1]*. Once the* connection is established, client may exchange information with server, based on *commands* formed into *packets*.

PG4UWMC software, if switched to *network mode,* is capable to manage (search, start, control and monitor instances of PG4UW on network computers) up to 64 programming sites at the same time. Check Menu / Options / Network mode and read section *Network mode in PG4UWMC* in help.

*1 *Pipe is a section of shared memory used by processes for communication purposes. The process that creates a pipe is the pipe server. A process that connects to a pipe is a pipe client. For more information on this, try http://en.wikipedia.org/wiki/Named_pipe*

2. Virtual Universal Handler (VUH) – example

This is an **example application** written as reference for you **to ease your own driver development.** Implements all available commands (described later) and shows how the automated multiprogramming system should work. Gives you possibility to run operation in automatic mode or pass particular commands manually and see how programmer and handler are working together.

Let's presume that you have connected programmer (for example BeeHive204AP), installed control software (PG4UW) and also multiprogramming support (PG4UWMC), prepared testing project and proper modules and devices to run demonstration with.

How to start VUH demonstration step by step:

1. Set path to installed PG4UWMC.exe
2. Start PG4UWMC (remotely from VUH)
3. Initialize connection to PG4UWMC server
4. Load Project file (the Load Project dialog appears, see Figure 3)
5. Start execution (operation type should be set by project, otherwise “Program” operation will be set as default)
6. Stop execution
    1. after all running operations are finished
    2. immediately (Total-Stop)
7. Close PG4UWMC (remotely from VUH, only if wasn't stopped by Total-Stop)

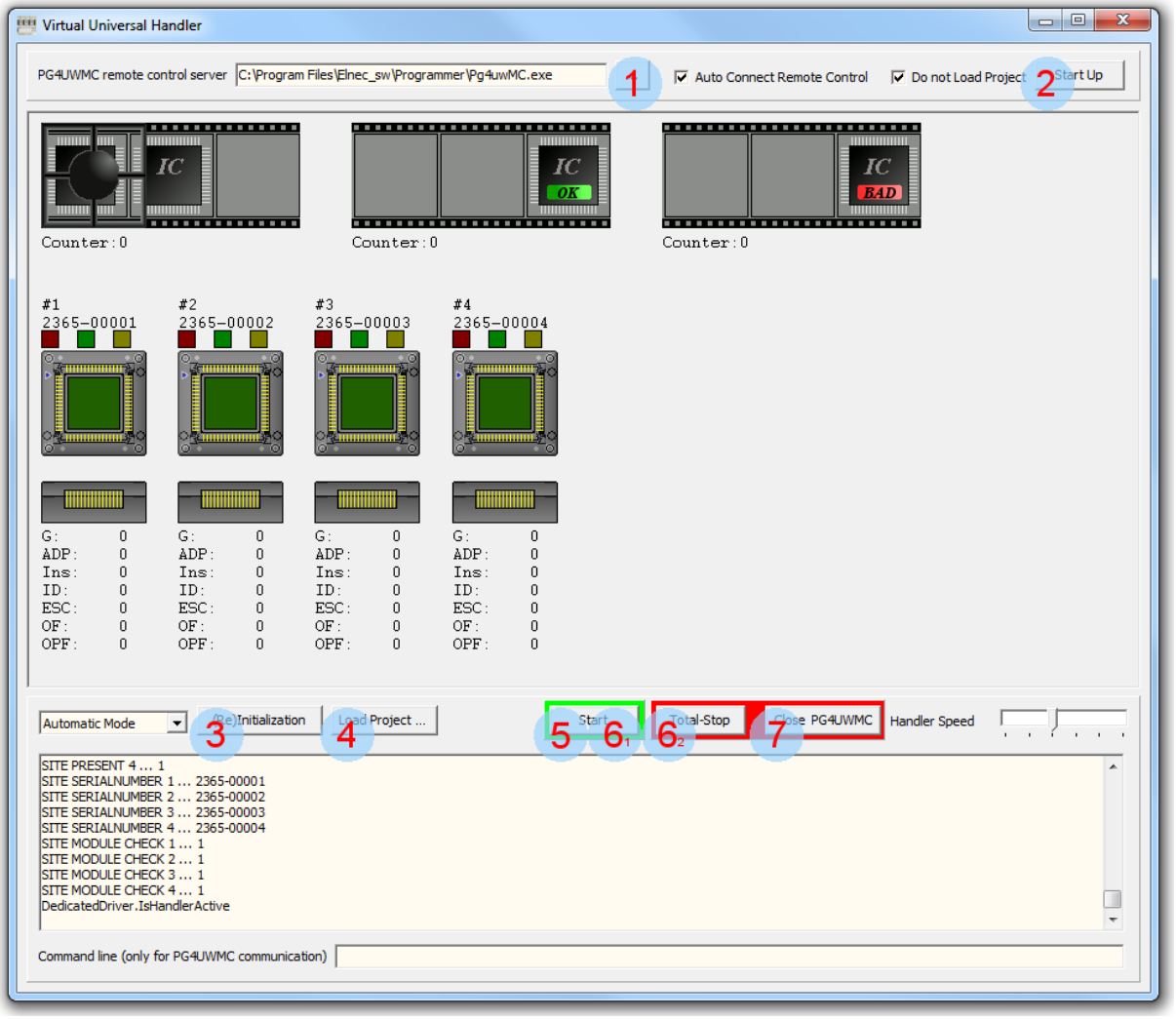

Figure 2: Virtual Universal Handler

Load Project dialog allows you to select project file to be loaded and pass some additional parameters to PG4UWMC to suppress particular dialogs which might appear (see Chapter 3.3 Dialogs which may appear during active remote control, Page 9).

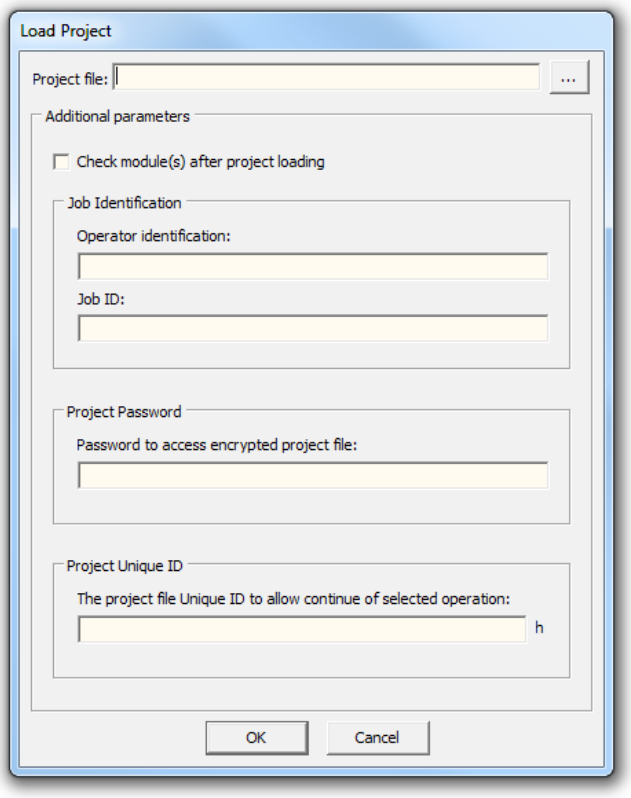

Figure 3: Load Project dialog

List of classes used in VUH example:

- (T)FormVUH class contains standard application window (form) and main control logic for Virtual Universal Handler
- (T)UniHandlerClient class contains the declarations for implementation of Pipes (a section of shared memory that processes use for communication)
- (T)DedicatedDriver class contains dedicated driver functions for real handler
- (T)Statistics class contains statistics functions
- (T)Sprite class contains base class for sprite graphics
- (T)ReelSprite class contains sprite graphics class for device reel visualization
- (T)PusherSprite class contains sprite graphics class for device pusher visualization
- (T)SocketSprite class contains sprite graphics class for programmer socket visualization
- (T)FormOperationalResult class contains form for semiautomatic mode
- (T)FormLoadProject class contains form for load project dialog

