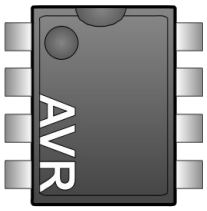


AVRPascal

Manual

version 07/07/2026

1. Introduction

AVRPascal is an IDE designed for programming AVR microcontrollers (ATtiny and ATmega families) in the Pascal language. It leverages the Free Pascal[1] compiler (FPC) to compile source code and AVRdude[2] to transfer the compiled code to the microcontroller's memory using programmers like USBasp[3] or Arduino[4]. Key features:

- Syntax highlighting for improved code readability.
- Error indication: Points to the line of code containing compilation errors.
- Tooltips: Provide information about function and procedure parameters, record fields, and class methods.
- Tabbed document interface for efficient code organization.
- Microcontroller fuse-bit configuration capability.
- Automatic detection of supported USB devices.
- Serial port monitor for Arduino boards.

AVRPascal is a 64-bit application. It is available in three versions for the following operating systems: Windows, Linux (Debian-based distributions), and MacOS. The functionality of these versions is almost identical, with minor differences arising from the specific requirements of each operating system.

2. Installation

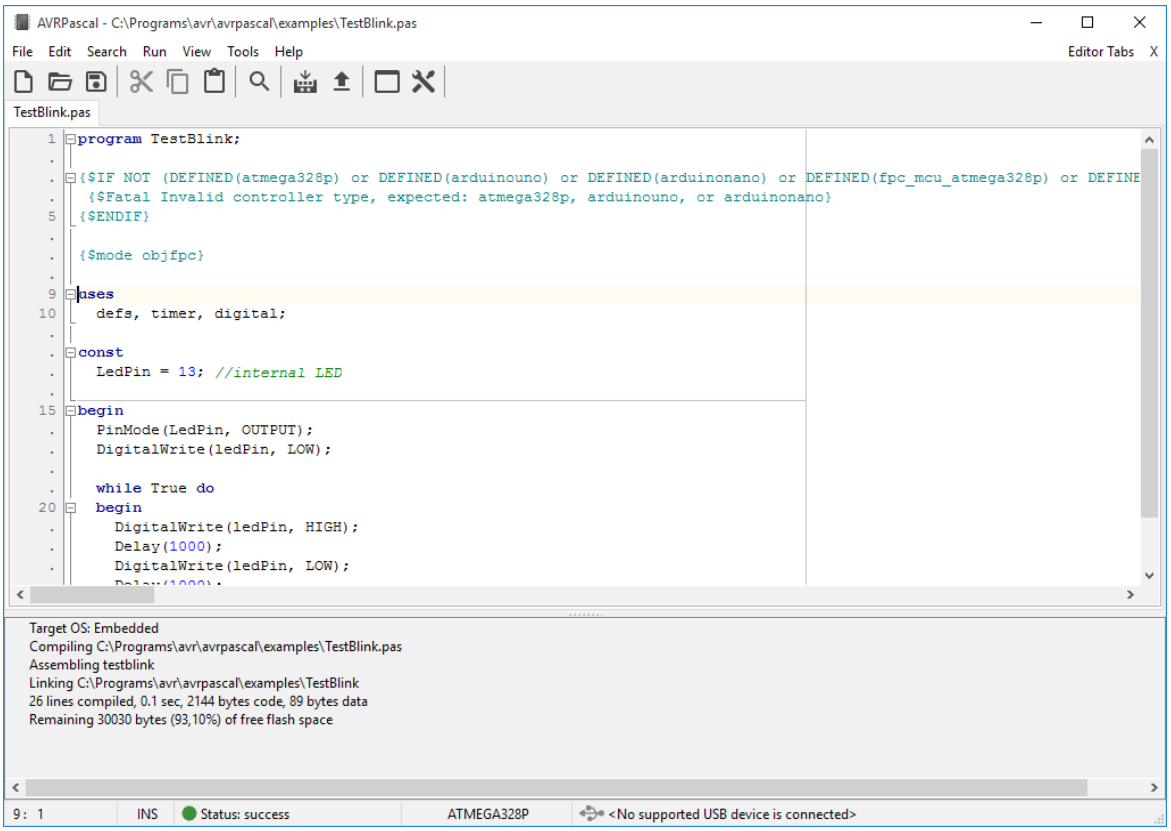
The application should be installed in the target operating system using the appropriate installer. On Windows, use the *.exe* file. Additional drivers may be required for the USBasp programmer (*libusb*[5] library driver), or Arduino[6] board. On Linux (Debian-based distributions) use the *.deb* package. Additional drivers are usually not required as they are included in the operating system. For USBasp or Arduino, ensure AVRPascal has administrator privileges or add the current user to the *dialout* group. On MacOS use the *.pkg* file. No additional procedures should be necessary.

---

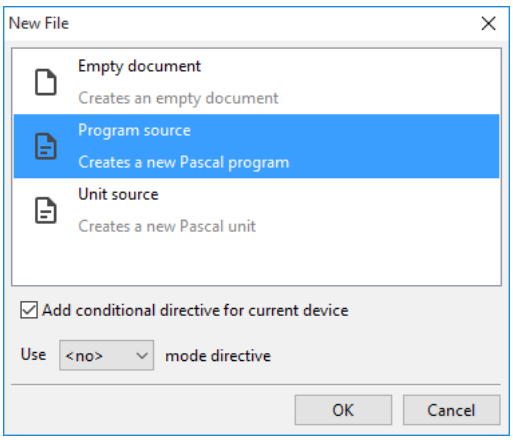
1 https://www.freepascal.org/
2 https://savannah.nongnu.org/projects/avrdude/
3 https://www.fischl.de/usbasp/
4 https://store.arduino.cc/products/. For a list of supported Arduino boards, see chapter 5.
5 http://zadig.akeo.ie/
6 https://docs.arduino.cc/tutorials/generic/DriverInstallation/

3. Editor

The application window, similar to other code editing and compiling applications, is divided into several areas: menu bar, a toolbar, a tab area, a messages area, and a status bar. The application supports editing multiple files in tabs, with at least one editor tab always open. The message list includes notifications from the compiler, uploader and debugger (if installed).



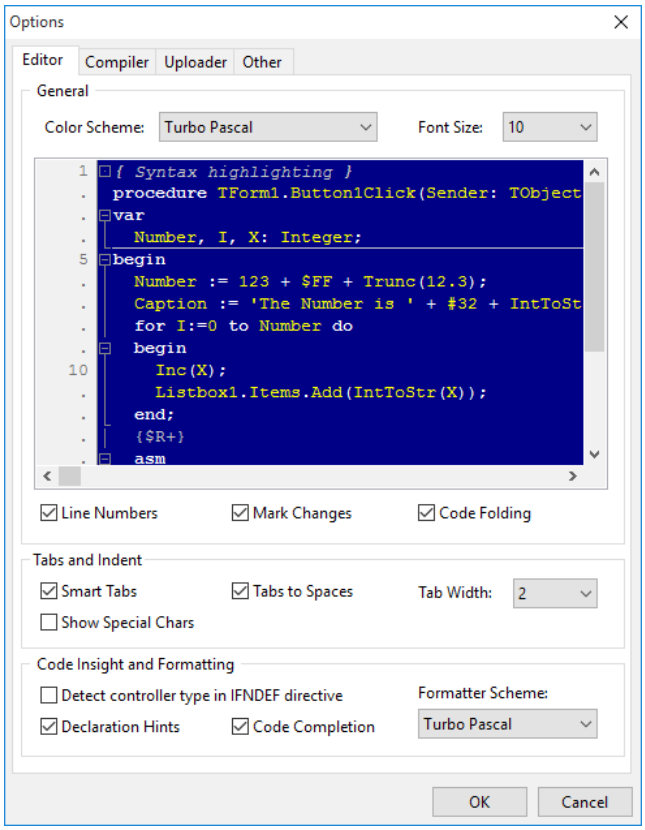
The application's menu offers standard editor functions for opening, editing, and saving Pascal source files (.pas, .pp, .inc). Text search is also available within the currently edited file. When creating a new file, simple templates can be used to generate program or unit skeletons.



