Remote control of PG4UW, user's manual

Version: 1.0
Date: March 8th 2005

Preface

There exist more reason why to control the programmer not by PG4UW software directly, but from other software. Therefore we provide a remote controlling capability of PG4UW software.

The remote control feature allows to be PG4UW software flow controlled by other application – either using BAT file commands or using DLL file. This manual describes basic features of remote control capability of PG4UW control program and also explain how to implement the remote control functions available in the PG4UW *remotelb.dll* library file.

Contents

I. Basic facts about remote control

Remote control of PG4UW control program allows to control some functions of PG4UW application by other application. This is very suitable feature for integrating ELNEC programmer to mass-production handler system or other useful application.

Remote control main features are:

1. Remote control philosophy is:
   - remote application that controls PG4UW acts as Server
   - PG4UW acts as Client
2. the communication between PG4UW and remote Server is realized by set of commands available in DLL library remotelb.dll
3. communication is asynchronous and it uses events to handle received messages from PG4UW
4. the order of starting PG4UW and Server application is not important but better way is to start the Server application as the first and PG4UW (Client) the second
5. communication between PG4UW and remote control program is made via TCP protocol - this allows the PG4UW to be installed on one computer and remote control application to be installed on another computer, and these computers will be connected together via network
6. remote control library remotelb.dll is written in Borland® Delphi Pascal and is usable by Borland® Delphi Pascal but also by other C/C++ environments

Default TCP communication settings for remote control are:

Port: **telnet**   Address: **127.0.0.1** or **localhost**

Address setting applies for PG4UW (Client) only. Port setting applies for PG4UW (Client) and also for Server application.

Default settings allows to use remote control on one computer (address localhost). PG4UW (Client) and remote control Server have to be installed on the same computer.

Note: If **firewall** is installed on system, firewall can display warning message when remote control Server or Client is starting. When firewall is showing warning with question asking to allow or deny network access for remote Server or Client, please select 'Allow' option, otherwise remote control will not work. Of course you can specify in firewall options more strict rights to allow remote Server/Client access on specified address and port only.

II. Published functions in library remotelb.dll

Following description presents purpose of each function. For more particular description of function declarations and parameters, please see the file RemoteCtrl.pas.

If the remote Server application is written in C language, there is necessary to write appropriate header .h file to make functions from library remotelb.dll available in C/C++ project.
The Pascal unit file RemoteCtrl.pas shows function declarations and parameters. This can be used for writing C/C++ header file.

a) general functions for Client/Server remote control connection establishing and connection parameters setting

To see the usage of following functions and procedures, see the example application PG4UWrem, especially Pascal unit remoteform.pas.

procedure **CreateClientAndMakeConnectionToServer**(vProcessProc: TProcWithPChar;
vWriteToLogProc: TProcWithPChar;
vPort, vAddr: PChar);

Procedure creates Client communication object, with defined parameters and tries to connect to Server. This procedure is used internally in PG4UW (Client) application. Server application should not use this procedure.

procedure **CreateServerAndMakeListenToClients**(vProcessProc: TProcWithPChar;
vWriteToLogProc: TProcWithPChar;
onClientConnectProc: TProc;
onClientDisconnectProc: TProc;
vPort: PChar);

Procedure creates Server communication object, with defined parameters and starts waiting for Client(s). This procedure is used by Server application - remote control program.

Input parameters are:
vProcessProc: TProcWithPChar - define pointer to procedure which will be called every time Server receives message(s) from Client.

vWriteToLogProc: TProcWithPChar - define pointer to procedure which is useful especially during debugging of Server program. Procedure is called when any of basic Client-Server communication events occures.
Communication events are: connect/disconnect Client-Server, send message to Client, receive message from Client. Procedure can contain user defined write to memo or log window of Server application.

onClientConnectProc: TProc - procedure is called as event when Client is connected to Server

onClientDisconnectProc: TProc - procedure is called as event when Client is disconnected from Server

vPort - defines port for TCP communication (default is 'telnet')

Server does not have address defined itself. Internally Server has defined address 0.0.0.0, which means, that Server accepts Clients from all interfaces.

procedure **MakeClientConnectionToServer**(FailedErrDisplay: boolean);

Procedure tries to connect Client to Server. The procedure is used internally in PG4UW (Client) application. Server application should not use this procedure.

procedure **MakeClientDisconnectionFromServer**;

Procedure tries to disconnect Client from Server. The procedure is used internally in PG4UW (Client) application.
Server application should not use this procedure.

