

Gobetwino 0.5 user manual.

What is Gobetwino?.....	3
Requirements, limitations and credits.....	4
Installing Gobetwino.....	4
The interface.	5
Commands and parameters.	5
Special command types.....	6
Communication between Arduino and Gobetwino.....	7
The command editor.	7
How to send commands from Arduino sketches.	12
Command type descriptions.....	14
Start program return id command type (SPRID).	15
Start program wait for exit command type (SPWEX).	16
Send keys command type (SENDK).....	17
Download file command type (DLFIL).	18
Read file line command type (RFLIN).	19
Log to file command type (LGFIL).	20
Copy file command type (CPFIL).	21
Send mail command type (SMAIL).....	22
Ping command type (PING).....	23
Time command type (T).	24
Date command type (D).	25
Receiving e-mails in Gobetwino.....	26
Settings.....	28
The status and log settings tab.	28
The process settings tab.	29
The mail settings tab.	30
The serial port settings tab.	31
Appendix A syntax for SENDK command.....	33

What is Gobetwino?

Gobetwino is kind of a “generic proxy” for Arduino. It’s a program running on a PC (Windows only – sorry), that will act on behalf of Arduino and do some of the things that Arduino can’t do on its own. So it works like a go between, hence the name.

Gobetwino can be downloaded here: <http://www.mikmo.dk/gobetwino>. This is also the place to watch for updates and info.

The current version is to be considered a beta version, it probably still has bugs in it, and no long time stability test has been performed yet. If you have comments, questions or whatever, mail me at gobetwino@mikmo.dk I can’t guarantee that every mail will receive an answer but I will try.

Gobetwino is not restricted to work with Arduino, the serial port can be configured so it can work with most devices that can send and receive strings over a serial line.

Gobetwino is listening on the serial port, for “commands” coming from Arduino, and in response it will do something for Arduino and possibly return something to Arduino.

Gobetwino defines a set of command types that can be used as templates to create actual commands. Arduino can ask Gobetwino to execute these commands, and return something to Arduino.

So what can Gobetwino do? Using the defined command types you can create commands in Gobetwino that Arduino can ask Gobetwino to execute. These commands can:

- Start a program on the PC.

- Start a program, and wait until it finishes, and tell Arduino it finished.

- Send data to any windows program from Arduino, like it was typed on the keyboard.

- Send email, optionally with an attached file.

- Download a file from the internet.

- Read a file and return data to Arduino.

- Log data from Arduino to a file, with an optional timestamp.

- Periodically check a POP3 mailbox for incoming mails and send commands from the mail to Arduino.

- Get the time from the PC.

- Get the date from the PC.

- Ping a host or IP address.

- Copy a file on the PC.

With combinations of these commands you can do things like:

- Start any program on your PC, either directly or via an associated file type.

Start Excel, send sensor readings from Arduino directly into the Excel sheet, save the sheet and email it to someone, without touching your PC.

Send e-mails to a POP3 mailbox and have Arduino react to the contents of the emails.

Log data directly to a CSV file on the PC, so the data can be used in spreadsheets or databases.

Download a file from the internet and have Arduino ask for a specific line of data from the file.

Of course there is no magic involved, and Gobetwino only defines and do things on the PC side, you have to program anything needed on the Arduino side. But programming Arduino to send commands to Gobetwino is a matter of sending specially formatted strings over the serial line, and then wait for responses from Gobetwino.

Requirements, limitations and credits.

Gobetwino requires that you have the Microsoft .net 2.0 runtime installed on your PC. If needed you can download it from Microsoft's web site. Of course you also need to have the FTDI drivers for the virtual serial port for Arduino installed, but you probably have them already if you have worked with Arduino.

Gobetwino is developed and checked on a PC running Windows XP servicepack 3. It has not yet been tested under Windows Vista or Windows 7. Some of the functionality, especially the command that "sends keypresses" to programs might not work under Windows Vista due to security restrictions. I have not had the opportunity to check this yet.

The POP3 functionality in the program is implemented using the fabulous open source Indy Sockets from the Indy project <http://www.indyproject.org/index.en.aspx>. If you ever have to implement any internet protocol functionality in a program under linux or Windows you should seriously consider checking it out.

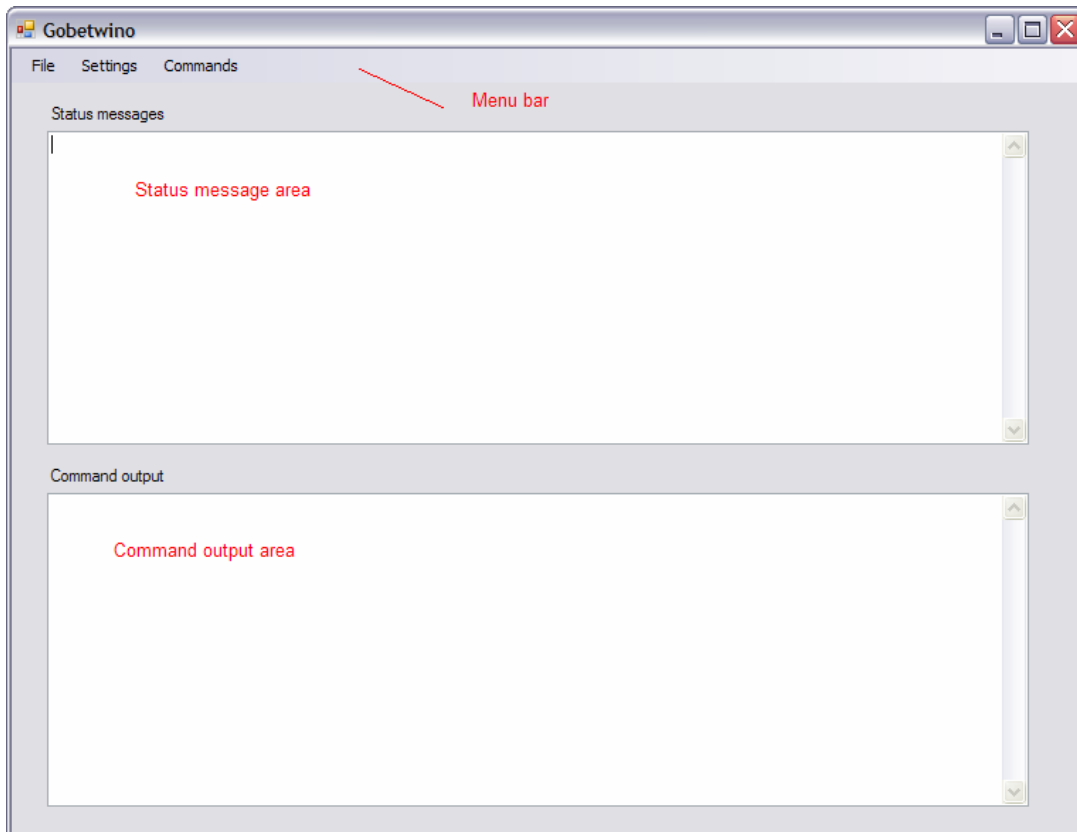
Installing Gobetwino.

Installing Gobetwino is easy. All you need to do is unzip the zip file to a folder. You might want to create a desktop shortcut to the Gobetwino.exe file. I suggest you put it in C:\gobetwino, as this will make using the sample code easier.

The interface.

The Gobetwino user interface is quite simple, there's a main window and two dialogs for creating/editing commands, and configuring the settings for Gobetwino.