Additionally, when selecting a program or module, two additional options are available. Selecting *Add conditional directive for current device* adds a directive to the template. This directive will cause a compilation error if the code is compiled for a device other than the one currently selected (see Chapter 4). The second option, *Use mode directive*, enables switching between compiler language dialect modes. If objpas is selected, the compiler mode is set to Object Pascal compatibility.

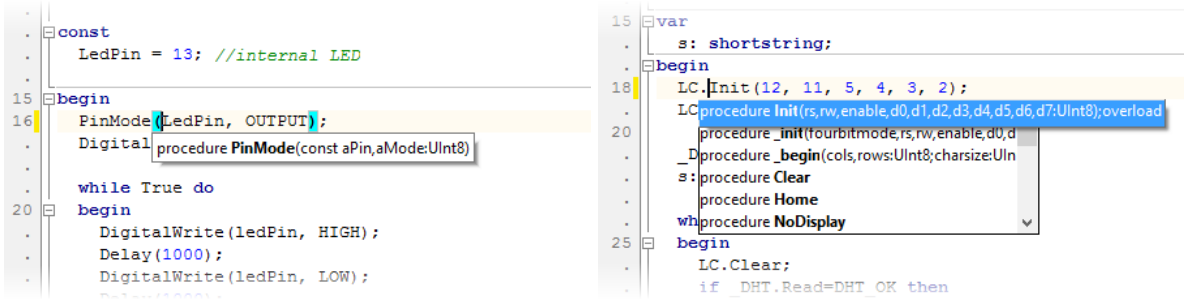
Additionally, standard text editing functions like selection, cutting, copying, pasting, deleting, and undo/redo are available.

The main feature of AVRPascal that makes working with source code easier is Pascal syntax highlighting. Currently, the application supports three schemes: Delphi, Turbo Pascal (referring to the colors used in these applications) and Twilight (high-contrast colors). The default color scheme is the *Delphi* scheme. It can be changed by selecting the *Options* menu function and going to the *Editor* tab. Font size and editor gutter features can also be configured here, including line number visibility, highlighting of modified lines, and code folding. Additionally, TAB key and indentation behavior can be configured in the *Tabs and Indent* frame, and source code can be exported to an HTML file with syntax highlighting. (*File->Export to HTML*)[7].



7 The MacOS version does not include the ability to export Pascal source files to HTML format.

AVRPascal provides two additional source code assistance features: declaration hints and code completion (lists of record fields and class methods). These features, common in code editors, provide suggestions while typing. Parameter hints appear after typing an opening parenthesis, while a list of record fields and class methods pops up after a dot.



A similar feature is the ability to detect the microcontroller type based on the $IFNDEF or $IF NOT DEFINED directives (see Chapter 4). If this option is enabled, when opening and saving a file and finding such a directive in the source code, a dialog window will appear asking whether to change the current microcontroller type to the one required by this directive.

AVRPascal also has a built-in source code formatter supporting two formatting schemes (styles): *Delphi* and *Turbo Pascal*, defined in the *Options* window, *Editor* tab. The formatter can be invoked by the menu *Edit->Format Source* and formats the text of the active program tab[8].

For increased efficiency, AVRPascal provides a set of keyboard shortcuts for working with source code:

Shortcut	**Function**
Ctrl+N	Opens a new editor tab (*File->New*).
Ctrl+O	Opens the source file (*File->Open*).
Ctrl+S	Saves the source file (*File->Save*).
Ctrl+Z	Undoes the last change in the code (*Edit->Undo*).
Ctrl+X	Cuts the selected text and places it in the clipboard (*Edit->Cut*).
Ctrl+C	Copies selected text (*Edit->Copy*).
Ctrl+V	Pastes the selected text (*Edit->Paste*).
Ctrl+A	Selects the entire text of the document (*Edit->Select All*).
Del	Deletes the selected text or character after the cursor.
Ins	Switches between insert and overwrite characters.
Ctrl+F	Displays a search dialog for the entered characters in the text

8 The formatter functionality is based on a modified code of the DelForEx library written by Egbert van Nes (http://www.aew.wur.nl/UK/Delforexp).

	(*Search->Find*).
F3	Finds the next occurrence of the entered characters in the text (*Search->Find Next*).
Ctrl+/	Comments on the selected text using "//" characters or removes the comment.
Ctrl+U, Tab	Increases the indentation of selected text by two spaces.
Ctrl+I, Shift+Tab	Decreases the indentation of selected text by two spaces.
Ctrl+Up arrow	Moves the cursor to the procedure/function/method declaration.
Ctrl+Down arrow	Moves the cursor to the first line of procedure/function/method code.
Ctrl+[text under the mouse cursor]	When clicked, it searches the current document and other documents defined in the uses section for text declarations "under" the mouse cursor. Equivalent to the *Find declaration* context menu function.
Ctrl+ [+ on numeric keypad]	Increases the text font size.
Ctrl + [- on numeric keypad]	Decreases the text font size.
Ctrl+D	Formats the code according to the settings in the *Options* window, *Editor* tab, *Code Insight and Formatting* frame.
Ctrl+H	Moves the cursor to the previous point in the jump history.
Ctrl+Shift+H	Moves the cursor to the next point in the jump history.

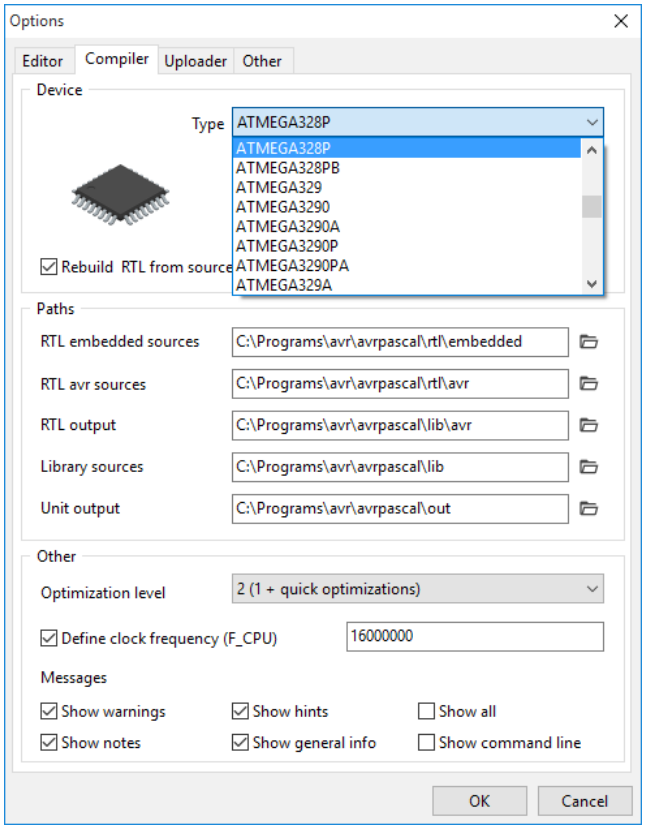
4. Compilation

AVRPascal uses the Free Pascal compiler version 3.3.1. To compile the code, Free Pascal translates Pascal code into assembly code and then utilizes the GNU toolchains for AVR microcontrollers, which includes the avr-as assembler and the avr-ld linker. This process generates three output files: *.bin (binary), *.elf (linker format), and *.hex (text format). Due to Free Pascal's capabilities, the list of supported AVR microcontrollers is quite wide, encompassing the ATtiny and ATmega families (see Appendix 1).