3. PG4UWMC remote control

3.1. Manual start-up

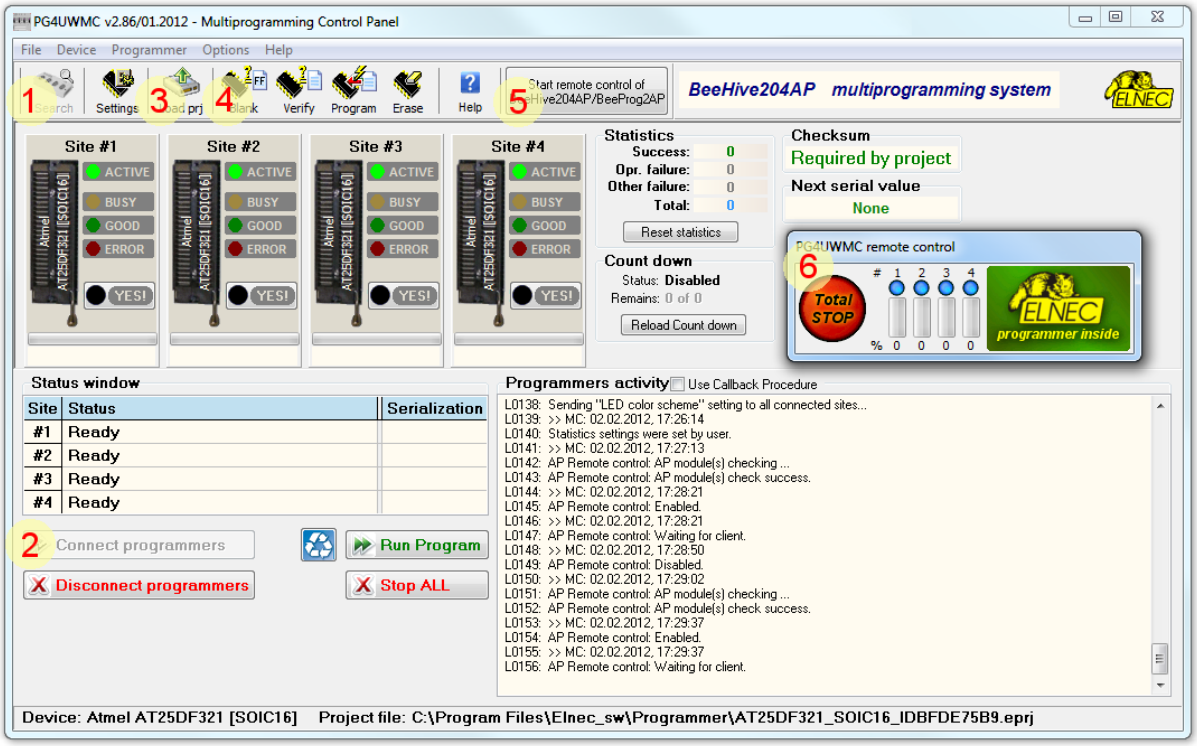

Figure 4: PG4UWMC in remote control mode

How to start remote control mode in PG4UWMC step by step:

1. Search industrial programmers (BeeProg2AP, BeeHive204AP, ...)
2. Connect programmers
3. Load project file
4. Select desired operation (if not defined by project)
5. Start remote control mode and communicate with PG4UWMC via remote control
6. Stop remote control mode

3.2. Start-up from command line

You can start PG4UWMC from another process (using API ShellExecute), and use following parameters to affect subsequent behavior:

Parameter	Description
/autoconnectremotecontrol	Remote control server will start immediately after start of PG4UWMC program
/donotloadproject	PG4UWMC will NOT load last project saved in INI file*1 during recent close

Table 1: PG4UWMC command-line parameters description

*1 *INI file – file caring PG4UWMC configuration settings*

3.3. Dialogs which may appear during active remote control

PG4UWMC remote control Info window is visible on top of PG4UWMC window during active remote control and is always visible on the desktop. You can adjust its position with respect to the main screen of handler control software.

Figure 5: PG4UWMC remote control Info window

Following dialogs can be suppressed using parameters passed to PG4UWMC during project load (see 4.2.17 SITES LOAD PROJECT.

Authentication dialog shown during encrypted project file load.

Figure 6: Authentication dialog

Job identification dialog is shown after project file load, if the project was created in protected mode.

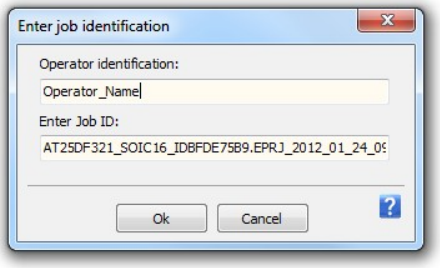

Figure 7: Job identification dialog

Project file unique ID required dialog is shown before first operation with device, if this feature was required by project settings.

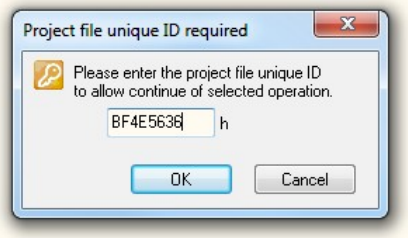

Figure 8: Project file unique ID required dialog

Job reports (generated before successive project load or during PG4UWMC close if any operation with device was performed) are automatically saved into defined location without displaying any dialog. notification.

4. Detailed description of protocol

Remote control protocol consists of 2 layers:

- Application layer
- Transport layer

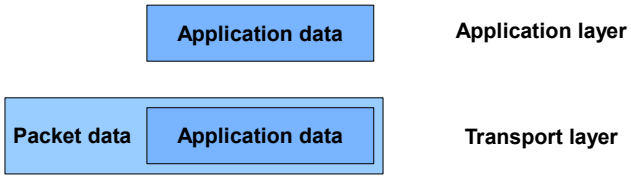

Figure 9: Encapsulation of application data descending through the protocol stack

4.1. Automated multiprogramming operation flowchart

Following figure contains simplified visualization of algorithm used in VUH example.

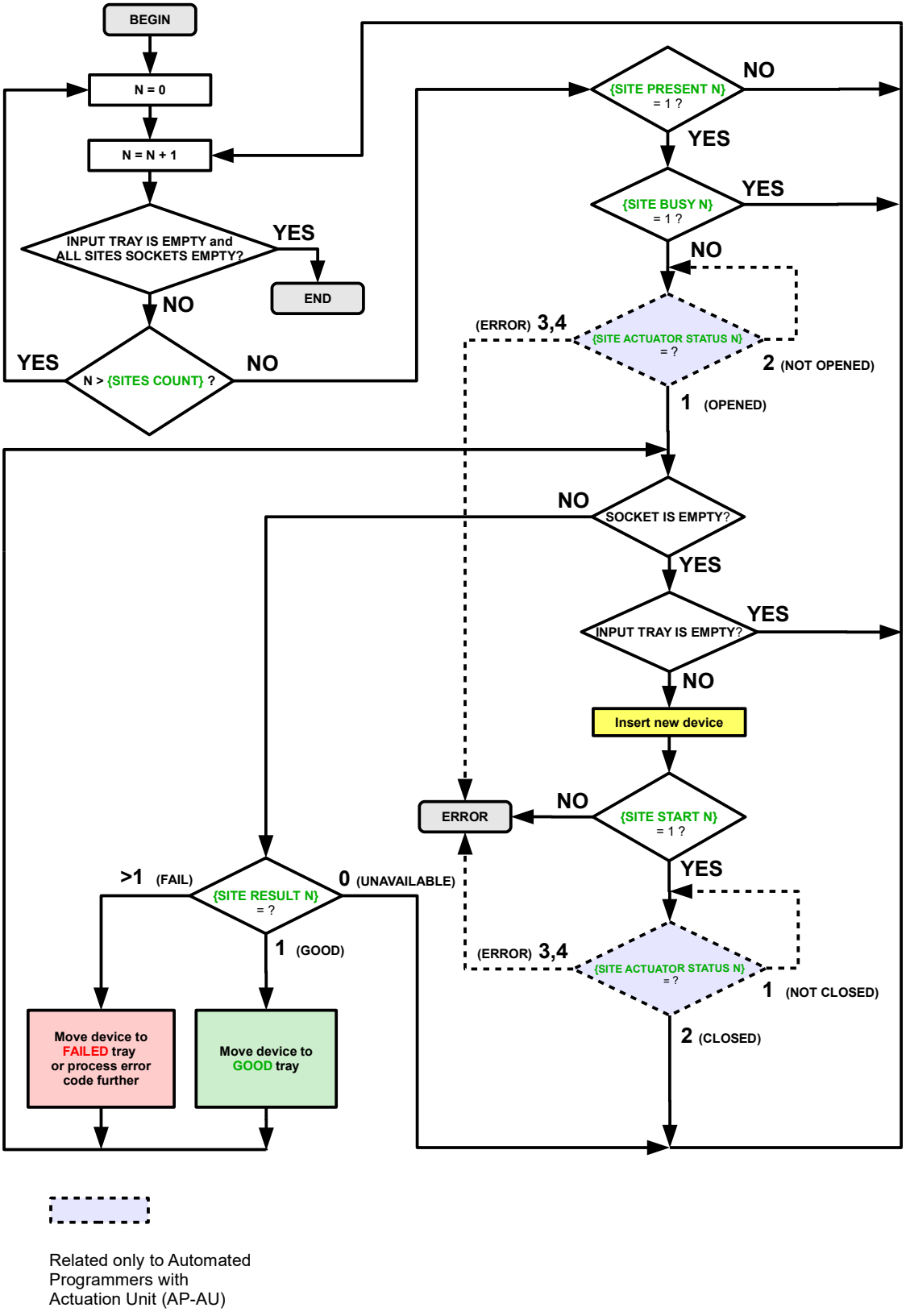

Figure 10: Example of automated multiprogramming operation flowchart
({text inside curly brackets} represents response received after issuing particular request (command),
N represents programmer's site number, details explained later)

4.2. Commands (Application layer)

The are two types of commands available:

- without parameter(s)
- with parameter(s)

Commands could be also grouped into two sets by their purpose:

- Common commands
- ZIF socket actuation unit related commands

Command [1]	Chapter with description
“PROTOCOL VERSION”	4.2.1
“PG4UWMC VERSION”	4.2.2
“PROGRAMMING ENVIRONMENT”	4.2.3
“SITES COUNT”	4.2.4
“SITES STOP”	4.2.5
“APPLICATION CLOSE”	4.2.6
“SITE PRESENT N” [2]	4.2.7
“SITE BUSY N”	4.2.8
”SITE BUFFER SYNCHRONIZATION N”	4.2.9
“SITE RESULT N”	4.2.10
“SITE START N”	4.2.11
“SITE SERIALNUMBER N”	4.2.12
“SITE MODULE CHECK N”	4.2.13
“SITE ZIF SOCKET EMPTY N”	4.2.14
“SITE ZIF GET COUNTER VALUE N”	4.2.15
“SITE ZIF DISABLE LIFETIME WARNINGS AND GET COUNTER VALUE N”	4.2.16
“SITES LOAD PROJECT X [Y]” [3]	4.2.17
“SITES COMMENT M” [4]	4.2.18
“SITE ACTUATOR STATUS N”	4.2.19
“SITE ACTUATOR CLOSE ZIF N”	4.2.20
“SITE ACTUATOR OPEN ZIF N”	4.2.21
“SITES ACTUATOR CLOSE ZIFS EN BLOC”	4.2.22
“SITES ACTUATOR OPEN ZIFS EN BLOC”	4.2.23
“SITE ACTUATOR READ CONFIGURATION N” [5]	4.2.24
“SITES ACTUATOR EJECT MODE START” [5]	4.2.25
“SITES ACTUATOR EJECT MODE STOP” [5]	4.2.26
“SITES ACTUATOR EJECT MODE GET STATUS” [5]	4.2.27
“SITES ACTUATOR INSTALL MODE START” [5]	4.2.28
“SITES ACTUATOR INSTALL MODE STOP” [5]	4.2.29
“SITES ACTUATOR INSTALL MODE GET STATUS” [5]	4.2.30
“SITE ACTUATOR INVOKE EVENT N” [5]	4.2.31

1 *Commands and parameters are normal strings, not case sensitive*

2 *N represents number of site (once configured by user and assigned by PG4UWMC), which the request is related to*

3 *X represents path to project file, [Y] stands for optional parameters*

4 *M represents comment message*

5 *Commands are related to the BeeHive204AP-AU systems with ZIF socket actuation unit, for other models of the ZIF socket actuation systems these command are ignored or the response could be erroneous*

“SITE STATISTICS N”	4.2.32
“SITES STATISTICS”	4.2.33
Any other, e. g. “INVENTED COMMAND”	“UNKNOWN COMMAND”

Table 2: Command set description

4.2.1. PROTOCOL VERSION

Mandatory command, must be issued as first command after establishment of connection to obtain information about version of *PG4UWMC remote control* protocol.

Response [6]	Description
e. g. “1.05”	-

Table 3: Parameters and Responses for command “PROTOCOL VERSION”

4.2.2. PG4UWMC VERSION

Use this command to obtain information about version of *PG4UWMC*.

Response	Description
e. g. “2.92”	-

Table 4: Parameters and Responses for command “PG4UWMC VERSION”

4.2.3. PROGRAMMING ENVIRONMENT

Use this command to obtain information about programming environment, e.g. programmers names and S/N, project (multiproject) file name, project (multiproject) file MD5 checksum, manufacturer, device name, etc. MD5 checksum of project file is calculated during creating project (see Log of PG4UW), and then each time after project load, therefore we recommend to call PROGRAMMING ENVIRONMENT command after SITES LOAD PROJECT command.

Keys and **Values** in response are delimited by colon “**:**” (e.g. Key**:**Value). Key:Value pairs are concatenated with pipe “**|**” (e.g. Key:Value**|**Key:Value**|**etc.**|**).

Response
“Programmer(s):Programmer1 (S/N:1200-00001), Programmer2 (S/N:1200-00002)|ProjectFileName:C:\ W25Q32JV.eprj|ProjectFileMD5:6A955CE168AB1997BD3D6F62205AED97|DeviceManufacturer:Winbond| DeviceNameFromProject:W25Q32JV [SOIC8-200] (QuadSPI)|DeviceNameBySW:W25Q32JVxxxQ [SOIC8-200] (QuadSPI)|ModuleType:AP3 SOIC8-200 SFlash-1|MainChecksumValue:00818CDAh| MainChecksumType:x8|MainChecksumForm:Straight|MainChecksumCalcultationOptions:'Buffer(s) included to Main Checksum calculation:\n 1. Buffer #1 "Program Memory" (0h..FFFFh) (x8)\n Excluded blocks used:\n Block #0: 0h..1h (x8)\n 2. Buffer #2 "Data EEPROM" (0h..3FFh) (x8)\n Excluded blocks used:\n Block #0: 0h..1h (x8)'|”

Table 5: Example of response for command “PROGRAMMING ENVIRONMENT”

Items (Key, Value pairs) obtained in response may vary with different environment configuration, therefore should not be expected as mandatory. Items availability, order and key names may also vary with different protocol version. Complete list of currently available items is stated in next table.

6 *response is normal string, upper case*

Items							
Key	**Value (example)**						
Programmer(s)	Programmer1 (S/N:1200-00001), Programmer2 (S/N:1200-00002),...						
ProjectFileName	C:\W25Q32JV.eprj						
ProjectFileMD5[7]	6A955CE168AB1997BD3D6F62205AED97						
MultiProjectFileName	C:\W25Q32JV.eprj-m						
MultiProjectFileMD5[8]	C8D8A6A96BD26DA3D25CE47BA11D4D5B						
DeviceManufacturer	Winbond						
DeviceNameFromProject	W25Q32JV [SOIC8-200] (QuadSPI)						
DeviceNameBySW	W25Q32JVxxxQ [SOIC8-200] (QuadSPI) (this value can be sometimes different from value of DeviceNameFromProject because of SW changes during updates)						
ModuleType	AP3 SOIC8-200 SFlash-1						
MainChecksumValue	00818CDAh						
MainChecksumType	x8	x16 LE	x16 BE	CRC-CCITT	CRC-XModem	CRC-16	CRC-32
MainChecksumForm	Straight		Negated		Supplement		
MainChecksumCalcultation Options	'Buffer(s) included to Main Checksum calculation:\n 1. Buffer #1 "Program Memory" (0h..FFFFh) (x8)\n Excluded blocks used:\n Block #0: 0h..1h (x8)\n 2. Buffer #2 "Data EEPROM" (0h..3FFh) (x8)\n Excluded blocks used:\n Block #0: 0h..1h (x8)'						

Table 6: List of available items in response for command "PROGRAMMING ENVIRONMENT"

4.2.4. SITES COUNT

Use this command to obtain information about count of sites of all programmers connected to *PG4UWMC,* including also inactive sites.

Response	Description
e. g. "4"	-

Table 7: Parameters and Responses for command "SITES COUNT"

4.2.5. SITES STOP

Issue this command if you want to immediately stop running operation on each connected site and disconnect PG4UWMC remote control interface (after that, you won't be able to communicate with PG4UWMC). This is used to perform for example *Total-Stop.*

Response	Description
"0"	Not accepted
"1"	Accepted

Table 8: Parameters and Responses for command "SITES COUNT"

4.2.6. APPLICATION CLOSE

Issue this command if you want to immediately close PG4UWMC application.

7 *ProjectFileMD5 means MD5 checksum of loaded project file*
8 *MultiProjectFileMD5 means MD5 checksum of loaded multi project file*

Response	Description
“0”	Error: Unknown
“1”	OK: Successfully closed
“2”	Error: Some site(s) is busy
“3”	Error: Impossible to disconnect site(s)

Table 9: Detailed description of responses for command “APPLICATION CLOSE”

4.2.7. SITE PRESENT N

Use this command to obtain information whether particular site is connected and active (present).

Parameters	Mandatory	Description
N	Yes	Represents number of site (once configured by user and assigned by PG4UWMC), which the request is related to

Table 10: Parameters for command “SITE PRESENT N”

Response	Description
“0”	Not present
“1”	Present

Table 11: Detailed description of responses for command “SITE PRESENT N”

4.2.8. SITE BUSY N

Use this command to obtain information whether particular site finished previous operation and is ready.

Parameters	Mandatory	Description
N	Yes	Represents number of site (once configured by user and assigned by PG4UWMC), which the request is related to

Table 12: Parameters for command “SITE BUSY N”

Response	Description
“0”	Not busy
“1”	Busy

Table 13: Detailed description of responses for command “SITE BUSY N”

4.2.9. SITE BUFFER SYNCHRONIZATION N

Use this command to obtain information whether particular SITE is performing buffer synchronization (related only to programmers with internal SSD). Can be used e.g. after command SITE START N.

Keys and **Values** in response are delimited by colon “**:**” (e.g. Key**:**Value). Key:Value pairs are concatenated with pipe “**|**” (e.g. Key:Value**|**Key:Value**|**etc.**|**).

Items (Key, Value pairs) obtained in response may vary with different environment configuration, therefore should not be expected as mandatory. Items availability, order and key names may also vary with different protocol version. Complete list of currently available items is stated in next table.

Items	
Key	**Value (example)**
Status	Unavailable
	Inactive
	Active
Progress	<Value> (in %)

Table 14: List of available items in response for command "SITE BUFFER SYNCHRONIZATION N"

4.2.10. SITE RESULT N

Use this command to obtain information about result of operation.

Parameters	Mandatory	Description
N	Yes	Represents number of site (once configured by user and assigned by PG4UWMC), which the request is related to

Table 15: Parameters for command "SITE RESULT N"

Response	Description
"0"	Unavailable
"1"	Success
"2"	Module Test Failure
"3"	Insertion Test Failure
"4"	Identification Check Failure
"5"	Access Canceled By User Failure
"6"	Other Failure
"7" - "999"	Operational Fail

Table 16: Detailed description of responses for command "SITE RESULT N"

4.2.11. SITE START N

Issue this command if you want to immediately start operation (previously defined in PG4UWMC manually, or by project load).