The main window looks like this:



It consists of a menu bar and two large text areas, one for input (commands) from Arduino and status messages, and one that shows the output from the commands, i.e. what is send back to Arduino. These two text areas are really just for monitoring what is going on.

The menu bar has three menus, the File menu that only has an Exit menu item, the Settings menu that displays the settings dialog and the Command menu the displays the Command editor.

Commands and parameters.

Prepare for some techno babble, but be assured that using things is much easier than it sounds. You just need to understand a couple of concepts.

As stated earlier Gobetwino defines a set of command types that can be used to as templates to create commands. Each command type has a name that is an abbreviation of its function. The

difference between a command type and a command can best be explained with an example. The Log to file command type (LGFIL) defines a command type that has three parameters, File, Data and Timestamp. This allows you to create commands that will log some specific data to a specific file with an optional timestamp. There is only one LGFIL command type but you can create as many commands based on this type as you want, each having the same type, but different names. So you can create one LGFIL type command with the name LOGDATA that logs sensor readings from Arduino to a file called temp-log.txt, without timestamps, and another LGFIL type command called LOGGPS that logs GPS data from Arduino to a file called GPS-log.txt with timestamps.

So the command type defines an action and a set of parameters that each has a data type. But a command is an actual implementation of that action with specific parameters.

This sounds complicated, but there is a command editor that makes it super simple to create and edit commands in Gobetwino.

Most, but not all command types in Gobetwino have parameters, and there are two kinds of parameters. This is because not all parameters can be defined when the command is created. Let's look at the Log to file command again. Assume you have set up your Arduino with a temperature sensor and you want to log the temperature to a file on the PC every 5 minutes with a timestamp for each log entry. When you create the command, you can tell Gobetwino to use c:\temp-log.txt as the log file, and that there should be a time stamp with each log entry. But you can't tell Gobetwino what temperature to log, because that is not known yet, but coming from Arduino later. So a placeholder for the data is created. So Gobetwino only knows that each time Arduino ask to have the command executed some data should be written to the file c:\temp-log.txt together with a timestamp. The actual data is coming from Arduino together with the request to execute the command.

The parameters you can define when the command is created are called internal parameters, and the parameters that come from Arduino when the command is executed are called external parameters.

In the command editor you specify the internal parameters and the external parameters just have some placeholders starting with a \$ sign followed by a number. You can't do anything with them, but when you write your Arduino sketch you will supply actual values for the external parameters.

Some command types only have internal parameters that can all be defined when the command is created, some have both internal and external parameters. A single command type has only external parameters. And finally there are two command types that have no parameters at all.

Another important thing is that Gobetwino does not know anything about the data it is logging, it just knows that each time the command is executed it should append a line to the file c:\temp-log.txt with some data (a string actually) received from Arduino, and that a timestamp should be put in there as well. Gobetwino does not know that the data is temperature readings it only sees a string, nothing else.

Special command types.

Gobetwino has three special command types, the Time command type (T), the Date command type (D) and the Send keys command type (SKEYS). These command types are special because they do

not have any internal parameters, and therefore they do not have anything that you need to specify in the command editor. So those three commands are already made for you, you can use them from Arduino without doing anything. See the section where each command type is explained, for instructions on how to use them.

Communication between Arduino and Gobetwino.

The communication between Arduino and Gobetwino takes place over the usual serial line. For this to work the serial line has to be configured with the same settings in Arduino and Gobetwino. See the chapter on Settings later in this manual.

All communication is done by sending and receiving strings or single characters. It is up to you to convert any data you want to send from Arduino to Gobetwino to a string, this includes the commands, and any data required as parameters to the commands. The opposite is also true, you have to convert anything received from Gobetwino to the required data types in your Arduino sketch. See the Arduino sample sketches for possible solutions to this. Just remember that as far as Gobetwino goes every bit of data send and received is a string or a char.

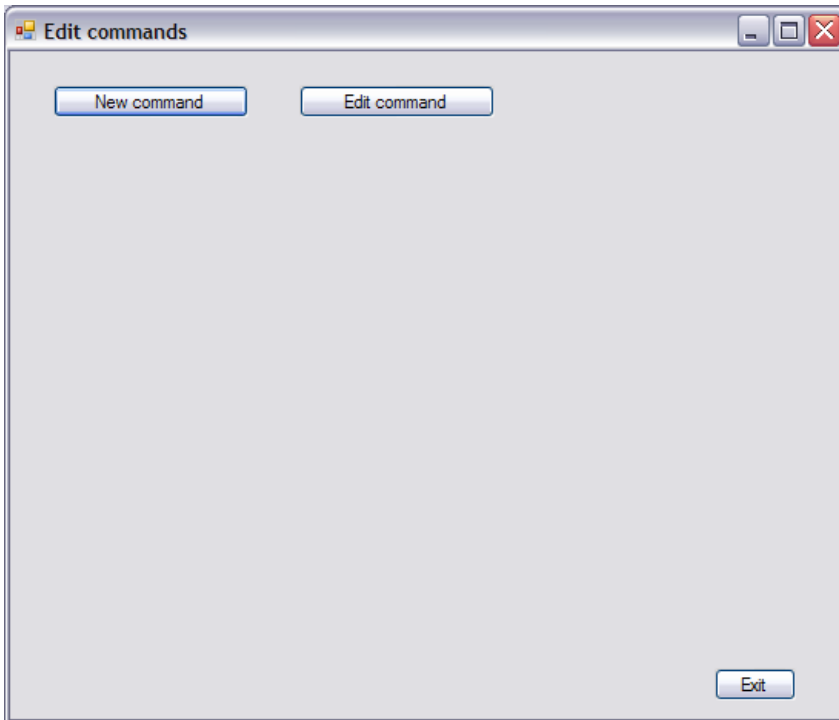
Another important point is that Gobetwino always sends and reads whole lines of data. So when you send command strings from Arduino to Gobetwino you should always use the `Serial.println()` function. You will often need to send a command and it's parameters from Arduino by using several `Serial.print(ln) ()` calls. If you do that, only the last one should be `Serial.println()`, all the others should be `Serial.print()`. Gobetwino is constantly listening to the serial port, but only reads data when there is something in the buffer that ends with `CR + LF`. This is exactly what the `Serial.println()` method sends. If this sounds confusing just check the Arduino sample sketches and you will get it.

Some of the command types return data to Arduino, and some just returns a "result code", indicating the result of executing the requested command. This is explained in detail under the description of each command type later.

Ok, enough techno babble, let's see some real stuff.

The command editor.