Compiler settings can be found in the *Compiler* tab of the *Options* window (accessible through the *View->Options* menu). These settings are global, meaning any source file compiled in AVRPascal will use the configurations defined here.

In the *Device* frame, choose the target microcontroller type or Arduino board. This ensures specific definitions are included during compilation for the selected microcontroller[9]. The *Rebuild RTL from source if needed* option determines whether AVRPascal automatically rebuilds Free Pascal runtime library (RTL) files after changing the microcontroller type. If unchecked, AVRPascal displays a confirmation dialog. Additionally, the *Run->Rebuild RTL Sources* menu provides a command for rebuilding the RTL.[10].

9 The current microcontroller type is also displayed on the status bar.



The *Paths* frame manages the locations for files used by Free Pascal during compilation. Paths defined in *RTL avr sources*[11] and *RTL embedded sources*[12] determine the source files for the Free Pascal runtime library (RTL), which provide essential functionalities used by compiled programs. *RTL Output* and *Library Sources* paths specify the locations where the compiled versions of the RTL files or other library files[13] are saved. *Unit Output* path determines where the compiled code for program files (units) will be placed. The paths are automatically set during AVRPascal installation and changing them is not recommended for beginners as it can affect compilation processes.

The *Other* frame includes additional compiler settings like optimization level[14], target microcontroller's clock frequency, and the verbosity level of compiler messages displayed in the AVRPascal messages area.

As mentioned earlier, AVRPascal utilizes global compiler settings and doesn't rely on project files or additional definition files. Consequently, it uses the traditional Pascal

10 In practice, AVRPascal rebuilds three files: system.pp, objpas.pp and the appropriate microcontroller configuration file, e.g. atmega328p.pp. It uses the -O2 optimization level and the -dRELEASE mode.
11 The *RTL avr sources* path is used to store definition files for individual types of AVR microcontrollers.
12 The *RTL embedded sources* path is used to store Free Pascal "system" files from which the system.o file is built.
13 The *Library sources* path is used to point to the sources of AVRPascal's UnoLib library.
14 Using high build optimization levels is not recommended, as it may increase the chance of encountering compilation errors or issues with correctly compiling the source files. Moreover, when compiling source files, support for labels (-Sg) and procedure/function inlining (-Si) are enabled by default.

source file types: defined by *program* keyword, for creating executable files, and defined by *unit* keyword, for creating module files. Moreover during compilation AVRPascal defines a symbol corresponding to the global microcontroller type (e.g. *atmega328p*, see Appendix 1) and a symbol indicating the family it belongs to (*familytiny* or *familymega*)[15]. These symbols enable features like conditional compilation (include or exclude code segments based on the chosen microcontroller) or defining compilation errors (generate specific error messages based on the microcontroller type), e.g.:

```
{$IFNDEF atmega328p}
 {$Fatal Invalid controller type, expected: atmega328p}
{$ENDIF}
```

In this example, if the global microcontroller type is other than *atmega328p*, compilation of the source file will be interrupted and an error message *Invalid controller type, expected: atmega328p* will be displayed in the messages area.

Moreover, during compilation, special symbols are defined regarding the presence and version of AVRPascal: *AVRPASCAL*, *AVRPASCAL_VERSION*, *AVRPASCAL-_RELEASE*, *AVRPASCAL_PATH*, *AVRPASCAL_BUILD*, *AVRPASCAL_FULL-VERSION* and UnoLib: *UNOLIB*, *UNOLIB_VERSION*, *UNOLIB_RELEASE*, *UNOLIB_FULLVER-SION*[16]

Compilation and execution commands for the currently active editor tab are located in the *Run* menu. They allow compilation (*Compile*), build (*Build All*), deletion of compilation result files (*Clear All*) and rebuilding RTL files depending on selected microcontroller (*Rebuild RTL Sources*). The *Upload* function transfers the compiled program to the target microcontroller's flash memory using the uploader. This enables running the program directly on the device. AVRPascal also offers keyboard shortcuts to streamline compilation and execution processes:

Shortcut	**Function**
Ctrl+F9	Compiles the source file from the active tab (*Run->Compile*)
Shift+F9	Builds the source file from the active tab (*Run->Build All*)
F9	Exports the compiled source file to the microcontroller's flash memory via the uploader (*Run->Upload*)

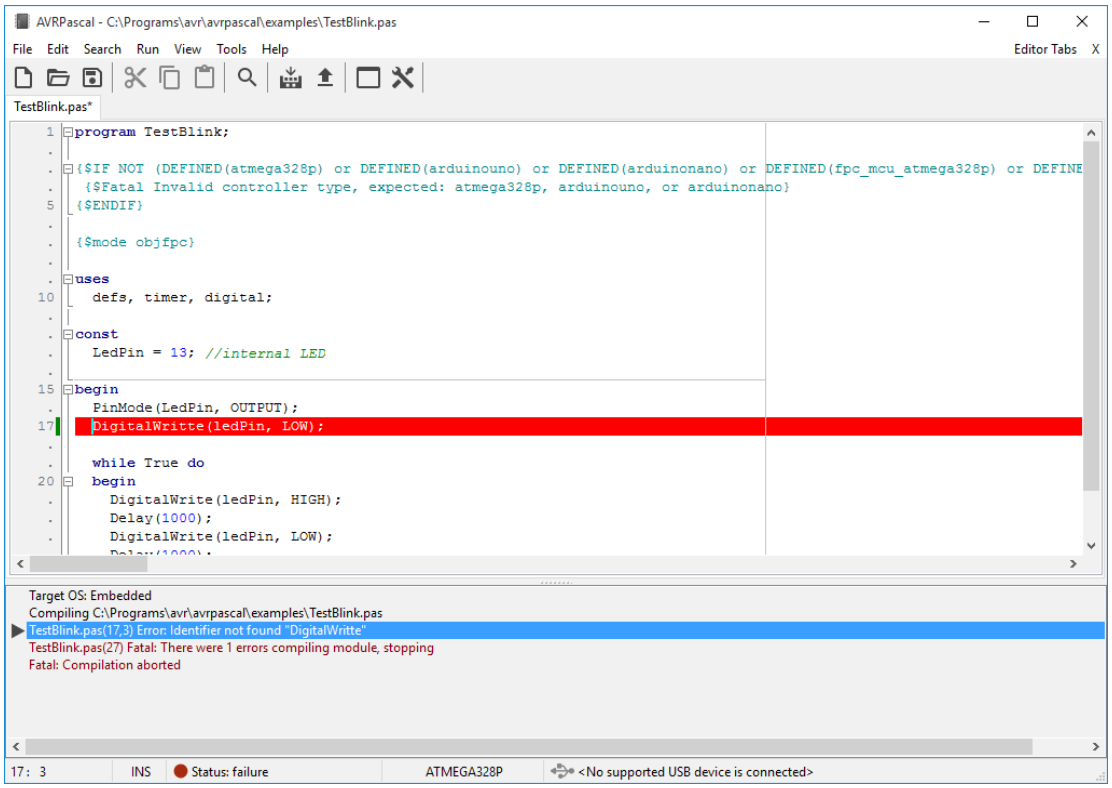
During compilation, information returned by the compiler is automatically displayed in the *Messages* area. If the compilation was successful, the status bar will display *Status: success*, and additional information about the device flash memory usage will be displayed in the *Messages* area[17]. Otherwise, *Status: failure* will show in the status bar. Additionally, if