procedure **MakeClientServertCloseConnectionAndFree**;

Procedure is used for Client and Server applications to close connection and free TCP Client/Server communication object.
The Client and Server applications call this routine automatically when they closed.

procedure **SendOperationResultToServer**(op_result: TOpResultForRemote);

Procedure is used by Client applications for sending messages to Server. Server application should not use this procedure.

procedure **SendLineToServer**(line: PChar);

Procedure is used by Client applications for sending messages to Server. Server application should not use this procedure.

function **ClientServerObjectExists**: boolean;

Function returns true, if Client/Server object already exists, otherwise returns false. Function does not test connection status.

function **ClientConnectionIsClosed**: boolean;

Function returns true, if Client connection status is 'Closed', otherwise returns false. Function is used by PG4UW (Client) application
Server application should not use this procedure.

function **ServerHasConnectedClient**: boolean;

Function is used by Server application. Function returns true, if Client is connected to Server, otherwise returns false.

procedure **EnableWriteEventsToLog**(value: boolean);

Procedure is used by Client and Server applications. The purpose of the procedure is to block calls of procedure vWriteToLogProc defined
as parameter of procedures CreateClientAndMakeConnectionToServer and CreateServerAndMakeListenToClients.

function **GetCurrentPort**: PChar;

Function is used by Client and Server applications. Function returns the value of current port (for example '2020', 'telnet', etc.).

function **GetCurrentAddr**: PChar;

Function is used by Client applications. Function returns the value of current address (for example

'localhost' or '127.0.0.1',
'192.168.0.10', ...). For Server the function returns always value '0.0.0.0'.

---

b) procedures used for sending basic types of commands from remote Server to PG4UW (Client)

procedure **SEND_CMD_BringToFront**;

Procedure is used to send message 'bringtofront' to Client. When Client receives the message, it tries to make BringToFront
operation. BringToFront operation is basically activation of Client application window.

procedure **SEND_CMD_BlankCheckDevice**;

Procedure is used to send message 'blankcheck' to Client. When Client receives the message, it starts device Blank check operation. In the end of operation Client sends operation result to Server. Operation result command received from client can be processed by Server application in procedure defined by pointer parameter vProcessProc in procedure CreateServerAndMakeListenToClients.
Client PG4UW can accept and start 'blankcheck' command only when no operation is currently running in PG4UW. If PG4UW is running some operation, the 'blankcheck' command will be ignored. To receive the current status of PG4UW, use command SEND_CMD_GetProgStatus described later.

procedure **SEND_CMD_ReadDevice**;

Procedure is used to send message 'readdevice' to Client. When Client receives the message, it starts device Read operation. In the end of operation Client sends operation result to Server. Other properties of PG4UW behaviour are same as for command procedure SEND_CMD_BlankCheckDevice.

procedure **SEND_CMD_VerifyDevice**;

Procedure is used to send message 'verifydevice' to Client. When Client receives the message, it starts device Verify operation. In the end of operation Client sends operation result to Server. Other properties of PG4UW behaviour are same as for command procedure SEND_CMD_BlankCheckDevice.

procedure **SEND_CMD_ProgramDevice**;

Procedure is used to send message 'programdevice' to Client. When Client receives the message, it starts device Program operation. In the end of operation Client sends operation result to Server. Other properties of PG4UW behaviour are same as for command procedure SEND_CMD_BlankCheckDevice.

procedure **SEND_CMD_EraseDevice**;

Procedure is used to send message 'erasedevice' to Client. When Client receives the message, it starts device Erase operation. In the end of operation Client sends operation result to Server. Other properties of PG4UW behaviour are same as for command procedure SEND_CMD_BlankCheckDevice.

procedure **SEND_CMD_RepeatLastDevOperation**;

Procedure is used to send lastly used device operation command to Client. For example if lastly used command is SEND_CMD_ProgramDevice, the call of procedure SEND_CMD_RepeatLastDevOperation will be the same as call of SEND_CMD_ProgramDevice.

procedure **SEND_CMD_Stop**;

Procedure is used to send message 'stopoperation' to Client. When Client receives the message, it stops current device operation. If no operation is running, the 'stopoperation' command does nothing. Other function of 'stopoperation' is closing message window(s) in Client application.

procedure **SEND_CMD_SelectDevice**(devmanuf, devname: PChar);

Procedure is used to send message 'selectdevice:' to Client. When Client receives the message, it tries to select specified device. Device is specified by parameters devmanuf and devname. Parameters are not case sensitive. The Client sends select device result to Server. The result command received from client can be processed by Server application in procedure defined by pointer parameter vProcessProc in procedure CreateServerAndMakeListenToClients.