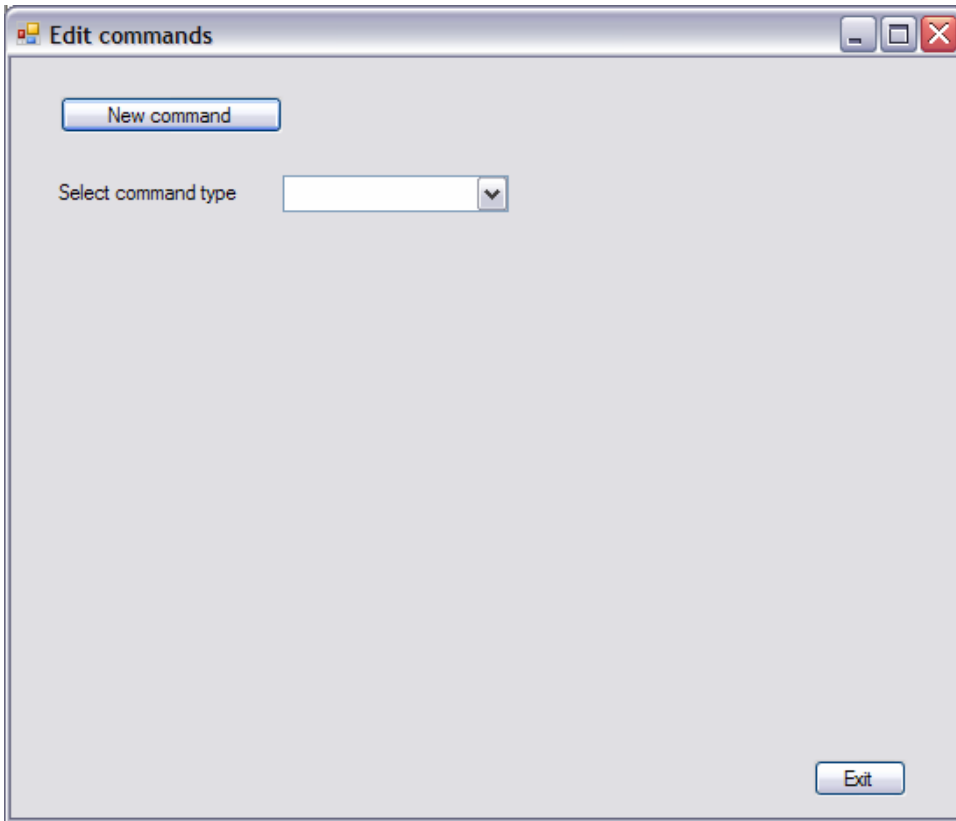
When you click the Command menu, the Command editor appears, looking like this:



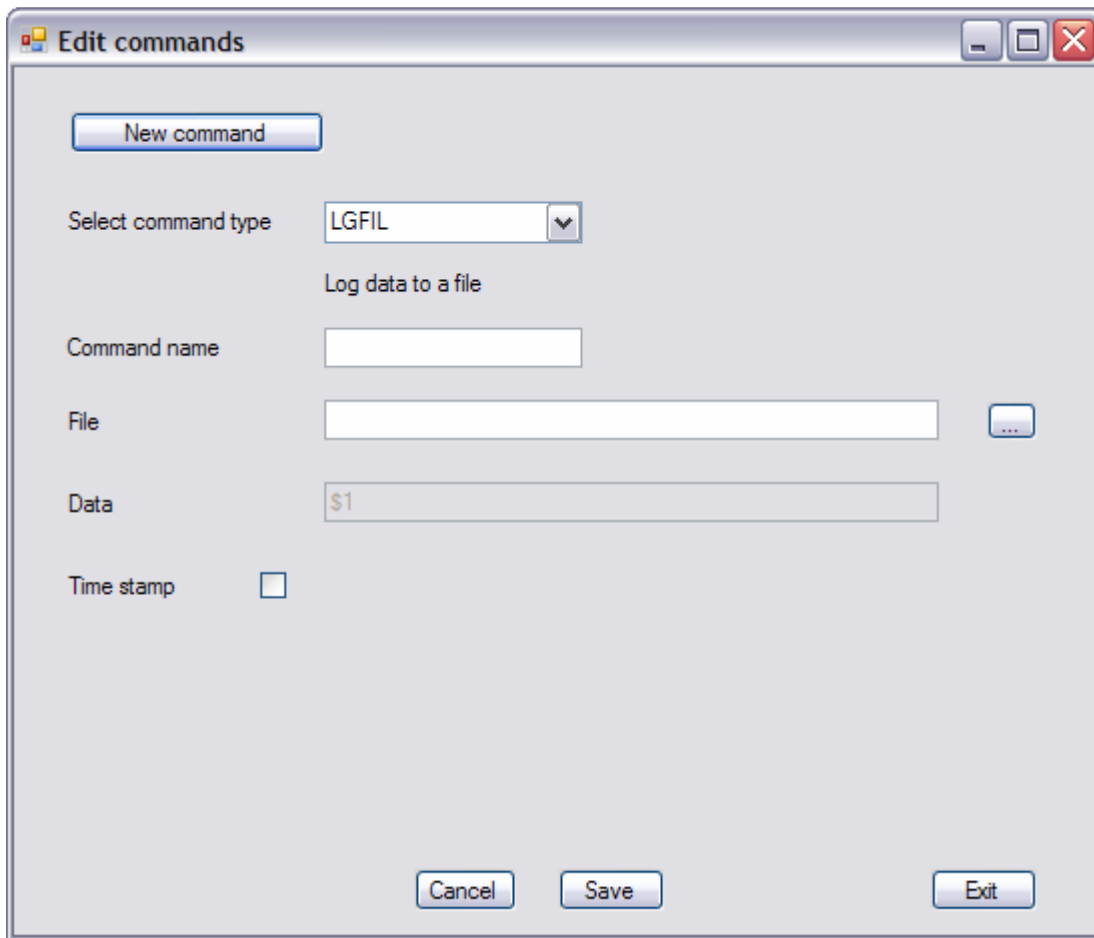
As you can see you have the options to create a new command, edit an existing command or close the editor with the Exit button.

Let's hang on to the Log to file example, and create a new Log to file command.

Click the New command button, now a drop down box with all the available command types appear:



From the select command type drop down, select the Log to file command type (LGFIL)

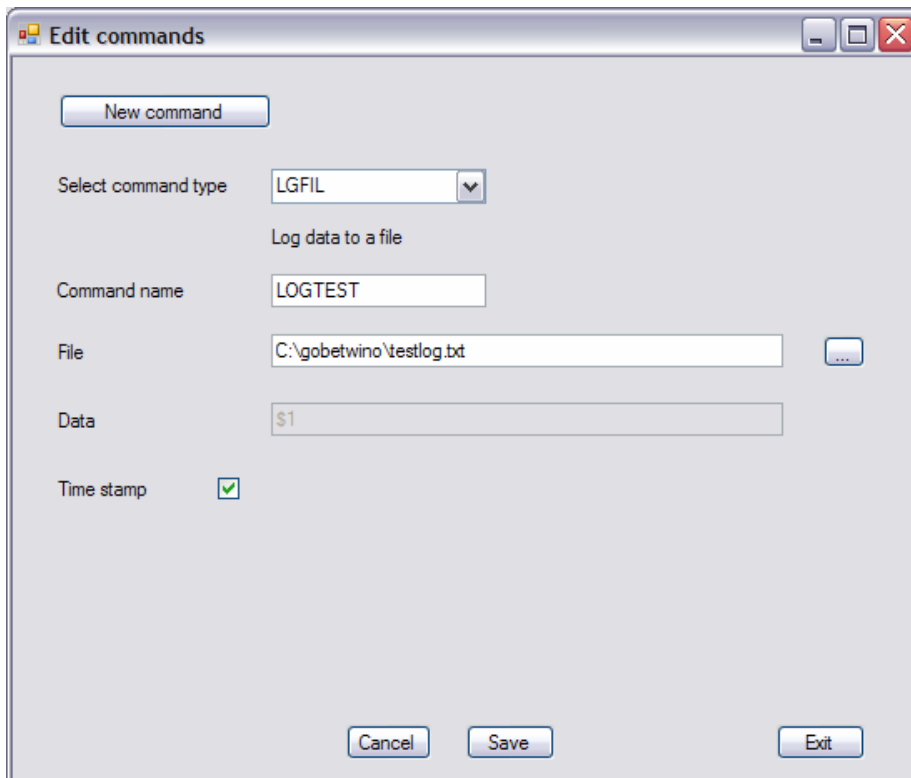


Now fields for all the parameters for the LGFIL command type appears.