Parameters	Mandatory	Description
N	Yes	Represents number of site (once configured by user and assigned by PG4UWMC), which the request is related to

Table 17: Parameters for command "SITE START N"

Response	Description
"0"	Not started
"1"	Started OK

Table 18: Detailed description of responses for command "SITE START N"

4.2.12. SITE SERIALNUMBER N

Use this command to obtain information about serial number of particular site. It is useful to verify correct position of sites in automated programming system.

Parameters	Mandatory	Description
N	Yes	Represents number of site (once configured by user and assigned by PG4UWMC), which the request is related to

Table 19: Parameters for command "SITE SERIALNUMBER N"

Response	Description
"0"	Site not ready
"1180-00070"	Site #1180-00070 ready

Table 20: Detailed description of responses for command "SITE SERIALNUMBER N"

4.2.13. SITE MODULE CHECK N

Use this command to verify that expected type of programming module (defined by project) is inserted into particular programmer site (to prevent collisions between automated programming system and modules, and their damage). May be also used to determine which sites should be treated by automated programming system if not all connected sites have inserted programming module. **Modules should be checked after each project load** (see 4.2.17 SITES LOAD PROJECT).

Parameters	Mandatory	Description
N	Yes	Represents number of site (once configured by user and assigned by PG4UWMC), which the request is related to

Table 21: Parameters for command "SITE MODULE CHECK N"

Response	Description
"0"	Module Test Failure
"1"	Present

Table 22: Detailed description of responses for command "SITE MODULE CHECK N"

4.2.14. SITE ZIF SOCKET EMPTY N

Use this command to check presence of *device* in ZIF socket, typically before entering automated programming system flowchart (Figure 10: Example of automated multiprogramming operation flowchart).

Parameters	Mandatory	Description
N	Yes	Represents number of site (once configured by user and assigned by PG4UWMC), which the request is related to

Table 23: Parameters for command "SITE ZIF SOCKET EMPTY N"

Response	Description
"0"	Error: Unknown
"1"	OK: ZIF socket is empty
"2"	Error: ZIF socket is occupied
"3"	Error: ZIF socket was not tested yet
"4"	Error: Programmer can not be accessed
"5"	Error: No device is selected

Table 24: Detailed description of responses for command "SITE ZIF SOCKET EMPTY N"

4.2.15. SITE ZIF GET COUNTER VALUE N

Use this command to get actual value of a ZIF socket lifetime counter of the AP module inserted in particular site.

Parameters	Mandatory	Description
N	Yes	Represents number of site (once configured by user and assigned by PG4UWMC), which the request is related to

Table 25: Parameters for command "SITE ZIF GET COUNTER VALUE N"

Response	Description
"0"	Error: unspecified error
"1 <ActCounts> <MaxCounts>"	Read values of ZIF socket lifetime counter, where: <ActCounts> is actual value of lifetime counter <MaxCounts> is final value of lifetime counter
"2"	No AP module is detected in requested site

Table 26: Detailed description of responses for command "SITE ZIF GET COUNTER VALUE N"

4.2.16. SITE ZIF DISABLE LIFETIME WARNINGS AND GET COUNTER VALUE N

Use this command to get actual value of a ZIF socket lifetime counter of the AP module inserted in particular site and disable displaying the warning windows (for this site) if lifetime counter reaches 90% or 95% of its final value. This setting is valid until new project is load (see 4.2.13 SITES LOAD PROJECT) or until close PG4UWMC application. We recommend use this command after project load and before start operation on particular site.

Parameters	Mandatory	Description
N	Yes	Represents number of site (once configured by user and assigned by PG4UWMC), which the request is related to

Table 27: Parameters for command "SITE ZIF DISABLE LIFETIME WARNINGS AND GET COUNTER VALUE N"

Response	Description
"0"	Error: unspecified error
"1 <ActCounts> <MaxCounts>"	Warning windows are disabled and read values of ZIF socket lifetime counter, where: <ActCounts> is actual value of lifetime counter <MaxCounts> is final value of lifetime counter
"2 <ActCounts> <MaxCounts>"	Warning windows cannot be disabled. <ActCounts> is actual value of lifetime counter <MaxCounts> is final value of lifetime counter
"3"	No AP module is detected in requested site

Table 28: Detailed description of responses for command "SITE ZIF DISABLE LIFETIME WARNINGS AND GET COUNTER VALUE N"

4.2.17. SITES LOAD PROJECT

Use this command to load project into PG4UWMC. Project file name is mandatory parameter, other optional parameters are available. **After project load, we recommend to check modules in sites** (see 4.2.13 SITE MODULE CHECK N).

Parameters		Description
X	[Y]	X represents path to project file, [Y] stands for optional parameters
	/JobOperatorID:	Supply this parameter to suppress Job identification dialog, Figure 7 Applicable only for protected project file
	/JobID:	Supply this parameter to modify default JobID in Job identification dialog, Figure 7 Applicable only for protected project file
	/ProjectPassword:	Supply this parameter to suppress Authentication dialog, Figure 6 Applicable only for encrypted project file
	/UniqueID:	Supply this parameter to suppress Project file unique ID required dialog, Figure 8 Applicable only for protected project file if this feature was required by project settings

Table 29: Detailed description of parameters for command "SITES LOAD PROJECT"

Response	Description
“0”	Error: Unknown
“1”	OK: Successfully loaded
“2”	Error: Check "Use Site #1 project for all Sites" in "PG4UWMC Settings" dialog
“3”	Error: Some site(s) is busy, stop running operation before loading
“4”	Error: Master site was not correctly started. Try to do following steps in PG4UWMC: 1. disconnect sites 2. connect sites 3. load project
“5”	Error: Failure during loading process
“6”	Error: Bad project file password
“7”	Error: project load forbidden
“8”	OK: Successfully loaded, but forced default "Program" operation
“9”	Error: bad project file unique ID

Table 30: Detailed description of responses for command “SITES LOAD PROJECT”

4.2.18. SITES COMMENT

Use this command to send comment message to log window of PG4UWMC remote control software.

Parameters	Mandatory	Description
M	Yes	Represents comment message, which will be shown in PG4UWMC log window

Table 31: Parameters for command “SITES COMMENT”

4.2.19. SITE ACTUATOR STATUS N

Use this command to obtain information about actual status of particular ZIF actuator.

Response	Description
“0”	Error: actuator unit is not present
“1”	ZIF socket is opened by actuator
“2”	ZIF socket is closed by actuator
“3”	Communication error with actuator
“4”	ZIF socket actuator is in unknown position, please re-configure

Table 32: Detailed description of responses for command “SITE ACTUATOR STATUS N”

4.2.20. SITE ACTUATOR CLOSE ZIF N

Use this command to close particular ZIF socket. Only ZIF socket with enabled and configured ZIF socket actuation unit can be closed.

Parameters	Mandatory	Description
N	Yes	Represents number of site (once configured by user and assigned by PG4UWMC), which the request is related to

Table 33: Parameters for command “SITE ACTUATOR CLOSE ZIF N”

Response	Description
“0”	Error, any error occurred during closing the ZIF socket
“1”	ZIF socket is closed

Table 34: Detailed description of responses for command “SITE ACTUATOR CLOSE ZIF N”

4.2.21. SITE ACTUATOR OPEN ZIF N

Use this command to open particular ZIF socket. Only ZIF socket with enabled and configured ZIF socket actuation unit can be opened.

Parameters	Mandatory	Description
N	Yes	Represents number of site (once configured by user and assigned by PG4UWMC), which the request is related to

Table 35: Parameters for command “SITE ACTUATOR OPEN ZIF N”

Response	Description
“0”	Error, any error occurred during opening the ZIF socket
“1”	ZIF socket is opened

Table 36: Detailed description of responses for command “SITE ACTUATOR OPEN ZIF N”

4.2.22. SITES ACTUATOR CLOSE ZIFS EN BLOC

Use this command to close all ZIF sockets en bloc. Only ZIF sockets with enabled and configured ZIF socket actuation units are closed.

Response	Description
“0”	Error: any error occurred during closing the ZIF sockets
“1”	ZIF sockets are closed

Table 37: Detailed description of responses for command “SITES ACTUATOR CLOSE ZIFS EN BLOC”

4.2.23. SITES ACTUATOR OPEN ZIFS EN BLOC

Use this command to open all ZIF sockets en bloc. Only ZIF sockets with enabled and configured ZIF socket actuation units are opened.

Response	Description
“0”	Error: any error occurred during opening the ZIF sockets
“1”	ZIF sockets are opened

Table 38: Detailed description of responses for command “SITES ACTUATOR OPEN ZIFS EN BLOC”

4.2.24. SITE ACTUATOR READ CONFIGURATION N

Use this command to read configuration (position for opened and closed ZIF socket) of the ZIF socket actuation unit.