15 If the device type selected by the user is an Arduino board, then the corresponding microcontroller type is additionally defined. Regardless of this, FPC version 3.3.1 defines analogous symbols, preceded by the prefix *FPC_MCU*, e.g. *FPC_MCU_ATMEGA328P*, although in the case of Arduino boards it does not define the corresponding microcontroller types.

16 These symbols may be useful for conditional compilation. In the case of UnoLib, they refer to the version of the library distributed with AVRPascal.

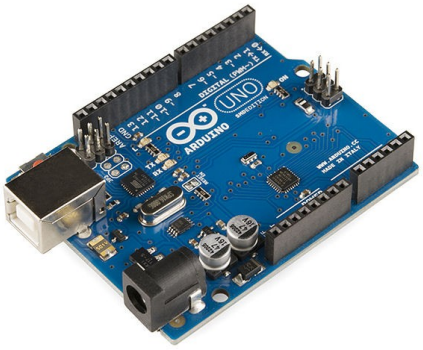
17 The amount of free flash memory remaining after uploading the program to the target device's flash memory.

the compiler indicated an incorrect place in the source code, the line with the error will be highlighted in red.



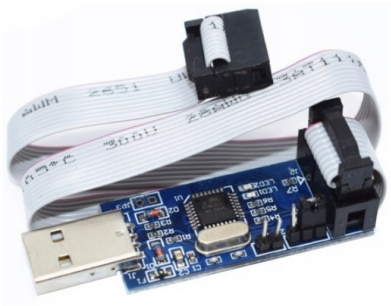
5. Using the programmer and uploader

To run the compiled source code, it must be loaded into the microcontroller's flash memory using a programmer, which is a physical device that communicates with the microcontroller. The programmer is typically connected to the computer via a USB port[18]. An uploader, which is a program that sends binary data to the programmer, is used to transfer the compiled code. AVRPascal uses AVRDude in version 8.0 as an uploader, which supports most AVR microcontrollers and works with both the USBasp programmer and Arduino boards.

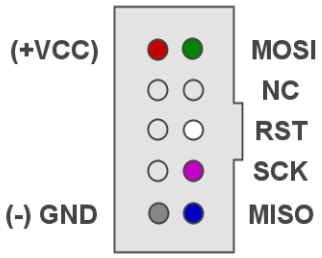


Arduino Uno

18 There are other methods used in older generations of PCs, using e.g. the COM or LPT port, but these solutions are impractical because the standard is currently the USB port and few computers have a COM or LPT connection.



USBasp programmer with tape

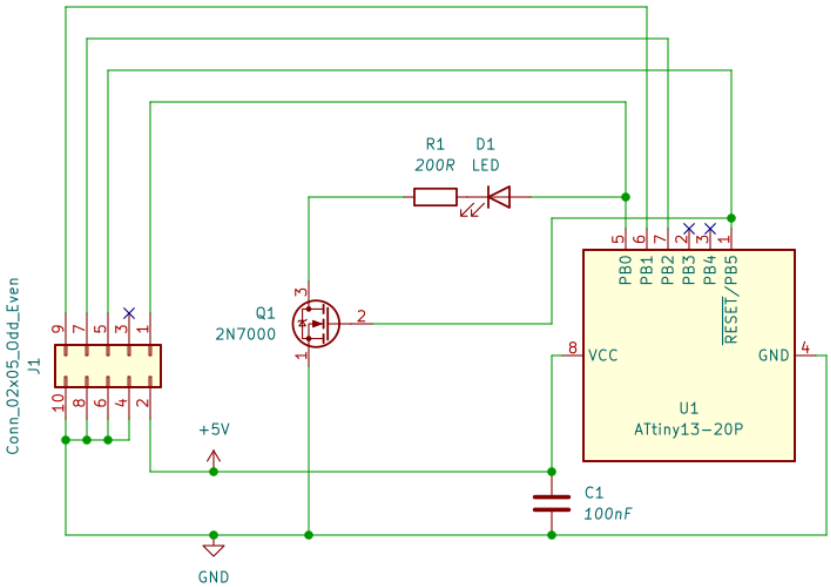


USBasp programmer connector

When using an Arduino board as a programmer, simply connect it to the computer via USB. However, it is important to remember to use sources designed for the microcontroller on which the board is based when programming Arduino. AVRPascal supports the following Arduino boards (the type of microcontroller of the board is given in brackets):

ARDUINO LEONARDO (ATMEGA32U4)	*ARDUINO NANO (ATMEGA328P)*	*ARDUINO NANO EVERY (ATMEGA4809)*
ARDUINO MEGA (ATMEGA2560)	*ARDUINO MICRO (ATMEGA32U4)*	*ARDUINOUNO (ATMEGA328P)*

When using a USBasp programmer, the signal lines (MISO, MOSI, RST, SCK) and power supply connections (VCC, GND) must be connected correctly to the microcontroller. The method of connecting the pins of specific microcontrollers should be checked in their datasheets. For ATtiny13 it may be similar to the following:

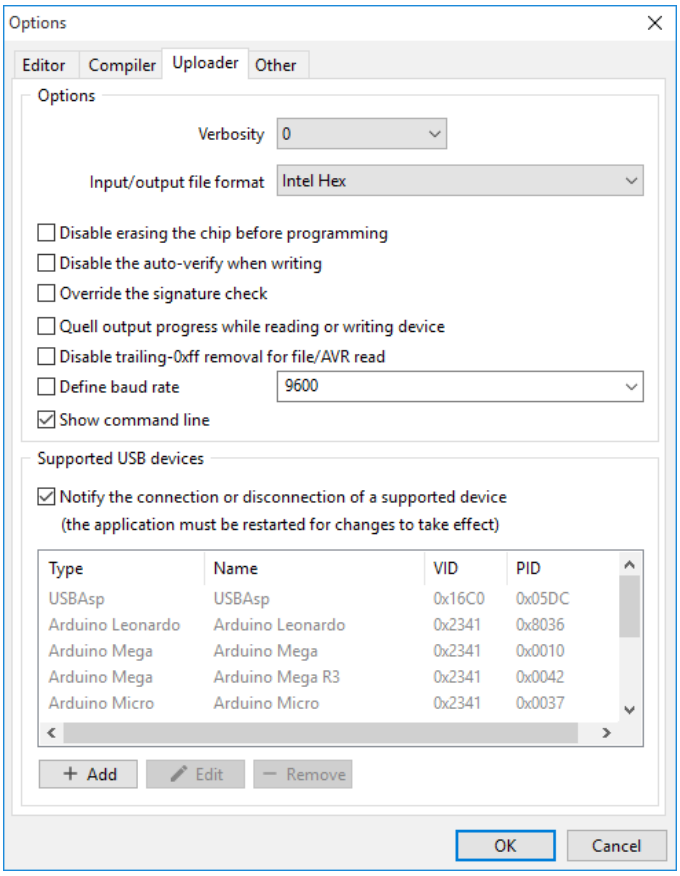


Here, the programmer should be connected to a 10-pin IDC10 (Kanda) connector, pins 4, 6, 8 and 10 of which are connected to ground, pin 2 to the 5V power supply, pin 1 to the PB0 port, pin 5 to PB5, pin 7 to PB2, pin 9 from PB1 of the microcontroller, while pin 3