All commands you create must have a name that consists of 1 to 10 alpha numeric characters. This is the name you will use in your Arduino sketch when you refer to this command. The name will automatically be capitalized when you type it in. Let's call this command LOGTEST.

Below the name field you can see the three fields for the parameters for the LGFIL command type. Two internal parameters (File and Time stamp) and one external (Data), already filled in with the placeholder \$1, and disabled so you can't type in the field.

To create a Log to file command you must already have a log file prepared, this is just an empty .txt file, you can create it with notepad, or whatever tool you prefer. Just make sure you have a file ready. Use the browse button to the right of the File textbox, to browse to your log file. I will use the file **c:\gobetwino\testlog.txt** for this example



Check the Time stamp checkbox to tell Gobetwino that you want a timestamp on the beginning of every line in the log file.

Click the save button, and you are done.

Congratulations, you have just created your first Gobetwino command.

When you click the save button Gobetwino will try to validate the contents of the parameter fields. It will check that files and folders entered exists, that email addresses are at least in the correct format, and that values that should be integers actually are integers etc.

If something is wrong the command will not be saved and you will be notified, so you can correct the mistake.

To edit a command, just click the “Edit command” button, select the command from the drop down box, now the parameters of the command will be shown in the same user interface as when it was created. Make the changes you need, and click the “Save” button. You can not change the name of a command.

How to send commands from Arduino sketches.

Ok I lied a little, there's a little more techno babble.

To use a command that you have created in Gobetwino from your Arduino sketch, you have to send a string from Arduino to Gobetwino in a special format. This is to ensure that Gobetwino can make sure that the command received is complete and has all the parameters needed with it.

The general format of a command string is:

```
#S|commandname|[parameter1&parameter2&...&last parameter]#
```

The string to send ALWAYS starts with the hash mark # to indicate the "start of command, and it always ends with a hash mark to indicate the end of command #. This also means that the hash mark CAN NEVER be a part of anything inside a command, this goes for the command name and for any data in the command's parameters. The same goes for the pipe symbol | and the ampersand &. These three characters are used to indicate to Gobetwino where various parts of the command start and end, if you put them in your command or its parameters it will mess up Gobetwino's ability to make sense of the command.

After the first # there is always a capital S, don't worry about it, just make sure it is always there. Then follows a pipe symbol | then the command name, IN CAPITAL LETTERS, this should be the name you gave the command when it was created in Gobetwino. Command names can only contain capital letters and the digits 0-9, nothing else. Then another pipe symbol | and all the EXTERNAL parameters (the ones with the \$ names in the command editor) for the command enclosed in square brackets [and] and separated with ampersands &. You do not send the INTERNAL parameters from Arduino. If a command has no external parameters, the square brackets should still be there, just with nothing between them.

So to use our log to file command we created and assuming we want to log the value 877 to the log file, the string to send should be this:

```
#S|LOGTEST|[877]#
```

Remember that you have to convert the number 877 to a string. Arduino code to send the above command could be:

This could be a sensor reading or whatever, but for this example it's just an arbitrarily chosen integer value.

```
int value = 877;  
char buffer[5];
```

```
Serial.print("#S|LOGTEST|");  
Serial.print(itoa((value), buffer, 10));  
Serial.println("]#");
```

First we send the string #S|LOGTEST|
Then we send the parameter converted to a string
And finally we send the string]# WITH THE Serial.println() METHOD.

Two important things to note in the code above:

First, notice the use of the itoa() function. Itoa() is a function that will take an integer and return it converted to its string representation. This is an easy way to convert numbers to strings in Arduino code. It's out of this manual's scope to explain the details of the function. Google it if you need to have it explained. There is also an atoi() function that does the opposite.

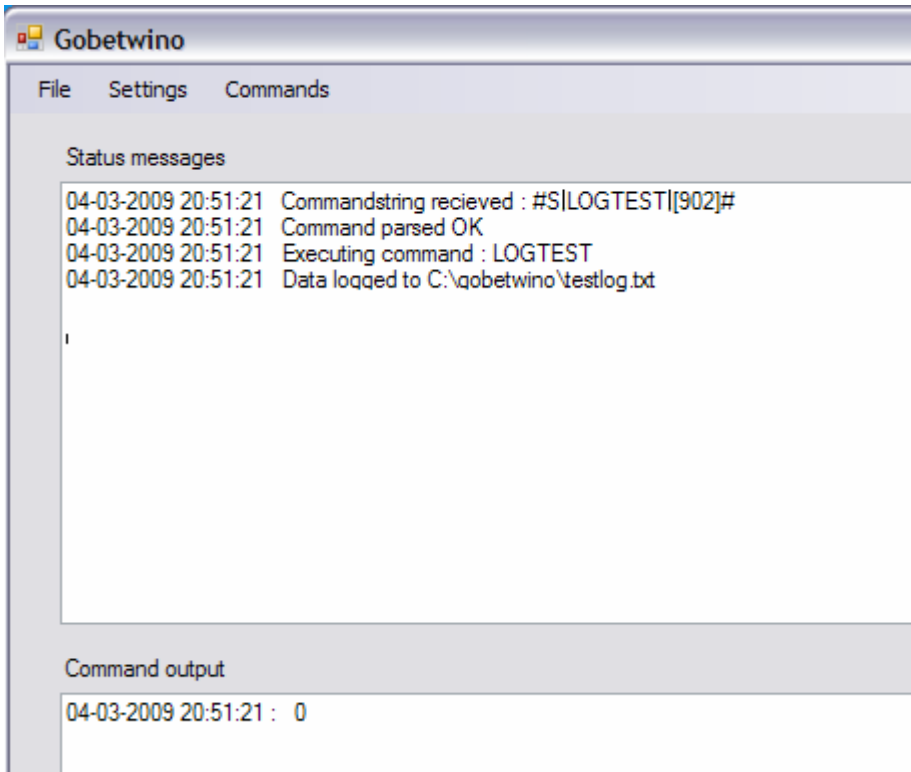
These functions are easy to use but they do add quite some bytes to the size of your Arduino sketch though.

Second, notice that first we use Serial.print() two times and only the last part of the command is send with Serial.println() THIS IS IMPORTANT. This way Gobetwino will receive the whole command as one single line.

When Gobetwino receives the Command string, it performs a series of checks to make sure that the command exists, and the parameters are correct etc. etc. and then execute the function. If you have configured the settings in Gobetwino to show all status messages (the default), you will see a number of lines of text in the status text area. These lines tell you something about what's going on and also tells you if something is wrong, for instance if you send a command string from Arduino that Gobetwino can not understand. Always look here to check that everything is going as expected.

The log to file command returns a result code to Arduino 0 if everything is ok, otherwise a negative number. Everything that is send back to Arduino is also printed in the command output text area.

Here's a screenshot of what is displayed after executing the LOGTEST command once:



As you can see everything that is displayed in the Status message area and command output area is prefixed with a time stamp so you can see what is going on and when. You can also see in the command output area that 0 was returned to Arduino as a response to executing the command. This number indicates “Ok, everything is good”. For all the possible result codes refer to the description of the individual command types.