Example: To select device Intel TE28F160C3B [TSOP48] call

```
  SEND_CMD_SelectDevice('Intel', 'TE28F160C3B [TSOP48]');
```

procedure **SEND_CMD_EPROMFLASH_AutoSelect**(pinsnumber: PChar);

Procedure is used to send message 'autoseldevice:' to Client. When Client receives the message, it starts autoselect device operation. Parameter pinsnumber can be used to specify the pin number of device which helps more reliable detection of device type. If parameter pinsnumber is blank Pchar (''), autoselect operation tries to detect inserted device pin number automatically. The Client sends currently selected device to Server but not result of autoselect detection success.

procedure **SEND_CMD_LoadProject**(prjname: PChar);

Procedure is used to send message 'loadproject:' to Client. When Client receives the message, it tries to load project file specified by parameter prjname. The Client sends load project result to Server. The result command received from client can be processed by Server application in
procedure defined by pointer parameter vProcessProc in procedure CreateServerAndMakeListenToClients.

procedure **SEND_CMD_LoadFile**(fname: PChar);

Procedure is used to send message 'loadfile:' to Client. When Client receives the message, it tries to load file specified by parameter fname. The Client sends load file result to Server. The result command received from client can be processed by Server application in procedure defined by pointer parameter vProcessProc in procedure CreateServerAndMakeListenToClients.

procedure **SEND_CMD_GetProgStatus**;

Procedure is used to send message 'getprogstatus' to Client. When Client receives the message, it sends its current status info to Server. The status info command received from Client can be processed by Server application in procedure defined by pointer parameter vProcessProc in procedure CreateServerAndMakeListenToClients.
Status info contains four basic info status data:
- busy status
- current device operation type
- current device operation progress
- current device operation result

For more information see the example unit

procedure **SEND_CMD_Quest_is_Client_Ready**;

Procedure sends question to Client, if the Client is Ready. Client sends ready yes/no status message to Server. The ready status command received from Client can be processed by Server application in procedure defined by pointer parameter vProcessProc in procedure CreateServerAndMakeListenToClients. Client is ready when no device operation is currently running and main Client's window is not hidden by any modal dialogs in Client.

procedure **SEND_CMD_PROCESS_CMDLINE**(cmdparams: PChar);

Procedure is used to send message 'cmdlineparams:' to Client. When Client receives the message, it starts processing of received parameters. The procedure is used for internal purposes. Please do not use the SEND_CMD_PROCESS_CMDLINE.

c) constants used by remote control communication

Following constants define command codes and status codes used by remote communication in the direction Client -> Server. These constants can be tested by Server application in receive message Event handle procedure by which Server can recognize which commands Client sent to Server.
For practical use of the constants see the example Server application Pascal source code.

Constants are defined in unit RemoteCtrl.pas

```
//-------------------------------------
//----- commands Client -> Server  -----
TCP_KEYWORD_OPTYPE = 'optype';
TCP_KEYWORD_OPBUSY = 'opbusy';
TCP_KEYWORD_PROGRESS = 'progress';
TCP_KEYWORD_OPRESULT = 'opresult';
// codes for operation results
TCP_CMD_OPRESULT       = 'opresult:';
TCP_KEYWORD_OPRESULT_GOOD  = 'oprGood';
TCP_KEYWORD_OPRESULT_FAIL  = 'oprFail';
TCP_KEYWORD_OPRESULT_HWERR = 'oprHWError';
TCP_KEYWORD_OPRESULT_NONE  = 'oprNone';

// codes for Load file/project result
TCP_CMD_LOAD_FILE_PRJ_RESULT   = 'loadresult:';
TCP_FILE_LOAD_GOOD            = 'frgood';
TCP_FILE_LOAD_ERROR           = 'frerror';
TCP_FILE_LOAD_CANCELLED       = 'frcancelled';

// codes for Select device result
TCP_CMD_SELECT_DEVICE_RESULT   = 'selectdeviceresult:';
TCP_SELECT_DEVICE_GOOD        = 'good';
TCP_SELECT_DEVICE_ERROR       = 'error';

// codes for Auto Select of EPROM/FLASH device
TCP_CMD_AUTSEL_EPRFLSH_RESULT  = 'autoseldeviceresult:'; // not used yet

// codes for server to client 'ready' question
TCP_CMD_CLIENT_READY_ANSWER = 'clienprogramisreadyanswer';
TCP_KEY_CLIENT_READY_YES    = 'isready';
TCP_KEY_CLIENT_READY_NO     = 'isnotready';

// codes for command line params result
TCP_CMD_PROCESS_CMDL_RESULT    = 'cmdlineparamsresult:';
TCP_CMD_PROCESS_CMDL_GOOD      = 'good';
TCP_CMD_PROCESS_CMDL_ERROR     = 'error';
```