remained unconnected. The circuit uses a 2N7000 MOSFET transistor that opens if the gate voltage exceeds approximately 2 V. During programming, USBasp shorts pin 1 (PB5) of the microcontroller to ground, closing the transistor and cutting off the line connected to the drain, in this case the LED. The transistor therefore plays the role of a switch that cuts off the ground line of the diode (or other elements connected in this way) during programming, and it does not negatively affect the operation of the programmer. On the other hand, pin 1 (PB5) can be used during normal operation of the microcontroller.

AVRPascal also supports programming using ArduinoISP. This allows an Arduino board to be used as an ISP programmer for microcontrollers equipped with an ISP interface, providing an alternative to the USBasp programmer[19].

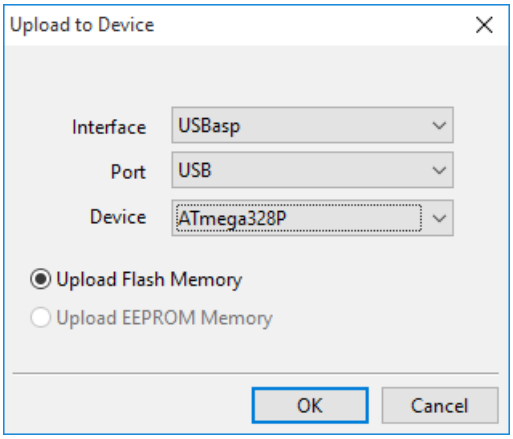
The uploader parameters are located in the *Uploader* tab of the *Options* window (*View->Options* menu). The *Uploader* tab provides settings for the verbosity level of uploader messages displayed in the Messages area (levels 0–4), input and output file formats (*.bin, .elf or .hex), and AVRdude options used during programming and data transfer operations. These include: disabling data clearing mode on the microcontroller before programming, disabling write verification, skipping microcontroller signature checking, visibility of read/write progress from/to the microcontroller, removing trailing 0xFF characters when reading from a file/microcontroller, and setting baud rate[20].

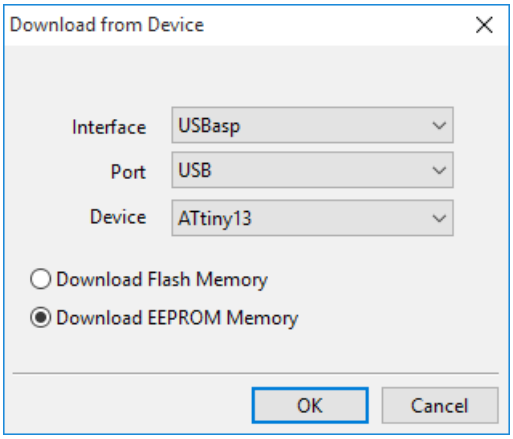


19 To use ArduinoISP, first upload the ArduinoISP.pas example from the *examples* directory to the Arduino board, then connect it to the target microcontroller as described in *docs/ArduinoISP.pdf*.

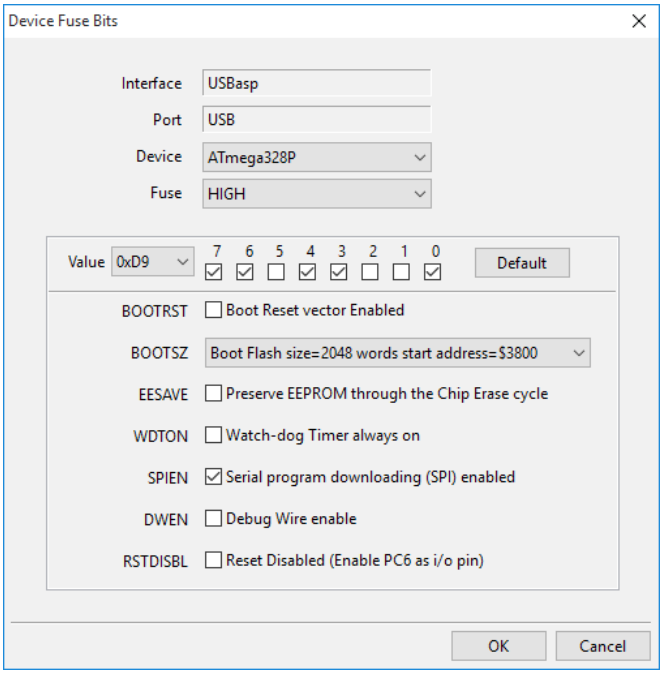
20 In most cases, when working in AVRPascal, changing the uploader parameters is not necessary.

The uploader used in AVRPascal is primarily responsible for transferring the compiled program to the microcontroller's flash memory. This can be done using the *Run->Upload* command (see Chapter 4). After running this command, a dialog box appears. The Interface field may show USBasp, ArduinoISP (if the microcontroller supports the ISP protocol) or Arduino (if one of the listed Arduino boards is set as the target device type for the compiler). In other cases, writing to the microcontroller's flash memory will not be possible. The *Device* field displays a list of subtypes of the selected microcontroller that are supported by the uploader. Similar to working with the compiler, uploader messages are displayed in the messages area and the result of the upload process appears in the status bar.