Parameters	Mandatory	Description
N	Yes	Represents number of site, which the request is related to

Table 39: Parameters for command “SITE ACTUATOR READ CONFIGURATION N”

Response	Description
“0”	Error: any error occurred during opening the ZIF sockets
“1 <OM1> <OM2> <CM1> <CM2> ”	Read configuration of ZIF socket actuation unit, where: <OM1> is position of motor #1 (upper) for opened ZIF socket <OM2> is position of motor #2 (lower) for opened ZIF socket <CM1> is position of motor #1 (upper) for closed ZIF socket <CM2> is position of motor #2 (lower) for closed ZIF socket

Table 40: Detailed description of responses for command “SITE ACTUATOR READ CONFIGURATION”

4.2.25. SITES ACTUATOR EJECT MODE START

Use this command to initiate mode where pressure plates of ZIF socket actuation units can be ejected automatically. In this mode the ZIF socket actuation units will not accept further commands not related to this mode. This mode MUST be finished by command "SITES ACTUATOR EJECT MODE STOP"!

In this mode is possible to eject pressure plate of any requested ZIF socket actuation unit. The ejecting is started by command "SITE ACTUATOR INVOKE EVENT" (from remote control application of PG4UWMC) for requested site or by touching the optical sensor on the requested site. There is possible to eject only one pressure plate in a time.

Response	Description
“0”	Error: any error
“1”	Mode is active

Table 41: Detailed description of responses for command “SITES ACTUATOR EJECT MODE START”

4.2.26. SITES ACTUATOR EJECT MODE STOP

Use this command to finish mode for automatic ejection of ZIF socket actuation unit pressure plates.

Response	Description
“0”	Error: any error
“1”	Mode is inactive

Table 42: Detailed description of responses for command “SITES ACTUATOR EJECT MODE STOP”

4.2.27. SITES ACTUATOR EJECT MODE GET STATUS

Use this command to get status of mode for automatic ejection of ZIF socket actuation unit pressure plates.

Response	Description
“negative number”	Error: communication
“0”	Mode is inactive
“1”	Mode is still active
“2”	Command to stop this mode is accepted, but mode is still active - waiting to finish mode
“3”	ZIF socket actuation unit hardware error

Table 43: Detailed description of responses for command “SITES ACTUATOR EJECT MODE GET STATUS”

4.2.28. SITES ACTUATOR INSTALL MODE START

Use this command to initiate mode where pressure plates of ZIF socket actuation units are installed automatically. In this mode the ZIF socket actuation units will not accept further commands not related to this mode. This mode MUST be finished by command "SITES ACTUATOR INSTALL MODE STOP"!

In this mode is possible to install and automatically configure pressure plate of any requested ZIF socket

actuation unit. There are two ways how the ZIF socket actuation unit can behave in this mode.

- If pressure plate was ejected before starting this mode and PG4UWMC wasn't restarted meantime, then only the installation of the pressure plate will be executed. The ZIF socket actuation unit is configured automatically immediately after the pressure plate is installed. Successful installation and configuration of the ZIF socket actuation unit should be indicated by opening and closing the ZIF socket. The ZIF socket will remain in open state after configuration.
- If pressure plate wasn't ejected before starting this mode or PG4UWMC was restarted meantime, then the pressure plate will be ejected firstly. After ejecting the pressure plate and/or replacing the AP1 programming module, the pressure plate can be installed again. The configuration of the ZIF socket actuation unit is made automatically after installation of the pressure plate. Successful installation and configuration of the ZIF socket actuation unit should be indicated by opening and closing the ZIF socket. The ZIF socket will remain in open state after configuration.

Each operation mentioned above (eject or installation of pressure plate) is started by issuing command "SITE ACTUATOR INVOKE EVENT" (from remote control application of PG4UWMC) or touching the optical sensor on the requested site. The operation is started immediately after receiving the command or detecting the touch. If a touch of the optical sensor is used to start particular operation, then software for ZIF socket actuation unit waits, before going to execute next operation, while a touch is positively detected. It's not recommended to combine command "SITE ACTUATOR INVOKE EVENT" with touching the optical sensors.

Parameters	Description
/ReadCfgFromINI	Optional parameter. Supply this parameter to force read configuration of ZIF socket actuation unit from INI file. Software will read configuration for detected AP1 programming module
/OpenPositionForUpperMotor: /OpenPositionForLowerMotor: /ClosePositionForUpperMotor: /ClosePositionForLowerMotor:	Optional parameters. Supply these parameters to suppress setting default configuration of ZIF socket actuation unit (by default values stored in software). We recommend to use command "SITES ACTUATOR READ SERVO CONFIGURATION" for already configured site to get these values.

Table 44: Parameters for command “SITES ACTUATOR INSTALL MODE START”

Examples of enabled combinations of parameters:

- no parameters ... The ZIF socket actuation unit will be configured with default values (stored in software) for detected AP1 programming module
- "/ReadCfgFromINI" ... Configuration for ZIF socket actuation unit for detected AP1 programming module will be read from INI file. If INI doesn't contain configuration for detected AP1 module, then default values from software will be used.
- "/OpenPositionForUpperMotor:XXXX /OpenPositionForLowerMotor:XXXX /ClosePositionForUpperMotor:XXXX /ClosePositionForLowerMotor:XXXX" ... Only supplied values will be used for configuration of the ZIF socket actuation unit.
- "/ReadCfgFromINI /OpenPositionForUpperMotor:XXXX /OpenPositionForLowerMotor:XXXX /ClosePositionForUpperMotor:XXXX /ClosePositionForLowerMotor:XXXX" ... Configuration for ZIF socket actuation unit for detected AP1 programming module will be read from INI file. If INI doesn't contain for detected AP1 module, then supplied values will be used.

Response	Description
“0”	Error: any error
“1”	Mode is active

Table 45: Detailed description of responses for command “SITES ACTUATOR INSTALL MODE START”

4.2.29. SITES ACTUATOR INSTALL MODE STOP

Use this command to finish mode for automatic installation of ZIF socket actuation unit pressure plates.

Response	Description
“0”	Error: any error
“1”	Mode is inactive

Table 46: Detailed description of responses for command “SITE PRESENT”

4.2.30. SITES ACTUATOR INSTALL MODE GET STATUS

Use this command to get status of mode for automatic installation of ZIF socket actuation unit pressure plates.

Response	Description
“negative number”	Error: communication
“0”	Mode is inactive
“1”	Mode is still active
“2”	Command to stop this mode is accepted, but mode is still active - waiting to finish mode
“3”	ZIF socket actuation unit hardware error

Table 47: Detailed description of responses for command “SITES ACTUATOR INSTALL MODE GET STATUS”

4.2.31. SITE ACTUATOR INVOKE EVENT N

Use this command to invoke appropriate event for ZIF socket actuation unit in mode for automatic **ejection** or for automatic **installation** of ZIF socket actuation units. This command "replaces" touching the optical sensor of ZIF socket actuation unit, for more details see help for PG4UW software.

Parameters	Mandatory	Description
N	Yes	Represents number of site, which the request is related to

Table 48: Parameters for command “SITE ACTUATOR INVOKE EVENT N”

Response	Description
“0”	Error: any error
“1”	OK, event sent to ZIF socket actuation unit

Table 49: Detailed description of responses for command “SITE ACTUATOR INVOKE EVENT N”

***Commands* must be formed into *packets* before sending.** Sending command comprises two operations, *”write of packet”* with wrapped command (request to server) and *”read of packet”* with wrapped response (answer from server). Be aware of two consecutive writes, or reads.

Example of implementation, see:
Code 7: Wrapping input data into packet, Page 33
Code 8: Extracting data from packet, Page 34
Code 9: Sending command, Page 35

4.2.32. SITE STATISTICS N

Use this command to obtain statistical information from site N, corresponding with PG4UW #N statistics.

Parameters	Mandatory	Description
N	Yes	Represents number of site (once configured by user and assigned by PG4UWMC), which the request is related to

Table 50: Parameters for command “SITE STATISTICS N”

Response
"**0**:Number|**1**:Number|**2**:Number|**3**:Number|**4**:Number|**5**:Number|**6**:Number| etc.|"

Table 51: Format of response for command “SITE STATISTICS N”

Response code	Code meaning
“0”	Unavailable statistics
“1”	Success
“2”	Module Test Failure
“3”	Insertion Test Failure
“4”	Identification Check Failure
“5”	Access Canceled By User Failure
“6”	Other Failure
“7” - “999”	Operational Fail

Table 52: Detailed description of response codes for command “SITE STATISTICS N”

Notes:

- command SITES LOAD PROJECT resets whole statistics.
- don’t use statistics commands for obtaining the results of operations, use SITE BUFFER SYNCHRONIZATION N command instead.

For example of postprocessing, see: Code 10: Postprocessing of received statistical information on Page 38

4.2.33. SITES STATISTICS

Use this command to obtain statistical information for all sites, corresponding with PG4UWMC statistics.

For description of response for this command, see:
Table 51: Format of response for command “SITE STATISTICS N” and
Table 52: Detailed description of response codes for command “SITE STATISTICS N”

4.3. Packet (Transport layer)

Packet is the basic element exchanged between server and client. Client is asking server by sending packet and receives packet as a response. Each packet is checked for integrity. If packet is not valid, negative – acknowledge is sent back and packet is dismissed. In this case, client must repeat the request to server again.

Leading byte	Data length	Data	Check sum
1 Byte	2 Bytes (little endian)	(Data length) Bytes	2 Bytes (little endian)

Table 53: Packet description

Leading byte is first byte in packet with fix value 7Eh.

Data length is Size of *Data* part of packet in bytes.