The status message area can be configured to only show errors, but for troubleshooting it is a good idea to show everything.

(EDITORS NOTE: this feature will first be available in the next version 😊)

Command type descriptions.

This chapter will explain all the available command types in detail, including their parameters and what they return.

Any dependencies on the settings you can configure in Gobetwino are also described.

There will also be a small Arduino sketch for each, showing how it can be used.

The Arduino sketches may not be the smartest or most useful in the world, they should just be seen as a simple way to demonstrate how the different command types work.

Start program return id command type (SPRID).

This command type will start a program on the PC and Gobetwino will keep a handle to the started program, and return that handle to Arduino. This handle can be used in your Arduino sketch later, to send “keystrokes” to the program. (See the SENDK command). But you don’t have to use it.

The command can start any executable file, like .exe, .bat or .cmd files. It can also start a program by “running” a file which file type is associated with a program in Windows. For instance you could start the Notepad program in Windows by having the SPRID command run either Notepad.exe or any file with a .txt extension.

If the started program takes any command line arguments you can specify them as well.

NOTE:

In the current version there is no way for Arduino to check if a process with a given process ID has ended.

Parameters:

Program path	Internal parameter. This is the fully qualified path to the program / file to start.
Cmd line args.	Internal parameter. This is any command line arguments you want to pass to the started program.

Return value:

0 – maxProcesses -1	If the program started a number between 0 and maxProcesses-1 is returned, this is the process ID.
-1	If maxProcesses programs has already been started.
-2	If the file in the Program path parameter could not be found.
-3	If an exception was generated trying to start the program.

Settings dependencies:

You can not start more concurrent programs than the value of maxProcesses. This limit can be configured in the settings dialog.

How to use in Arduino sketch:

Serial.println(“#S|NAME|[]# “) where NAME is the command name as specified in the command editor.

Arduino sample sketch:

gobetwinoXLtest.pde

Start program wait for exit command type (SPWEX).

This command type will start a program, and then it will wait until the program exits (ends). This command type is useful if you want to start a program on your PC from Arduino, and let the program do something and then let Gobetwino notify Arduino when the program has ended.

Its intended use is for relatively short tasks that Arduino need to know has been done on the PC. This Command type does not return a process ID to Arduino like the SPRID command type does.

The Timeout parameter indicates how long Gobetwino will wait for the started program to finish. If this time is exceeded the killMaxWaitExceedProcesses setting determines what happens. If this setting is true the process will be ended by Gobetwino, if it is false the process will be left running. The maxWaitForExit setting is the default timeout value, and it is used if Timeout is 0. This is to ensure that Gobetwino will not be waiting forever for a started program to finish.

NOTE:

This command type is blocking. Gobetwino can not do anything else while waiting for the started program to finish.

NOTE 2:

This command type is well suited to execute bat or cmd files on the PC, but any executable file can be run.

Parameters:

Program path	Internal parameter. This is the fully qualified path to the program / file to start.
Cmd line args.	Internal parameter. This is any command line arguments you want to pass to the started program.
Timeout	Internal parameter. The time in milliseconds to wait for the program to finish.

Return value:

0	If the program was started and finished within the timeout period.
-1	If the file in the Program path parameter could not be found.
-2	If the program did not end within the timeOut period
-3	If an exception was generated trying to start the program.

Setting dependencies:

This command type is depending on the killMaxWaitExceedProcesses setting and the maxWaitForExit setting. See description above.

How to use in Arduino sketch:

Serial.println("#S|NAME|[]# ") where NAME is the command name as specified in the command editor.

Arduino sample sketch:

Spwextest.pde

Send keys command type (SENDK).

This command type is one of the special commands that don't have any internal parameters. So there is nothing to specify in the command editor. There is a pre made command of this type that you can use without doing anything.

This command can simulate sending keystrokes to a program started with the SPRID command. The program will react just as if the keys were typed directly on the PC keyboard. This is an extremely powerful command, because you can literally control most aspects of almost any Windows program with it.

You can send single keystrokes, strings of characters, and special keys. See appendix A for a complete description.

NOTE:

You can only send keystrokes to a program started with the SPRID command, because a process ID is needed. And you should not send large amounts of text very fast, this can cause problems.

Parameters:

Process Id	External parameter. The Process ID for the program to send keystrokes to. This program must be started with the SPRID command
Data	External parameter. This is the characters (keystrokes) to send.

Return value:

0	If the keystrokes was successfully sent to the program.
-1	If the process ID is not valid.
-2	If an exception was generated trying to send the keystrokes.

Setting dependencies:

None.

How to use in Arduino sketch:

`Serial.println("#S|SENDK|[PID&keystrokes to send]#");` Where PID is the process ID of the program that the keystrokes is send to, and "keystrokes to send" is a string with the keystrokes (see appendix A for details).

Arduino sample sketch:

gobetwinoXLtest.pde

Download file command type (DLFIL).

This command can be used to download a file from the Internet.

NOTE:

Some firewall and security software might be configured to prevent programs on your PC to make connections to the Internet. If you have such a program on your PC it can prevent this command type from working properly.

Parameters:

Source URL	Internal parameter. This is the URL to the file to download.
Destination folder	Internal parameter. This is the folder the file should be saved in.
Overwrite existing	Internal parameter. If true an existing file with the same name in the destination folder will be overwritten.

Return value:

0	If the file was downloaded.
-1	If overwrite existing is false and the file already exists.
-2	If an exception was generated trying to start the program.

Settings dependencies:

None.

How to use in Arduino sketch:

`.println("#S|NAME|[]#");` Where NAME is the name of the command as entered in the command editor.

Sample Aduino sketch:

Downloadtest.pde

Read file line command type (RFLIN).

With commands of this type you can ask Gobetwino to read a line from a text file and return it to Arduino.

NOTE:

This command type returns the negative integers -1,-2 and -3 for various error conditions. If you have these numbers in your file you would not be able to distinguish the returned value from an error code.

Parameters:

File	Internal parameter. The full path to the file to read from.
Line nr.	External parameter. The number of the line in the file you want to read.

Return value:

string	The requested line of text from the file
-1	If the file does not exist.
-2	If the requested line number is larger than the number of lines in the file.
-3	If an exception was generated trying to read from the file.

Settings dependencies:

None.

How to use in Arduino sketch:

```
Serial.println("#S|NAME|[LINENR]#");
```

Where NAME is the name of the command as entered in the command editor, and LINENR is a positive integer indicating the number of the line in the file you want to read.

Sample Aduino sketch:

Downloadtest.pde

Log to file command type (LGFIL).

This command type allows you to log data from Arduino to a file on the PC. If for instance your Arduino board was set up with some sensors and you want to log the sensor readings on a regular interval, you could use LGFIL to do the logging for you. Each time the command is called with some data the data is written to a new line at the end of the log file. It is possible to have an optional timestamp logged with each piece of data as well.

NOTE:

By concatenating several values and separating them with semicolons and sending them as one piece of data it is possible to create CSV files that can be imported into spreadsheet programs or databases. See the logtest.pde Arduino sample sketch.

Parameters:

File	Internal parameter. The full path to the file to log data to.
Data	External parameter. The data you want to log to the file.
Time stamp	Internal parameter. If true, a timestamp is added to each log entry.

Return value:

0	The data was logged to the file.
-1	If the log file does not exist.
-2	If an exception was generated trying to read from the file.

Settings dependencies:

None.