III. Short example of remote control implementation

There are two basic ways how to use remote control:
- synchronous mode
- asynchronous mode

Synchronous mode

In synchronous mode when Server is sending any command to Client, Server will wait in cycle for Client response.
The Server jumps inside cycle until:
1. Server receives wished message from Client or
2. Server receives 'user break' request

Asynchronous mode

In asynchronous mode when Server is sending any command to Client, Server will not wait in cycle for Client response. The Server waits for Client messages indirectly by using Events. Procedure which handles messages from Client must be defined in parameter vProcessProc when creating Server communication object by calling procedure CreateServerAndMakeListenToClients.

Events are used both in synchronous and asynchronous modes. Procedure which handles messages from Client must be defined in parameter vProcessProc when creating Server communication object by calling procedure CreateServerAndMakeListenToClients.

To create remote control Server application, remote control functions have follow order as described bellow:

1. create Server object for remote TCP communication and start listening

```
CreateServerAndMakeListenToClients(ServerProcessProc,
                                   WriteToLogwindowProc,
                                   onClientConnectProc,
                                   onClientdisconnectProc,
                                   PChar(remote_ctrl_settings.Client_Server_Port));
```

2. run PG4UW Client application

3. when Server receives message that client is connected, Server can send commands SEND_CMD_...

It is recommended to ask Client if it is ready by **SEND_CMD_Quest_is_Client_Ready**.

If Client responds answer TCP_KEY_CLIENT_READY_YES it means Client is ready to receive executive commands (for example Load project, Program device and so on).

4. when closing Server application, call the **MakeClientServertCloseConnectionAndFree** procedure

Two examples of remote control Server implementation are enclosed.

1. **PG4UWrem.dpr** – windows application showing the usage of remote control features with windows, buttons etc.
2. **PG4UWcmd.dpr** – console application showing synchronous usage of remote control and usage of command line parameters

IV. Remote command line control of PG4UW

PG4UW accept set of commands from the command line (command line parameters). The remote control can be achieved also by this command line parameters, but more efficient way is to use special tool **PG4UWcmd.exe**, which has many advantages. The main advantage is size of the **PG4UWcmd**, which result the calling of **PG4UWcmd** results a much faster response than calling of PG4UW directly.

Program PG4UWcmd.exe can be used to:

1. start PG4UW application with specified command line parameters
2. force command line parameters to PG4UW that is already running

Very good feature of PG4UWcmd.exe is its return code according to command line parameters operation result in PG4UW.

Return values of PG4UWcmd.exe

If the command line parameters processed in PG4UW were successful, the Exitcode (or ErrorLevel) of PG4UWcmd.exe is zero. Otherwise the ExitCode value is number 1 or more.

Return value of program PG4UWcmd.exe can be tested in batch files.

Following executive command line parameters are available to use with PG4UWcmd.exe

/Prj:<file_name>

Loads project file. Parameter <file_name> means full or relative project file path and name.

/Loadfile:<file_name>

Loads file. Parameter <file_name> means full or relative path to file that has to be loaded. File format is detected automatically

/Program[:switch]

Forces start of "Program device" operation automatically when program is starting, or even if program is already running. Also one of following optional switches can be used:

1. switch 'noquest' forces start of device programming without question

2. switch 'noanyquest' forces start of device programming without question and after operation on device is completed, program doesn't show "Repeat" operation dialog and goes directly into main program window.

Examples:
1. /Program
2. /Program:noquest
3. /Program:noanyquest

/Close

This parameter has sense together with /Program parameter only, and makes program PG4UW to close automatically after device programming is finished (no matter if operation was successful or not).

/Eprom_Flash_Autoselect[:xx]

Forces automatic select EPROM or FLASH by ID when program is starting or even if program is already running. xx means pins number of device in ZIF (this time are valid 28 or 32 pins only) and it is required just for older programmers without insertion test capability. For others programmers is the value ignored.