Data contains commands, parameters or returned values.

Check sum is used to detect errors that may have been introduced during transmission.
Check sum count method: *CheckSum = Data[0] + Data[1] + ... + Data[DataLength - 1]*

Example of implementation, see:
Code 6: Check sum calculation, Page 31
Code 7: Wrapping input data into packet, Page 33
Code 8: Extracting data from packet, Page 34

4.3.1. Errors

If received packet is not valid, or extracted command is not recognized, server responds with negative – acknowledge (NACK), which means error.

Type	Meaning
“NACK 1”	communication error (wrong check sum, wrong packet)
“NACK 2”	buffer size is over limit of packet

Table 54: Errors

5. Implementation of transport layer into Delphi, C++, C#

Implementation of r*emote control of PG4UWMC* is not limited to Delphi, C++, C# languages. Also others may be used, assuming that there are Windows API available to work with pipes.

5.1. Basic operations with pipes

5.1.1. Establishing connection to server, disconnecting from server

1. ***PG4UWMC*** (the listening/responding server) creates an instance of named pipe with specific name (`\\.\pipe\PG4UWMCPipeName4RemoteControl`).
2. ***Remote application*** *(*client*)* have to connect to the server by opening the pipe and getting its handle. To prevent slow responses, it's recommended to keep connection alive also when the communication is idle.
3. After all communication is done (during remote application close), client have to disconnect from server by closing handle of pipe.

```
Syntax

[Delphi]
const TXT_PIPE_NAME = '\\.\pipe\PG4UWMCPipeName4RemoteControl';

const TXT_NACK = 'NACK';
      TXT_NACK_DELIMITER = ' ';
      TXT_NACK_ERROR_1 = '1';

      PACKET_PROTECTION_SIZE = 5; // Leading byte (1B), Data length (2B), Check sum (2B)
      PACKET_SIZE = High(Word) + PACKET_PROTECTION_SIZE;
      PACKET_LEADING_BYTE = $7E;

var hPipe: THandle;


[C++]
#define TXT_PIPE_NAME "\\\\.\\pipe\\PG4UWMCPipeName4RemoteControl"

#define TXT_NACK "NACK"
#define TXT_NACK_DELIMITER " "
#define TXT_NACK_ERROR_1 "1"

#define PACKET_PROTECTION_SIZE 5 // Leading byte (1B), Data length (2B), Check sum (2B)
#define PACKET_SIZE 0xFFFF + PACKET_PROTECTION_SIZE
#define PACKET_LEADING_BYTE 0x7E

HANDLE hPipe;


[C#]
const string TXT_PIPE_NAME = "PG4UWMCPipeName4RemoteControl";

public const string TXT_NACK = "NACK";
public const string TXT_NACK_DELIMITER = " ";
public const string TXT_NACK_ERROR_1 = "1";

// Leading byte (1B), Data length (2B), Check sum (2B)
private const int PACKET_PROTECTION_SIZE = 5;
private const int PACKET_SIZE = ushort.MaxValue + PACKET_PROTECTION_SIZE;
private const byte PACKET_LEADING_BYTE = 0x7E;

private NamedPipeClientStream hPipe = null;
```

Code 1: Declarations

The pipe is bidirectional, both server and client processes can read from and write to the pipe. The client can specify GENERIC_READ and GENERIC_WRITE, when it connects to the pipe using the *CreateFile* function. Data is written to the pipe as a stream of bytes (PIPE_TYPE_BYTE).

Syntax

```
[Delphi]
function Connect: Boolean;
begin
  Result := False;

  if not WaitNamedPipe(TXT_PIPE_NAME, NMPWAIT_USE_DEFAULT_WAIT) then
    Exit;

  hPipe := CreateFile(TXT_PIPE_NAME, GENERIC_READ or GENERIC_WRITE, 0,
                      nil,
                      OPEN_EXISTING,
                      0,
                      0);

  if (hPipe = INVALID_HANDLE_VALUE) then
    Exit;

  Result := True;
end;


[C++]
bool Connect()
{
  if (!WaitNamedPipe(TXT_PIPE_NAME, NMPWAIT_USE_DEFAULT_WAIT))
    return false;

  hPipe = CreateFile(TXT_PIPE_NAME, GENERIC_READ | GENERIC_WRITE, 0,
                     NULL,
                     OPEN_EXISTING,
                     0,
                     0);

  if (hPipe == INVALID_HANDLE_VALUE)
    return false;

  return true;
}


[C#]
public bool Connect()
{
  try
  {
    hPipe = new NamedPipeClientStream(".", TXT_PIPE_NAME, PipeDirection.InOut);
    hPipe.Connect(1000);

    if (!hPipe.IsConnected)
      return false;
  }
  catch (System.Exception e)
  {
    return false;
  }
  return true;
}
```

Code 2: Opening named pipe

If the function succeeds, the return value is an open handle to the named pipe.

Syntax

```
[Delphi]
function Disconnect: Boolean;
begin
  Result := False;

  if hPipe <> INVALID_HANDLE_VALUE then
  begin
    if CloseHandle(hPipe) = False then
    begin
      Exit;
    end;

    hPipe := INVALID_HANDLE_VALUE;
  end;

  Result := True;
end;


[C++]
bool Disconnect()
{
  if (hPipe != INVALID_HANDLE_VALUE)
  {
    if (CloseHandle(hPipe) == False)
    {
        return false;
    }

    hPipe = INVALID_HANDLE_VALUE;
  }

  return true;
}


[C#]
public bool Disconnect()
{
  try
  {
    hPipe.Close();
  }
  catch (System.Exception e)
  {
    return false;
  }
  return true;
}
```

Code 3: Destroying named pipe

5.1.2. Writing to pipe, reading from pipe

Syntax

```
[Delphi]
function Write(const Buffer: string): Boolean;
var lpNumberOfBytesWritten: DWORD;
    PacketData: array[0..PACKET_SIZE - 1] of Byte;
begin
  Result := False;

  if not WrapDataToPacket(PacketData, Buffer) then
    Exit;

  if hPipe = INVALID_HANDLE_VALUE then
    Exit;

  // write packet
  if not WriteFile(hPipe, PacketData, PACKET_PROTECTION_SIZE + Length(Buffer),
                   lpNumberOfBytesWritten, nil) then
  begin
    Exit;
  end;

  Result := True;
end;


[C++]
bool Write(const AnsiString Buffer)
{
  DWORD lpNumberOfBytesWritten;
  byte PacketData[PACKET_SIZE];

  if (!WrapDataToPacket(PacketData, Buffer))
    return false;

  if (hPipe == INVALID_HANDLE_VALUE)
    return false;

   // write packet
   if (!WriteFile(hPipe, PacketData, PACKET_PROTECTION_SIZE + Buffer.Length(),
                  &lpNumberOfBytesWritten, NULL))
   {
     return false;
   }

   return true;
}


[C#]
public bool Write(string buffer)
{
  byte[] packetData = new byte[PACKET_SIZE];

  if (!WrapDataToPacket(ref packetData, buffer))
    return false;

  try
  {
    // write packet
    hPipe.Write(packetData, 0, PACKET_PROTECTION_SIZE + buffer.Length);
  }
  catch(System.Exception e)
  {
    return false;
  }
  return true;
}
```

Code 4: Writing to named pipe

Syntax

```
[Delphi]
function Read(var Buffer: string): Boolean;
var lpNumberOfBytesRead: DWORD;
    PacketData: array[0..PACKET_SIZE - 1] of Byte;
begin
  Result := False;
  Buffer := '';

  if hPipe = INVALID_HANDLE_VALUE then
    Exit;

  // read packet
  if not ReadFile(hPipe, PacketData, PACKET_SIZE, lpNumberOfBytesRead, nil) then
  begin
    Exit;
  end;

  if not UnwrapDataFromPacket(Buffer, PacketData) then
  begin
    Buffer := TXT_NACK + TXT_NACK_DELIMITER + TXT_NACK_ERROR_1;
    Exit;
  end;

  Result := True;
end;


[C++]
bool Read(AnsiString &Buffer)
{
  DWORD lpNumberOfBytesRead;
  byte PacketData[PACKET_SIZE];
  Buffer = "";

  if (hPipe == INVALID_HANDLE_VALUE)
    return false;

   // read packet
   if (!ReadFile(hPipe, PacketData, PACKET_SIZE, &lpNumberOfBytesRead, NULL))
   {
     return false;
   }

   if (!UnwrapDataFromPacket(Buffer, PacketData))
   {
     Buffer = TXT_NACK + TXT_NACK_DELIMITER + TXT_NACK_ERROR_1;
     return false;
   }

   return true;
}
[C#]
public bool Read(ref string buffer)
{
  byte[] packetData = new byte[PACKET_SIZE];
  buffer = "";

  try
  {
    // read packet
    hPipe.Read(packetData, 0, PACKET_SIZE);
  }
  catch (System.Exception e)
  {
    return false;
  }

  if (!UnwrapDataFromPacket(ref buffer, packetData))
  {
    buffer = TXT_NACK + TXT_NACK_DELIMITER + TXT_NACK_ERROR_1;
    return false;
  }

  return true;
}
```