How to use in Arduino sketch:

```
Serial.println("#S|NAME|[DATA]#");
```

Where NAME is the name of the command as entered in the command editor, and DATA is the data you want to log to the log file.

Sample Arduino sketch:

Logtest.pde

Copy file command type (CPFIL).

Use this command type if you want to copy a file on the PC. This command is useful if for instance you are logging data to a log file and e-mailing it on a regular interval. You could have an empty log file that you copy over the “used” log file that has just been e-mailed. In this way you would get a fresh empty log file, and can start the logging cycle over again.

Parameters:

Source file	Internal parameter. The full path to the file to copy. The file must already exist, use the browse button to browse to the file
Destination file	Internal parameter. The full path and filename for the copy of the source file. Make sure you type this correct, otherwise the command will fail.
Overwrite existing	Internal parameter. If true, an existing file with the same name as the destination file will be overwritten.

Return value:

0	If the file was copied.
-1	If the destination file exists, and overwrite existing is false.
-2	If an exception was generated trying to copy the file. This could happen if the path to the destination entered is not valid.

Settings dependencies:

None.

How to use in Arduino sketch:

```
Serial.println("#S|NAME|[#");
```

Where NAME is the name of the command as entered in the command editor.

Sample Aduino sketch:

Logtest.pde

Send mail command type (SMAIL).

Use this command type when you want Gobetwino to send an e-mail on behalf of Arduino. You can optionally attach a single file to the e-mail.

NOTE:

To use the SMAIL command you must have the SMTP settings configured with a valid username (e-mail address), password, and SMTP server and port. The username you configure will always be used as the sender of any mails sent from Gobetwino.

Parameters:

To	Internal parameter. The recipient's e-mail address
Subject	Internal parameter. The subject of the e-mail
Body	Internal parameter. The body text of the e-mail
Attachment	Internal parameter. The full path to a file to attach to the e-mail. Can be left empty if no file should be attached.

Return value:

0	If the e-mail was sent.
-1	If the SMTP server is not configured in the settings.
-2	If the attachment file does not exist.
-3	If an exception was generated trying to copy the file. This could happen if the path to the destination entered is not valid.

Settings dependencies:

The four SMTP settings must be filled in with a valid SMTP server name, a user account on that server, a password for the account and a port (usually 25) for the SMTP server.

How to use in Arduino sketch:

```
Serial.println("#S|NAME|[#");
```

Where NAME is the name of the command as entered in the command editor.

Sample Arduino sketch:

gobetwinoXLtest.pde

Ping command type (PING).

Use this command type to ping an IP address or a host name to check if it is present on the network, or Internet.

Parameters:

Address	Internal parameter. The IP address or hostname to ping
----------------	--

Return value:

0	If the remote computer answers the ping.
-1	If the remote computer does not answer the ping.

Setting dependencies:

None.

How to use in Arduino sketch:

`Serial.println("#S|NAME|[#");` Where NAME is the name of the command as entered in the command editor

Arduino sample sketch:

downloadtest.pde

Time command type (T).

When executed the Time command will return the time from the PC's clock to Arduino. The format of the returned string is determined by the "Long time format" in the control panels Regional and language settings applet. On my computer running a Danish Windows XP the format is HH:MM:SS.

This is one of the special command types that have no parameters. You do not need to create a command before using it. A predefined command with the name T is ready for you to use:

Parameters:

None.

Return value:

The time	A string with the time.
-----------------	-------------------------

Setting dependencies:

None.

How to use in Arduino sketch:

```
Serial.println("#S!T![]#");
```

Arduino sample sketch:

No sample sketch for this command type.

Date command type (D).

When executed the Date command will return the date from PC's clock to Arduino. The format of the returned string is determined by the "Short date format" in the control panels Regional and language settings applet. On my computer running a Danish Windows XP the format is DD:MM:YYYY.

This is one of the special command types that have no parameters. You do not need to create a command before using it. A predefined command with the name D is ready for you to use:

Parameters:

None.

Return value:

The date	A string with the date.
-----------------	-------------------------

Setting dependencies:

None.

How to use in Arduino sketch:

```
Serial.println("#SID[[]#");
```

Arduino sample sketch:

No sample sketch for this command type.

Receiving e-mails in Gobetwino.

Gobetwino can be configured to check a POP3 mailbox for incoming mails on a regular interval. If one or more messages have arrived in the mailbox, the messages are retrieved, and read. The text in the messages body text is send to Arduino.

This way it is possible to send commands to Arduino via e-mail.

This feature only works with mailboxes you can access with the POP3 protocol. Not all web mail systems allow you to do that.

VERY IMPORTANT NOTE: the incoming mail feature in Gobetwino is intended to be used with a dedicated mail account for this purpose. When checking for incoming mails Gobetwino will first check for the number of messages in the mail box, then it will retrieve each message, check that the subject and sender is ok, extract the body text and send the entire body text to Arduino **AND THEN IT WILL AUTOMATICALLY DELETE ALL MAILS IT HAS READ FROM THE MAILBOX.**

This essentially means that after each check the mailbox will be empty. The reason for this drastic behaviour is that the POP3 protocol is rather primitive it has no way of telling you the number of unread messages, or if any specific message has already been read. So to keep track of things without having to implement a full mail client Gobetwino read all messages, send information to Arduino from those messages with the correct subject and sender, and then deletes the messages.

It is important to understand that the commands you can send to Arduino have nothing to do with the predefined command types in Gobetwino. What you send as the body text in the mails, and how Arduino interprets it is entirely up to you. But do remember that Arduino has very limited memory, so use short strings.

It is also important to understand that setting up Gobetwino to automatically check a mail box and send these commands to Arduino does not go very well along with using the regular Gobetwino commands. This is because if you mix the two, you can easily end up in a situation where Arduino has asked Gobetwino to execute a command and is waiting for the answer, at the same time the mail check kicks in and read a mail and send the command from the e-mails body text to Arduino. Arduino has no way to tell that this is a command from an e-mail, and not the return value from the command it asked Gobetwino to execute.

The next version of Gobetwino will implement a command type that allows Gobetwino to ask for a mail check. This will eliminate this problem. Until then, use the two sets of functionality independently.

To set up Gobetwino to automatically check for mails and send the commands to Arduino, you must configure all the POP3 settings in the settings dialog. See the chapter on settings.

In the settings you also define an “accept subject” setting. Gobetwino will only send command from e-mails with this setting to Arduino. This is to avoid problems with spam interfering with your e-mailed commands.

There is also an optional “Accept mail from” setting that allows you to define a sender e-mail address. If this setting is filled in, Gobetwino will only send commands to Arduino from e-mails coming from that sender. If the setting is left blank, e-mails from any sender will be accepted if they have the right subject.

NOTE: Gobetwino will try to figure out if a received mail is in plain text format or HTML format. If it is in HTML format it will try to strip away all HTML before sending the body text to Arduino. On a few occasions I have experienced that this stripping of HTML has not been 100 % successful, so whenever possible use plain text for sending mails to Gobetwino.

The mailtest.pde sample sketch demonstrates how you can send an e-mail with the one word “blink” in the body text, and makes a LED blink when Gobetwino reads the mail and send the command to Arduino.

Settings.

There are a number of settings and configuration parameters you can use to set up Gobetwino to work as you need.

You can access these settings from the Settings dialog. To open the settings dialog click the Settings menu.