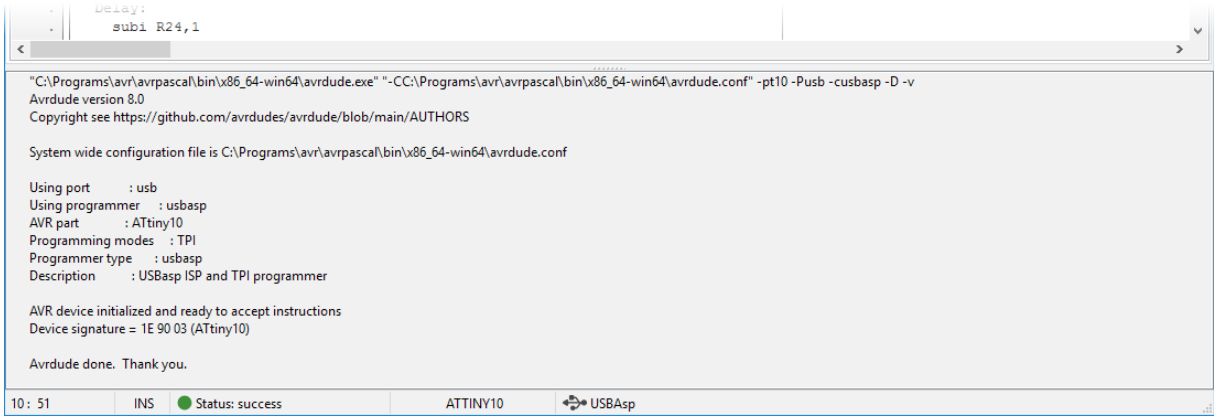




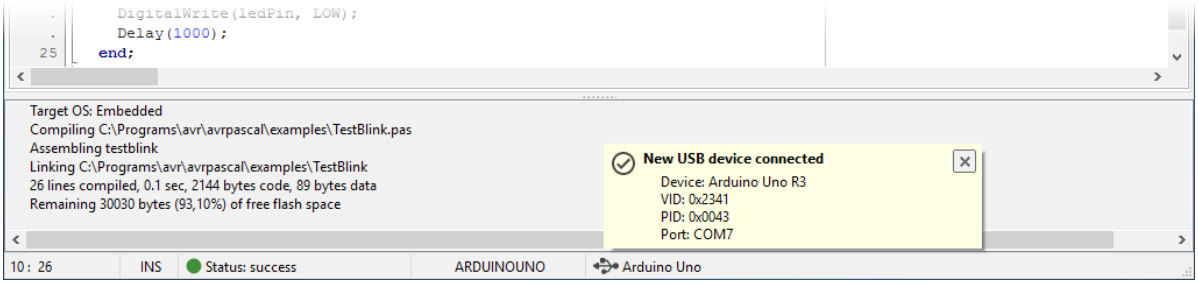
AVRPascal also supports importing microcontroller flash memory or EEPROM contents to a file (*Tools->Download to File* menu) and exporting data from a file to the microcontroller's flash memory or EEPROM (*Tools->Upload from File* menu). These features utilize the global microcontroller type and supported file format settings previously configured in the *Options* window. In addition, as with uploading, there are limitations regarding the supported interfaces.



Moreover, via the *Tools* menu it is possible to read and write the fuse values of microcontrollers supporting USBasp. When fuse values are read from the microcontroller (*Tools->Get Device Fuses*), the corresponding values (byte0/low/high/extended, among others) are displayed in the Messages area. Writing fuse values (*Tools->Set Device Fuse*) is performed through an extended dialog box for modifying individual bits.[21] When fuse-bit values other than the default settings are used, the descriptions provided in the microcontroller datasheet should be consulted carefully, as incorrect settings may prevent further programming via USBasp.



The *Tools->Check Device* function verifies that the microcontroller selected in the program settings matches the device connected to the programmer. To do this, the device signature is read and compared with the signature expected for the selected microcontroller.



An additional option of AVRPascal is the detection of USB devices supported by the application (USBasp, Arduino boards). This option improves work primarily with Arduino boards, in the case of using the USBasp programmer it has no major significance[22]. It can be activated in the Options window in the Uploader tab by checking *Notify the connection or disconnection of a supported device*[23]. Notifications about connecting a new device or disconnecting a previously connected one will appear just above the icon (USB connector symbol) in the status bar. Next to this icon USB devices currently connected to the computer are listed.

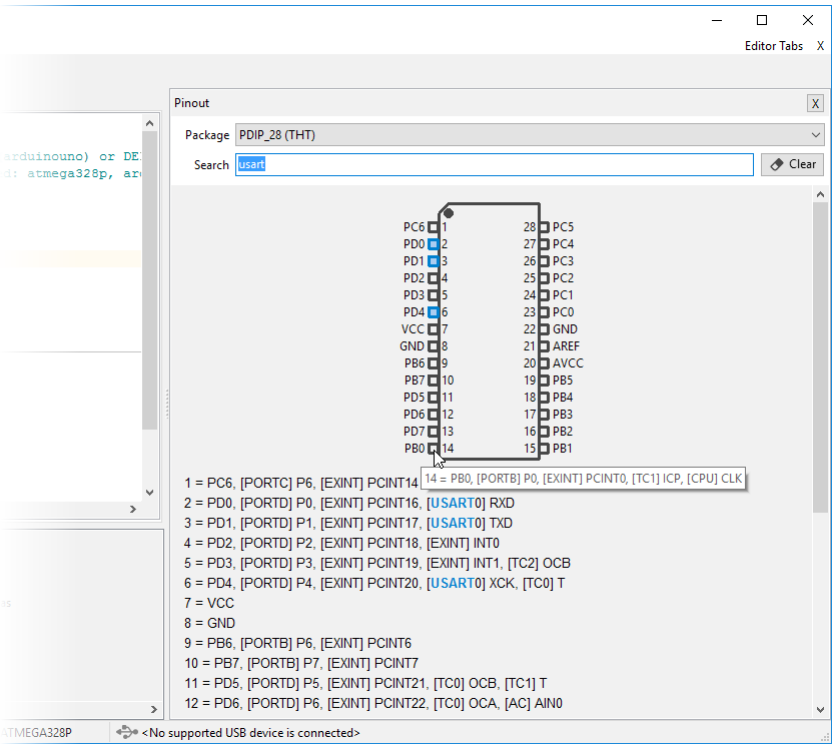
21 The Set Device Fuse dialog writes one selected fuse at a time.
22 This is because AVRdude automatically detects the USBasp programmer, while Arduino boards require explicit selection of the virtual serial port assigned to the board.
23 Activating or deactivating notifications will take effect after restarting the application.

AVRPascal has a list of predefined USB devices with their VID and PID (vendor and product identifiers), which allow the identification of a given device as a USBasp programmer or Arduino board. This list is also available in the *Uploader* tab and can be supplemented with custom device definitions, so that AVRPascal will be able to detect, for example, Arduino clones. The only limitation is the PID and VID, which must be unique for each definition.

An improvement resulting from the USB device detection option, apart from the informational value itself, is the suggestion of the serial port number/name associated with the required Arduino board in the *Upload to Device* and *Download from Device* dialog windows. Thanks to this, the user does not have to select the serial port number/name from the list currently available in the operating system.