Code 5: Reading from named pipe

5.2. Other codes

Syntax

```
[Delphi]
function CountCheckSum(const Data: array of Byte; const Length: Word): Word;
var i: integer;
begin
  Result := 0;

  for i := 0 to Length - 1 do
    Result := Result + Data[i];
end;

[C++]
WORD CountCheckSum(const byte Data[], const WORD Length)
{
  WORD Result = 0;

  for (int i = 0; i < Length; i++)
    Result += Data[i];
  return Result;
}

[C#]
private ushort CountCheckSum(byte[] data, ushort length)
{
  ushort result = 0;

  for (int i = 0; i < length; i++)
    result += (ushort)data[i];
  return result;
}
```

Code 6: Check sum calculation

Syntax

```
[Delphi]
function WrapDataToPacket(var PacketData: array of Byte; const Buffer: string): Boolean;
var CheckSum: Word;
    i, iLength: integer;
begin
  Result := False;

  // check capability for PacketData length range
  iLength := Length(Buffer);
  if iLength > High(Word) then
    Exit;

  // set leading byte
  PacketData[0] := PACKET_LEADING_BYTE;

  // set data length
  PacketData[1] := Lo(iLength);
  PacketData[2] := Hi(iLength);

  // copy data of packet
  for i := 0 to iLength - 1 do
    PacketData[3 + i] := Byte(Buffer[i + 1]);

  // count check sum
  CheckSum := CountCheckSum(PacketData, 3, iLength);

  // set check sum
  PacketData[3 + iLength] := Lo(CheckSum);
  PacketData[3 + iLength + 1] := Hi(CheckSum);

  Result := True;
end;

[C++]
bool WrapDataToPacket(byte PacketData[], const AnsiString Buffer)
{
  // check capability for PacketData length range
  int iLength = Buffer.Length();
  if (iLength > 0xFFFF)
    return false;

  // set leading byte
  PacketData[0] = PACKET_LEADING_BYTE;

  // set data length
  PacketData[1] = LOBYTE(iLength);
  PacketData[2] = HIBYTE(iLength);

  // copy data of packet
  for (int i = 0; i < iLength; i++)
    PacketData[3 + i] = (Byte)Buffer[i + 1];

  // count check sum
  WORD CheckSum = CountCheckSum(PacketData, 3, iLength);

  // set check sum
  PacketData[3 + iLength] = LOBYTE(CheckSum);
  PacketData[3 + iLength + 1] = HIBYTE(CheckSum);

  return true;
}
```

```
[C#]
private bool WrapDataToPacket(ref byte[] packetData, string buffer)
{
  // check capability for PacketData length range
  int iLength = buffer.Length;
  if (iLength > ushort.MaxValue)
    return false;

  // set leading byte
  packetData[0] = PACKET_LEADING_BYTE;

  // set data length
  packetData[1] = (byte)(iLength & 0xFF);
  packetData[2] = (byte)(iLength >> 8);

  // copy data of packet
  for (int i = 0; i < iLength; i++)
    packetData[3 + i] = (byte)(buffer[i]);

  // count check sum
  ushort checkSum = CountCheckSum(packetData, 3, (ushort)iLength);

  // set check sum
  packetData[3 + iLength] = (byte)(checkSum & 0xFF);
  packetData[3 + iLength + 1] = (byte)(checkSum >> 8);

  return true;
}
```

Code 7: Wrapping input data into packet

Syntax

```
[Delphi]
function UnwrapDataFromPacket(var Buffer: string; const PacketData: array of Byte): Boolean;
var CheckSumFromPacket: Word;
    i, iLength: integer;
begin
  Result := False;
  Buffer := '';

  // check packet integrity
  if PacketData[0] <> PACKET_LEADING_BYTE then
    Exit;

  iLength := (PacketData[2] shl 8) + PacketData[1];
  CheckSumFromPacket := (PacketData[3 + iLength + 1] shl 8) + PacketData[3 + iLength];
  // compare packet check sum with calculated
  if CheckSumFromPacket <> CountCheckSum(PacketData, 3, iLength) then
    Exit;

  // copy data
  for i := 0 to iLength - 1 do
    Buffer := Buffer + Char(PacketData[3 + i]);

  Result := True;
end;

[C++]
bool UnwrapDataFromPacket(AnsiString &Buffer, const byte PacketData[])
{
  Buffer = "";

  // check packet integrity
  if (PacketData[0] != PACKET_LEADING_BYTE)
    return false;

  int iLength = (PacketData[2] << 8) + PacketData[1];
  WORD CheckSumFromPacket = (PacketData[3 + iLength + 1] << 8) + PacketData[3 + iLength];
  // compare packet check sum with calculated
  if (CheckSumFromPacket != CountCheckSum(PacketData, 3, iLength))
    return false;

  // copy data
  for (int i = 0; i < iLength; i++)
    Buffer = Buffer + (Char)PacketData[3 + i];

  return true;
}

[C#]
private bool UnwrapDataFromPacket(ref string buffer, byte[] packetData)
{
  buffer = "";

  // check packet integrity
  if (packetData[0] != PACKET_LEADING_BYTE)
    return false;

   int iLength = (packetData[2] << 8) + packetData[1];
   ushort checkSumFromPacket = (ushort)((packetData[3 + iLength + 1] << 8) +
                                        packetData[3 + iLength]);
   // compare packet check sum with calculated
   if (checkSumFromPacket != CountCheckSum(packetData, 3, (ushort)iLength))
     return false;

   // copy data
   for (int i = 0; i < iLength; i++)
     buffer = buffer + (char)packetData[3 + i];

   return true;
}
```

Code 8: Extracting data from packet

Syntax

```
[Delphi]
function SendCommand(const Command: string; var Answer: string): Boolean;
begin
  Result := False;

  if Write(Command) then
    if Read(Answer) then
    begin
      Result := True;
      Exit;
    end
end;


[C++]
bool SendCommand(const AnsiString Command, AnsiString &Answer)
{
  if (Write(Command))
    if (Read(Answer))
    {
      return true;
    }

  return false;
}


[C#]
private bool SendCommand(string command, ref string answer)
{
  if (Write(command))
    if (Read(ref answer))
    {
      // successful finish
      return true;
    }

  return false;
}
```