Basic rules for using of executive command line parameters:

1. program PG4UWcmd.exe must be located in the same directory as program PG4UW.exe
2. if PG4UW.exe is not running when PG4UWcmd.exe is called, it will be automatically started
3. command line parameters are not case sensitive
4. command line parameters can be used when first starting of program or when program is already running
5. if program is already running, then any of command line operation is processed only when program was not busy (no operation was currently executing in program). Program must be in basic state, i.e. main program window focused, no modal dialogs displayed, no menu commands opened or executed
6. order of processing command line parameters when using more parameters together is defined firmly as following:
   step1 Load file (/Loadfile:...)
   step2 Load project (/Prj:...)
   step3 EPROM/FLASH autoselect
   step4 Program device (/Program[:switch])
   step5 Close of control program (/Close only together with parameter /Program)

Examples:

Example 1:

PG4UWcmd.exe /program:noanyquest /loadfile:c:\empfile.hex

Following operations will perform:
1. start PG4UW.exe (if not already running)
2. load file c:\empfile.hex
3. start program device operation without questions
4. PG4UWcmd.exe is still running and periodically checking status of PG4UW.exe
5. when device programming completes, PG4UWcmd.exe is closed and is returning ExitCode depending on load file and device programming results in PG4UW.exe. When all operations were successful, PG4UWcmd.exe returns 0, otherwise returns value 1 or more.

Example 2:

PG4UWcmd.exe /program:noanyquest /prj:c:\emproject.eprj

The operations are the same as in Example 1, just Load file operation is replaced by Load project file c:\emproject.eprj command.

Example 3:

Using PG4UWcmd.exe in batch file and testing return code of PG4UWcmd.exe.

```
rem ------- beginning of batch ------------------------
@echo off

rem Call application with wished parameters
PG4UWcmd.exe /program:noanyquest /prj:c:\emproject.eprj

rem Detect result of command line execution
rem Variable errorlevel is tested, value 1 or greater means the error occured

if errorlevel 1 goto FAILURE

echo Command line operation was successful
goto BATCHEND

:FAILURE
echo Command line operation error(s)

:BATCHEND
```

```
echo.
echo This is end of batch file (or continue)
pause
rem ------- end of batch -------------------------
```

V. System requirements

System requirements for remote control program are:

Minimum PC system requirements:

- Microsoft Windows® 98, 2000, XP or later
- PC Pentium 133
- 16 MB of RAM
- 10 MB of free disk space

Recommended PC system requirements:

- Microsoft Windows® XP Professional
- PC Pentium III 600
- 128 MB of RAM
- 10 MB of free disk space

System requirements for application PG4UW when using remote control function are:

Minimum PC system requirements:

- Microsoft Windows® 98, 2000, XP or later
- PC Pentium II 300
- 64 MB of RAM
- 80 MB of free disk space
- LPT printer port (for programmers connected via LPT port)
- USB port ver. 1.1 or later (for programmers connected via USB port)

Recommended PC system requirements:

- Microsoft Windows® XP Professional
- PC Pentium III 800
- 256 MB of RAM
- 80 MB of free disk space
- LPT printer port supporting EPP/ECP modes (for programmers connected via LPT port)
- USB port ver. 1.1 or later (for programmers connected via USB port)

Common requirements:

Both remote control program and PG4UW program need network adapter with TCP protocol support installed. The network adapter can be virtual (Microsoft loopback adapter) or real network adapter (network card with proper drivers installed).

When using remote control program and PG4UW program on the same computer, there is no need to have real network adapter (network card) installed.

When using remote control program and PG4UW program on the remote computers, real network adapter (network card) has to be installed on each computer.

Installation of virtual network adapter (example for Windows XP):

1. click **Start** menu
2. select **Settings** item
3. select **Control panel**
4. in Control panel window click on **Add Hardware**
5. in **Add Hardware Wizard** dialog click on button Next
6. on question “Have you already connected this hardware on computer?” select option “Yes, I have

already connected the hardware" and click on button Next

7. in the list of available hardware select the item "Add a new hardware device" and click on button Next
8. on question "What do you want the wizard to do?" select option "Install the hardware that I manually select from list (Advanced)" and click button Next
9. from the list "Common hardware types" select item "Network adapters" and click button Next
10. from left panel "Manufacturer" select "Microsoft"
11. from right panel "Network adapter" select "Microsoft Loopback adapter" and click button Next
12. on the confirmation click button Next again
13. the adapter is installed

Now you can configure the adapter to support TCP protocol. Configuration is made in the same way as for real network cards.

END of documentation