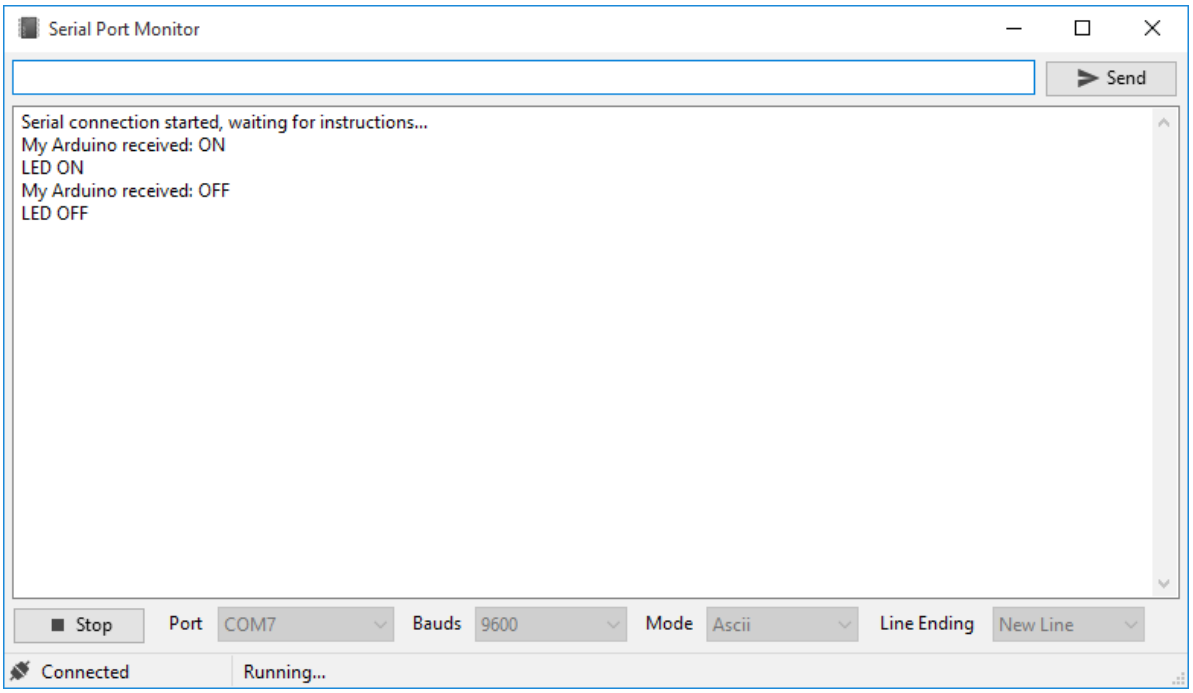
6. Additional features

The program also includes several auxiliary features that simplify working with AVR microcontrollers and Arduino boards. One of them is the pinout viewer for the currently selected microcontroller, available from the *View->Pinout* menu. Device definitions include THT, SMD and BGA packages, as well as detailed pin descriptions presented in the legend. When the mouse cursor is positioned over a pin, detailed information about that pin is displayed in a hint window. It is also possible to search for text within the legend; matching signals and their corresponding pins are highlighted after confirming the search with the ENTER key.



The pinout viewer makes it possible to quickly verify the assignment of peripheral functions (such as UART, SPI, I2C, PWM, ADC or external interrupts) to individual microcontroller pins without the need to consult the manufacturer's documentation.

Another auxiliary feature is the serial port monitor[24], available from the *View->Serial Port Monitor* menu.

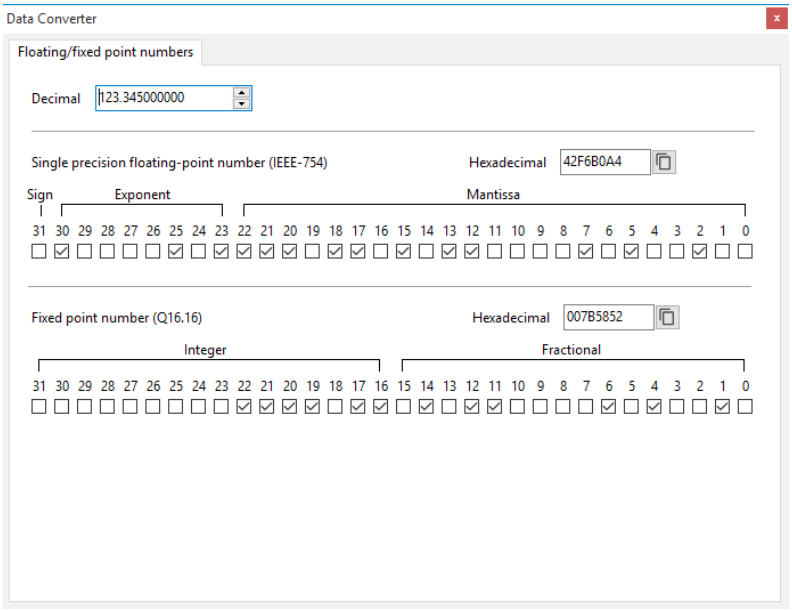


To track data received via a serial port, select it in the bottom panel of *Serial Port Monitor* window and specify the speed of data transmission (i.e. baud rate; it must match the speed at which the connected device sends data). Other, more detailed connection options are located in the *Options* window, *Others* tab.

Additionally, the *Serial Port Monitor* window includes the following options: *Mode*, which selects between character (*ASCII*) and numeric (*hexadecimal*) representation of received data, and *Line Ending*, which determines the termination characters appended to transmitted data. Sent data can be terminated with no additional character (*No*), a new line (*New Line*), a carriage return (*Carriage Return*), or both (*NL and CR*).

Additional features also include the *Data Converter* window, accessible via the *View->Data Converter* menu. This tool is designed for converting various data types.

24 Serial transmission is an extremely useful and convenient mechanism for communication between a microcontroller or Arduino and a computer. It can also be used to verify the correctness of a program running on a given microcontroller.



The Floating/Fixed Point Numbers tab provides conversion between decimal floating-point numbers and their IEEE-754 hexadecimal and binary representations (single-precision floating-point format). The corresponding Q16.16 fixed-point value is also displayed in hexadecimal and binary form. Numeric conversion is interactive: editing any numeric field or checkbox automatically updates all other controls. Conversion between floating-point and fixed-point numbers is made easier by using the TFloat32 and TFix16 types from the UnoLib library (see Chapter 7).

7. UnoLib library

The AVRPascal installer includes the core modules of UnoLib[25], an open-source library written in Pascal for the Arduino Uno platform. UnoLib translates part of the standard Arduino library, with the necessary modifications, to facilitate programming the Arduino Uno board in AVRPascal using familiar, Arduino-compatible function names. Modules included:

- *analog.pas* - support for analog pins
- *defs.pas* - definitions of constants, bit manipulations, port support
- *dht.pas* - support for DHT11/22 sensors
- *digital.pas* - support for digital pins
- *ds1302rtc.pas* - support for DS1302 real time clock
- *fix16.pas* - support for fixed point numbers
- *float32.pas* - support for floating point numbers (in collaboration with @Dzandaa)
- i2c.pas - support for I2C communication bus (by @Dzandaa, thanks to @ccrause)
- *hardwareserial.pas* - support for serial communication
- *liquidcrystal.pas* - support for LCD
- pulse.pas - routines for reading a pulse on a pin
- *spi.pas* - support for SPI interface
- stringutils.pas - string conversion routines (by @Dzandaa)

25 https://sourceforge.net/projects/unolib/

- *timer.pas* - time-related routines.

These modules are located in the AVRPascal *lib* directory and their documentation in the form of pdf files is located in the *docs* directory. In turn, the *examples* directory contains simple sample programs using UnoLib modules[26]:

- *ArduinoISP.pas* - simple ISP programmer
- *DS137ZN_RTC_Test.pas* - Real Time Clock test using I2C (by @Dzandaa)
- *HMC5883L_Magnetometer_Test.pas* - HMC5883L Magnetometer test using I2C (by @Dzandaa)
- *I2CScan.pas* - I2C bus scan (by @Dzandaa)
- *pcf8591t_ACDC_Read.pas* - ACDC read test (by @Dzandaa)
- *pcf8591t_ACDC_Write.pas* - ACDC write test (by @Dzandaa)
- *TestBlink.pas* – turns on and off the built-in LED
- *TestBlinkWithoutDelay.pas* - turns on and off the built-in LED using Millis
- *TestDHT11.pas* – displays information about the temperature and humidity of the air from the DHT11 sensor on an external LCD display
- *TestDigital.pas* – turns on and off the built-in LED based on the button state
- TestHC-SR04 – example of using HC-SR04 ultrasonic sensor
- *TestLCAutoscroll.pas* – scrolls text on the LCD display
- *TestLCBlink.pas* – displays the text “hello, world!” on the LCD display
- *TestLCChars.pas* – displays non-standard characters on the LCD display
- *TestLCCursor.pas* – turns the cursor on and off on the LCD display
- *TestLCDisplay.pas* – displays and turns off the text “hello, world!”
- *TestLCSerialDisplay.pas* – displays characters taken from the serial port on the LCD display
- *TestLCTextDirection.pas* – changes the direction of text display on the LCD display
- *TestLM35.pas* – displays the temperature value from the LM35 sensor on the LCD display
- *TestSerial.pas* – sends and receives data via the serial port
- *TestTone.pas* – plays a melody using _tone routine.