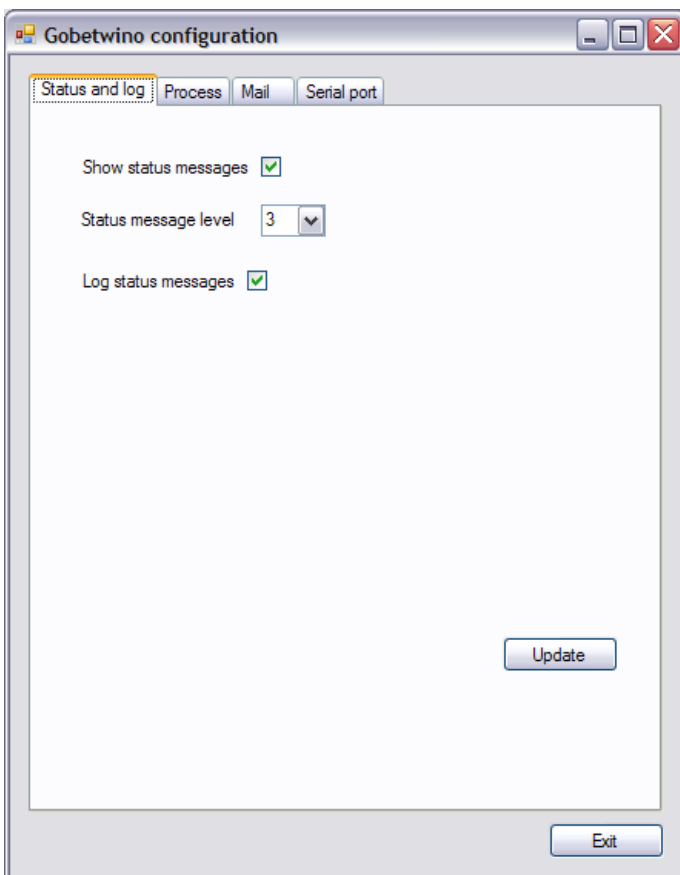
The settings dialog has 4 tabs, Status and log, Process, Mail and serial port. Each tab has an update button that updates any changes made to the settings in that tab.

NOTE:

Changing the settings does not take effect until Gobetwino is restarted.

The status and log settings tab.

These settings determine how Gobetwino show and log status messages.



Show status messages: determines if status messages are shown at all.

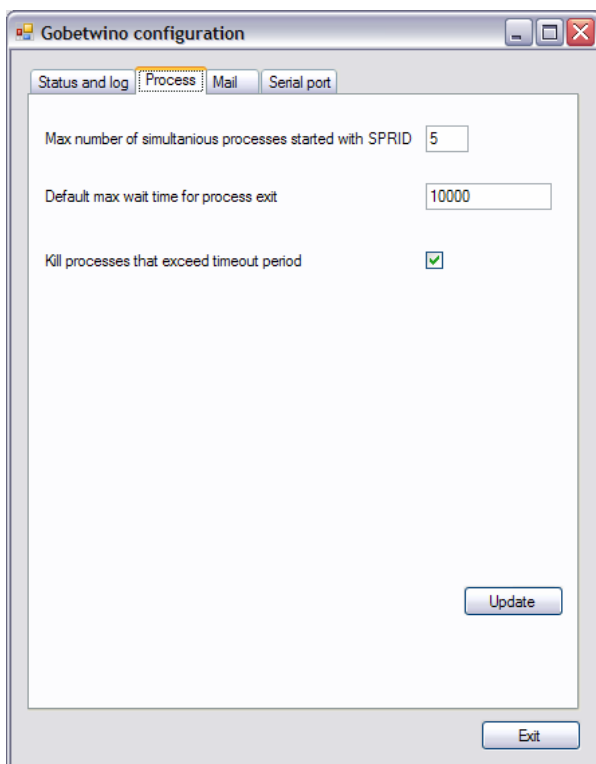
Status message level (1 – 3) this setting determines which status messages are shown. If 1 is selected only error conditions are shown, if 3 is selected everything is shown.

EDITORS NOTE: this feature is not implemented in the current version. Selecting a logging level does not actually influence the status messages shown.

Log status messages: this setting determines if status messages are logged to the log file. Logging status messages to the log file can be helpful when running Gobetwino for long periods of time, where you can't watch the status message area all the time. Status messages are logged to the gobetwino.log file in the folder where Gobetwino is installed. Just be aware that the log file can quickly grow to a very large size.

The process settings tab.

These settings control the behaviour of programs (processes) started with the SPRID and SPWEX command types.



The number of simultaneous processes started with SPRID: this setting is the limit for how many programs can be run at the same time, started with the SPRID command type. The reason for this limit is that it is very easy to accidentally have a loop in an Arduino sketch that gets a little out of control and runs for many more iterations than intended. I had a small accident of this nature when developing Gobetwino that resulted in around 100 instances of notepad.exe using up just about every available resource on my PC. 5 are probably more than enough.

Default max wait time for process exit: this setting determines how long Gobetwino waits for a program started with the SPWEX command type to exit, if the timeout is set to 0 in the command editor. This is to make sure that Gobetwino will not get stuck waiting for ever for a program that does not finish.

Kill processes that exceed timeout period: if checked, this setting will force Gobetwino to end processes started with the SPWEX command type, that do not end within the timeout period.

The mail settings tab.

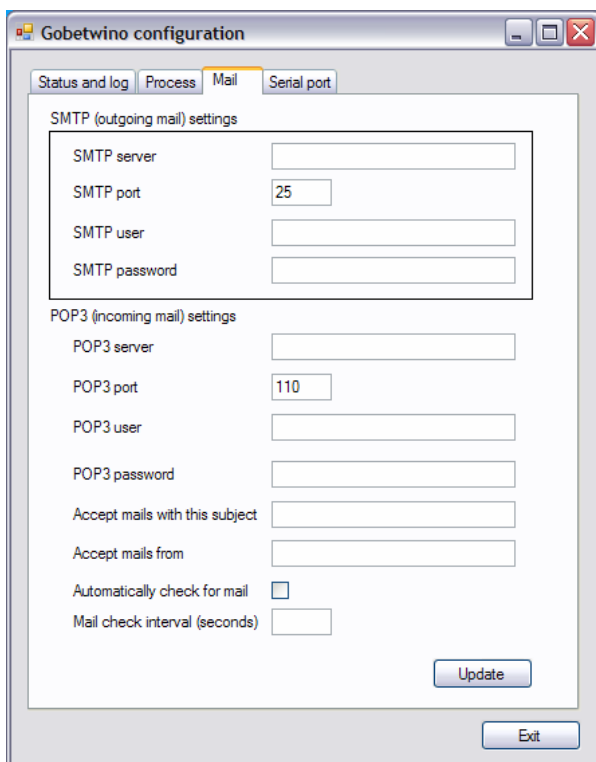
Pleas take some time to read the chapter on using incoming mails in Gobetwino, there are some VERY IMPORTANT information in that chapter.

GOBETWINO WILL DELETE MESSAGES IT READ FROM THE MAILBOX.

These settings control the sending and reception of e-mails in Gobetwino.

To send e-mails the SMTP settings must be filled in with valid information about an SMTP account that will relay e-mails for you.

To receive e-mails the POP3 settings must be filled in with valid information about a POP3 mailbox, you have access to.