Code 9: Sending command

Syntax

```
[Delphi]
function ReportStatistics(Text: string): string;
  var
    iIndexInText, iStatisticValue: Integer;
    sStatisticItem: string;
    iStatisticSuccess, iStatisticAdapterTestFailure, iStatisticInsertionTestFailure,
    iStatisticIDCheckFailure, iStatisticAccessCanceledByUserFailure,
    iStatisticOtherFailure, iStatisticOperationalFail: Integer;
  begin
    iStatisticSuccess := 0;
    iStatisticAdapterTestFailure := 0;
    iStatisticInsertionTestFailure := 0;
    iStatisticIDCheckFailure := 0;
    iStatisticAccessCanceledByUserFailure := 0;
    iStatisticOtherFailure := 0;
    iStatisticOperationalFail := 0;

    repeat
      iIndexInText := Pos('|', Text);
      if (iIndexInText > 0) then
      begin
        sStatisticItem := Copy(Text, 1, iIndexInText - 1);
        Delete(Text, 1, iIndexInText);

        iIndexInText := Pos(':', sStatisticItem);
        if (iIndexInText > 0) then
        begin
          iStatisticValue := StrToInt(Copy(sStatisticItem, iIndexInText + 1,
                                     Length(sStatisticItem)));
          case StrToInt(Copy(sStatisticItem, 1, iIndexInText - 1)) of
            0: begin
                 Result := ' Unavailable statistics.';
                 Exit;
               end;
            1: iStatisticSuccess := iStatisticValue;
            2: iStatisticAdapterTestFailure := iStatisticValue;
            3: iStatisticInsertionTestFailure := iStatisticValue;
            4: iStatisticIDCheckFailure := iStatisticValue;
            5: iStatisticAccessCanceledByUserFailure := iStatisticValue;
            6: iStatisticOtherFailure := iStatisticValue;
          else
            iStatisticOperationalFail := iStatisticOperationalFail + iStatisticValue;
          end;
        end;

      end;
    until Text = '';

    Result := ' Success:' + IntToStr(iStatisticSuccess) +
              '  Oper. failure(s):' + IntToStr(iStatisticOperationalFail) +
              '  Other failure(s):' + IntToStr(iStatisticAdapterTestFailure +
                                               iStatisticInsertionTestFailure +
                                               iStatisticIDCheckFailure +
                                               iStatisticAccessCanceledByUserFailure +
                                               iStatisticOtherFailure) +
              '  Total:' + IntToStr(iStatisticSuccess +
                                    iStatisticAdapterTestFailure +
                                    iStatisticInsertionTestFailure +
                                    iStatisticIDCheckFailure +
                                    iStatisticAccessCanceledByUserFailure +
                                    iStatisticOtherFailure +
                                    iStatisticOperationalFail);
  end;
```

```
[C++]
AnsiString TFormVUH::ReportStatistics(AnsiString Text)
{
        int iIndexInText, iStatisticValue;
        AnsiString sStatisticItem;
        int iStatisticSuccess, iStatisticAdapterTestFailure, iStatisticInsertionTestFailure,
            iStatisticIDCheckFailure, iStatisticAccessCanceledByUserFailure,
            iStatisticOtherFailure, iStatisticOperationalFail;

        iStatisticSuccess = 0;
        iStatisticAdapterTestFailure = 0;
        iStatisticInsertionTestFailure = 0;
        iStatisticIDCheckFailure = 0;
        iStatisticAccessCanceledByUserFailure = 0;
        iStatisticOtherFailure = 0;
        iStatisticOperationalFail = 0;

        do
        {
          iIndexInText = Text.Pos('|');
          if (iIndexInText > 0)
          {
                sStatisticItem = Text.SubString(1, iIndexInText - 1);
                Text.Delete(1, iIndexInText);

                iIndexInText = sStatisticItem.Pos(':');
                if (iIndexInText > 0)
                {
                  iStatisticValue = StrToInt(sStatisticItem.SubString(iIndexInText + 1,
                                             sStatisticItem.Length()));
                  switch (StrToInt(sStatisticItem.SubString(1, iIndexInText - 1)))
                  {
                        case 0: return " Unavailable statistics.";
                        case 1: iStatisticSuccess = iStatisticValue;
                                break;
                        case 2: iStatisticAdapterTestFailure = iStatisticValue;
                                break;
                        case 3: iStatisticInsertionTestFailure = iStatisticValue;
                                break;
                        case 4: iStatisticIDCheckFailure = iStatisticValue;
                                break;
                        case 5: iStatisticAccessCanceledByUserFailure = iStatisticValue;
                                break;
                        case 6: iStatisticOtherFailure = iStatisticValue;
                                break;
                        default:iStatisticOperationalFail = iStatisticOperationalFail +
                                                            iStatisticValue;
                        break;
                  }
                }
          }
        } while (!Text.IsEmpty());

        return " Success:" + IntToStr(iStatisticSuccess) +
                  "  Oper. failure(s):" + IntToStr(iStatisticOperationalFail) +
                  "  Other failure(s):" + IntToStr(iStatisticAdapterTestFailure +
                                                   iStatisticInsertionTestFailure +
                                                   iStatisticIDCheckFailure +
                                                   iStatisticAccessCanceledByUserFailure +
                                                   iStatisticOtherFailure) +
                  "  Total:" + IntToStr(iStatisticSuccess +
                                        iStatisticAdapterTestFailure +
                                        iStatisticInsertionTestFailure +
                                        iStatisticIDCheckFailure +
                                        iStatisticAccessCanceledByUserFailure +
                                        iStatisticOtherFailure +
                                        iStatisticOperationalFail);
}
```

```
[C#]
string ReportStatistics(string text)
{
    int iIndexInText, iStatisticValue;
    string sStatisticItem;
    int iStatisticSuccess, iStatisticAdapterTestFailure, iStatisticInsertionTestFailure,
        iStatisticIDCheckFailure, iStatisticAccessCanceledByUserFailure,
        iStatisticOtherFailure, iStatisticOperationalFail;

    iStatisticSuccess = 0;
    iStatisticAdapterTestFailure = 0;
    iStatisticInsertionTestFailure = 0;
    iStatisticIDCheckFailure = 0;
    iStatisticAccessCanceledByUserFailure = 0;
    iStatisticOtherFailure = 0;
    iStatisticOperationalFail = 0;

    do
    {
        iIndexInText = text.IndexOf('|');
        if (iIndexInText > -1)
        {
            sStatisticItem = text.Substring(0, iIndexInText);
           text = text.Remove(0, iIndexInText + 1);

            iIndexInText = sStatisticItem.IndexOf(':');
            if (iIndexInText > -1)
            {
                iStatisticValue = Convert.ToInt32(sStatisticItem.Substring(iIndexInText + 1,
                                        sStatisticItem.Length - (iIndexInText + 1)));
                switch (Convert.ToInt32(sStatisticItem.Substring(0, iIndexInText)))
                {
                       case 0: return " Unavailable statistics.";
                       case 1: iStatisticSuccess = iStatisticValue;
                               break;
                       case 2: iStatisticAdapterTestFailure = iStatisticValue;
                               break;
                       case 3: iStatisticInsertionTestFailure = iStatisticValue;
                               break;
                       case 4: iStatisticIDCheckFailure = iStatisticValue;
                               break;
                       case 5: iStatisticAccessCanceledByUserFailure = iStatisticValue;
                               break;
                       case 6: iStatisticOtherFailure = iStatisticValue;
                              break;
                       default:iStatisticOperationalFail = iStatisticOperationalFail +
                                                           iStatisticValue;
                              break;
                }
            }

        }
    } while (text != "");

    return " Success:" + iStatisticSuccess.ToString() +
           "  Oper. failure(s):" + iStatisticOperationalFail.ToString() +
           "  Other failure(s):" + (iStatisticAdapterTestFailure +
                                    iStatisticInsertionTestFailure +
                                    iStatisticIDCheckFailure +
                                    iStatisticAccessCanceledByUserFailure +
                                    iStatisticOtherFailure).ToString() +
           "  Total:" + (iStatisticSuccess +
                         iStatisticAdapterTestFailure +
                         iStatisticInsertionTestFailure +
                         iStatisticIDCheckFailure +
                         iStatisticAccessCanceledByUserFailure +
                         iStatisticOtherFailure +
                         iStatisticOperationalFail).ToString();
}
```

Code 10: Postprocessing of received statistical information

6. Version history

Ver. 1.17 (Jan, 2023)
- remote control protocol version updated to 1.17 (since PG4UWMC 3.81w)
- Main Checksum information added to command PROGRAMMING ENVIRONMENT chapter 4.2.3

Ver. 1.14 - 1.16 skipped

Ver. 1.13 (Sept, 2019)
- remote control protocol version updated to 1.13 (since PG4UWMC 3.52e)
- added new command SITE BUFFER SYNCHRONIZATION N chapter 4.2.9

Ver. 1.12 (Aug, 2019)
- remote control protocol version updated to 1.12 (since PG4UWMC 3.51r)

Ver. 1.11 (Jan, 2019)
- remote control protocol version updated to 1.11 (since PG4UWMC 3.47)
- added new command PROGRAMMING ENVIRONMENT chapter 4.2.3

Ver. 1.10 (Dec, 2018)
- remote control protocol version updated to 1.10 (since PG4UWMC 3.45j)
- added new command SITE STATISTICS N chapter 4.2.32
- added new command SITES STATISTICS chapter 4.2.33

Ver. 1.09 (Jan, 2018)
- added new command SITE ZIF GET COUNTER VALUE N chapter 4.2.15
- added new command SITE ZIF DISABLE LIFETIME WARNINGS AND GET COUNTER VALUE N chapter 4.2.16

Ver. 1.08 (May, 2016)
- added comment 5 on page 12
- modification of Figure 10: Example of automated multiprogramming operation flowchart

Ver. 1.07 (July, 2013)
- improved response speed for command SITE START on BeeHive204AP-AU (since PG4UWMC 2.99)

Ver. 1.06 (May, 2013)
- remote control protocol version updated to 1.06 (since PG4UWMC 2.97q)
- added new Actuator related commands

Ver. 1.05 (October, 2012)
- remote control protocol version updated to 1.05 (since PG4UWMC 2.92)
- added new Actuator related commands

Ver. 1.04 (June, 2012)
- remote control protocol version updated to 1.04 (since PG4UWMC 2.89v)
- added new command SITE ZIF SOCKET EMPTY N chapter 4.2.14

Ver. 1.03 (April 4, 2012)
- remote control protocol version updated to 1.03 (since PG4UWMC 2.88)
- modification of Figure 10: Example of automated multiprogramming operation flowchart, correction of FIRST_PASS_OF_SITE initialization
- added new command SITES COMMENT chapter 4.2.18
- enhanced description of commands SITE MODULE CHECK N chapter 4.2.13, SITES LOAD PROJECT chapter 4.2.17
- added some details to chapter 5.1.1 Establishing connection to server, disconnecting from server
- modified example and description of Code 2: Opening named pipe

Ver. 1.02 (March 21, 2012)
- remote control protocol version updated to 1.02 (since PG4UWMC 2.87m)

- added new parameters for command “SITES LOAD PROJECT X [Y]” to suppress particular dialogs, see 29, and Figure 3
- changed structure of Commands (Application layer) chapter 4.2
- changed responses for command “SITES LOAD PROJECT X [Y]”, some codes added, see 30,
- Removed Figure: PG4UWMC warning dialog “The loaded project hasn't defined default operation...”, dialog is suppressed and following new response for command is returned instead:
  “8” - OK: Successfully loaded, but forced default "Program" operation, see 30
- changed title in Figure 8 to Project file unique ID required dialog
- modified Example of automated multiprogramming operation flowchart, added variable FIRST_PASS_OF_SITE

Ver. 1.01 (February 10, 2012)
- remote control protocol version updated to 1.01 (since PG4UWMC 2.77)
- added new commands “APPLICATION CLOSE”, “SITES LOAD PROJECT” and “SITE ACTUATOR STATUS”
- added option to start PG4UWMC remote control by command line parameters /autoconnectremotecontrol, /donotloadproject
- modified structure of document

Ver. 1.00 (October 21, 2011)
- preliminary version updated to normal version
- added new commands “SITES STOP” and “SITE MODULE CHECK N”

Ver. 0.93 (September 28, 2011)
- modification of Figure 10: Example of automated multiprogramming operation flowchart

Ver. 0.92 (July 29, 2011)
- added new chapter about Virtual universal handler (VUH) mode activation in PG4UWMC

Ver. 0.91 (April 18, 2011)
- added sample source code in C++, C# languages
- added new chapter 2Virtual Universal Handler (VUH) – example

Ver. 0.90 (March 25, 2011)
- Initial release, preliminary