The screenshot shows the 'Gobetwino configuration' window with the 'Mail' tab selected. The window is divided into two main sections: 'SMTP (outgoing mail) settings' and 'POP3 (incoming mail) settings'. The SMTP section includes fields for 'SMTP server', 'SMTP port' (set to 25), 'SMTP user', and 'SMTP password'. The POP3 section includes fields for 'POP3 server', 'POP3 port' (set to 110), 'POP3 user', 'POP3 password', 'Accept mails with this subject', and 'Accept mails from'. There is also a checkbox for 'Automatically check for mail' and a 'Mail check interval (seconds)' field. At the bottom right of the configuration area, there are 'Update' and 'Exit' buttons.

Most of these settings should be obvious.

The standard port used for SMTP is 25, but some Internet service providers may use a different one.

The same goes for the POP3 port which I usually 110.

To avoid problems with spam you have to fill in a subject that Gobetwino will check before sending commands from the e-mail to Arduino. Only commands from e-mails with this subject will be accepted.

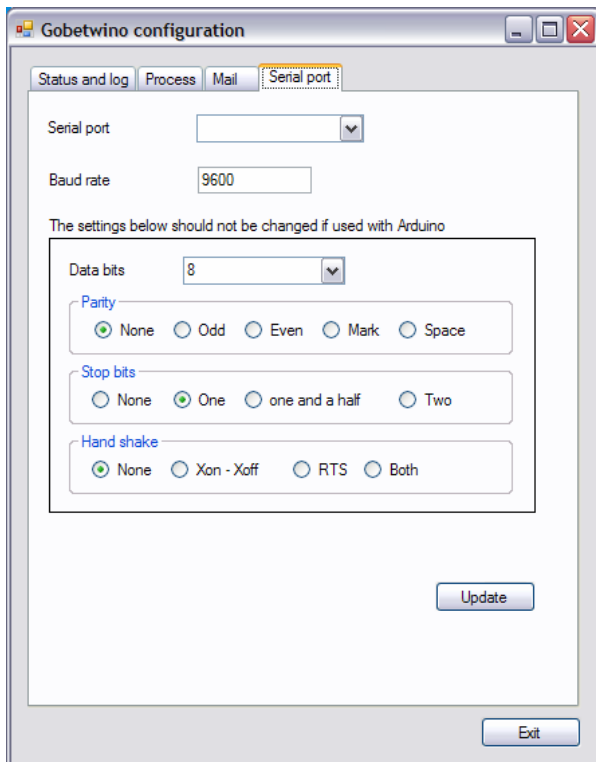
You can optionally fill in the Accept mails from field with an e-mail address. If it is filled in, only commands in mails from that address will be sent to Arduino, and only if the subject is also correct.

The serial port settings tab.

Use the settings in this tab to configure the PC's serial port.

If you use Gobetwino with Arduino you should only change the serial port and baud rate settings. The other settings should be left as they are.

If you use Gobetwino with anything else than Arduino change all the settings to match the serial port of the device you want Gobetwino to communicate with.



Arduinos serial port is a virtual serial port based on the FTDI chip on the Arduino board, and the drivers for that chip installed on the PC. This port only exists in Windows when the Arduino board is connected to one of the PC's USB ports. To avoid problems make sure your Arduino board is connected to your PC before starting Gobetwino, otherwise Gobetwino can not open the port.

You need to know the port name of this virtual serial port, this name is the port you select in the Serial port drop down box.

The baud rate setting should always match the baud rate setting of the device you are communicating with. In the case of Arduino, the setting should be the same as the value you use in the `Serial.begin(XXXX)` statement in your Arduino sketch.

Appendix A syntax for SENDK command.

The SENDK command type is based on the Windows API function SendKeys.

This is the part of the Microsoft documentation for SendKeys that describes how to format the keystrokes to send.

Most keyboard characters are represented by a single keystroke. Some keyboard characters are made up of combinations of keystrokes (CTRL+SHIFT+HOME, for example). To send a single keyboard character, send the character itself as the *DATA* parameter. For example, to send the letter x, send the *DATA* parameter "x".

Note:

To send a space, send the string " ".

You can use **SendKeys** to send more than one keystroke at a time. To do this, create a compound string that represents a sequence of keystrokes by appending each keystroke in the sequence to the one before it. For example, to send the keystrokes a, b, and c, you would send the string argument "abc". The **SendKeys** method uses some characters as modifiers of characters (instead of using their face-values). This set of special characters consists of parentheses, brackets, braces, and the:

- plus sign "+"
- caret "^"
- percent sign "%"
- and tilde "~"

Send these characters by enclosing them within braces "{}". For example, to send the plus sign, send the string argument "{+}". Brackets "[]" have no special meaning when used with **SendKeys**, but you must enclose them within braces to accommodate applications that do give them a special meaning (for dynamic data exchange (DDE) for example).

- To send bracket characters, send the string argument "{[" for the left bracket and "]" for the right one.
- To send brace characters, send the string argument "{{" for the left brace and "}" for the right one.

Some keystrokes do not generate characters (such as ENTER and TAB). Some keystrokes represent actions (such as BACKSPACE and BREAK). To send these kinds of keystrokes, send the arguments shown in the following table:

Key	Argument
BACKSPACE	{BACKSPACE}, {BS}, or {BKSP}
BREAK	{BREAK}
CAPS LOCK	{CAPSLOCK}
DEL or DELETE	{DELETE} or {DEL}
DOWN ARROW	{DOWN}
END	{END}
ENTER	{ENTER} or ~
ESC	{ESC}
HELP	{HELP}

HOME	{HOME}
INS or INSERT	{INSERT} or {INS}
LEFT ARROW	{LEFT}
NUM LOCK	{NUMLOCK}
PAGE DOWN	{PGDN}
PAGE UP	{PGUP}
PRINT SCREEN	{PRTSC}
RIGHT ARROW	{RIGHT}
SCROLL LOCK	{SCROLLLOCK}
TAB	{TAB}
UP ARROW	{UP}
F1	{F1}
F2	{F2}
F3	{F3}
F4	{F4}
F5	{F5}
F6	{F6}
F7	{F7}
F8	{F8}
F9	{F9}
F10	{F10}
F11	{F11}
F12	{F12}
F13	{F13}
F14	{F14}
F15	{F15}
F16	{F16}

To send keyboard characters that are comprised of a regular keystroke in combination with a SHIFT, CTRL, or ALT, create a compound string argument that represents the keystroke combination. You do this by preceding the regular keystroke with one or more of the following special characters:

Key	Special Character
SHIFT	+
CTRL	^
ALT	%

 **Note:**

When used this way, these special characters are not enclosed within a set of braces.

To specify that a combination of SHIFT, CTRL, and ALT should be held down while several other keys are pressed, create a compound string argument with the modified keystrokes enclosed in parentheses. For example, to send the keystroke combination that specifies that the SHIFT key is held down while:

- e and c are pressed, send the string argument "+(ec)".
- e is pressed, followed by a lone c (with no SHIFT), send the string argument "+ec".

You can use the **SendKeys** method to send a pattern of keystrokes that consists of a single keystroke pressed several times in a row. To do this, create a compound string argument that specifies the keystroke you want to repeat, followed by the number of times you want it repeated. You do this using a compound string argument of the form `{keystroke number}`. For example, to send the letter "x" ten times, you would send the string argument `"{x 10}"`. Be sure to include a space between keystroke and number.

 **Note:**

The only keystroke pattern you can send is the kind that is comprised of a single keystroke pressed several times. For example, you can send "x" ten times, but you cannot do the same for "Ctrl+x".

 **Note:**

You cannot send the PRINT SCREEN key `{PRTSC}` to an application.