8. Updates

AVRPascal updates are regularly released on the "Electronics" page of akarwowski.pl. Announcements about new versions are also available in the “News” section of the website. Each installer includes a changelog detailing the latest improvements and fixes[27].

AVRPascal includes a built-in update check function accessible through the *Help->Check for Updates* menu command. An internet connection is required. If a newer version is available, AVRPascal displays a notification. Updates are not applied automatically and require downloading and running the installer from the project website.

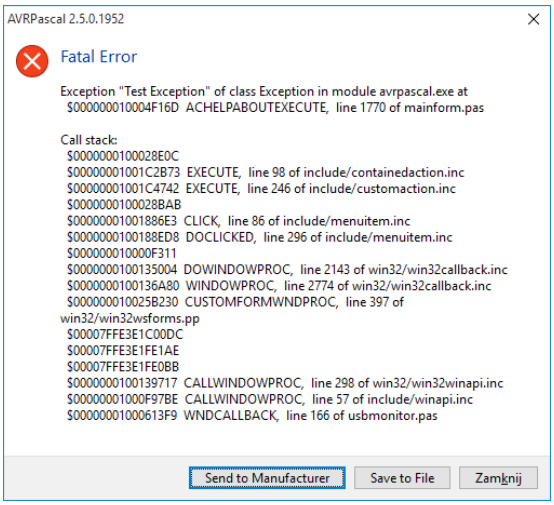
26 Before compiling a given program, make sure that the library modules listed in the *uses* section have already been compiled.

27 The description of current changes and improvements in the program is available on the akarwowski.pl website in the file AVRPascal_Changes.txt.

9. User feedback

Users of the program can actively influence the direction of development of AVRPascal. Any suggestions and comments regarding the program's operation, including critical ones, are welcome. They can be reported by e-mail to the author at ackarwow@gmail.com.

AVRPascal can capture information about program crashes that might cause instability. These crash reports typically contain details like the location of the error and the procedures leading to it.



When a crash occurs, a crash report window is displayed. The report can be ignored, saved to a file, or submitted directly to the author using the *Send to Manufacturer* function. Report submission requires an internet connection. Crash reports are sent to akarwowski.pl.

Moreover, in the *Options* window, in the *Compiler* and *Uploader* tabs, there are options to show the command line. After selecting them, the command line of the compiler and uploader will appear in the *Messages* area, respectively, with all the parameters used by AVRPascal, both user-defined and internally used. This feature can be helpful if there is a need to verify the correctness of the program's calling the compiler or uploader.

10. Licence

AVRPascal is free of charge (*freeware – closed source*) and may be used for any legally permissible purpose (e.g., hobby, educational, commercial). Commercial use includes the creation, distribution, and sale of software and devices with source or binary code generated within AVRPascal, provided the licence terms of external libraries, such as UnoLib, are respected. For educational and commercial projects, it is recommended to state that the code was created using AVRPascal (e.g., in the documentation or product description). Free redistribution is allowed only for the unmodified installer. It is recommended to provide the official download link for the installer from akarwowski.pl. Inclusion into other software

packages requires the author's prior written consent. Selling the AVRPascal environment itself is prohibited.

The AVRPascal software is provided "as is," without a guarantee of technical support. This means that while the author has made every effort to ensure the program functions correctly, they offer no warranties and shall not be liable for any damages (direct, indirect, incidental, or consequential) arising from the use or inability to use the software.

In the interest of the tool's continued development, the Author encourages all AVRPascal users to report any bugs and suggest new features. This feedback is crucial for improving AVRPascal's stability and usability.

This project is independent and not affiliated with the probably discontinued software available at avrpascal.com.

AVR® is a registered trademark of Microchip Technology Inc., used here for descriptive purposes only.

Andrzej Karwowski

email: ackarwow@gmail.com

Appendix 1. Supported AVR microcontrollers

AT90CAN128	*ATMEGA2560*	*ATMEGA645*	*ATTINY212*
AT90CAN32	*ATMEGA2561*	*ATMEGA6450*	*ATTINY214*
AT90CAN64	*ATMEGA2564RFR2*	*ATMEGA6450A*	*ATTINY2313*
AT90PWM1	*ATMEGA256RFR2*	*ATMEGA6450P*	*ATTINY2313A*
AT90PWM161	*ATMEGA32*	*ATMEGA645A*	*ATTINY24*
AT90PWM216	*ATMEGA3208*	*ATMEGA645P*	*ATTINY24A*
AT90PWM2B	*ATMEGA3209*	*ATMEGA649*	*ATTINY25*
AT90PWM316	*ATMEGA324A*	*ATMEGA6490*	*ATTINY26*
AT90PWM3B	*ATMEGA324P*	*ATMEGA6490A*	*ATTINY261*
AT90PWM81	*ATMEGA324PA*	*ATMEGA6490P*	*ATTINY261A*
AT90USB1286	*ATMEGA324PB*	*ATMEGA649A*	*ATTINY28*
AT90USB1287	*ATMEGA325*	*ATMEGA649P*	*ATTINY3214*
AT90USB162	*ATMEGA3250*	*ATMEGA64A*	*ATTINY3216*
AT90USB646	*ATMEGA3250A*	*ATMEGA64C1*	*ATTINY3217*
AT90USB647	*ATMEGA3250P*	*ATMEGA64HVE2*	*ATTINY4*
AT90USB82	*ATMEGA3250PA*	*ATMEGA64M1*	*ATTINY40*
ATA6285	*ATMEGA325A*	*ATMEGA64RFR2*	*ATTINY402*
ATA6286	*ATMEGA325P*	*ATMEGA8*	*ATTINY404*
ATMEGA128	*ATMEGA325PA*	*ATMEGA808*	*ATTINY406*
ATMEGA1280	*ATMEGA328*	*ATMEGA809*	*ATTINY412*
ATMEGA1281	*ATMEGA328P*	*ATMEGA8515*	*ATTINY414*
ATMEGA1284	*ATMEGA328PB*	*ATMEGA8535*	*ATTINY416*
ATMEGA1284P	*ATMEGA329*	*ATMEGA88*	*ATTINY416AUTO*
ATMEGA1284RFR2	*ATMEGA3290*	*ATMEGA88A*	*ATTINY417*
ATMEGA128A	*ATMEGA3290A*	*ATMEGA88P*	*ATTINY4313*
ATMEGA128RFA1	*ATMEGA3290P*	*ATMEGA88PA*	*ATTINY43U*
ATMEGA128RFR2	*ATMEGA3290PA*	*ATMEGA88PB*	*ATTINY44*
ATMEGA16	*ATMEGA329A*	*ATMEGA8A*	*ATTINY441*
ATMEGA1608	*ATMEGA329P*	*ATMEGA8HVA*	*ATTINY44A*
ATMEGA1609	*ATMEGA329PA*	*ATMEGA8U2*	*ATTINY45*
ATMEGA162	*ATMEGA32A*	*ATTINY10*	*ATTINY461*
ATMEGA164A	*ATMEGA32C1*	*ATTINY102*	*ATTINY461A*
ATMEGA164P	*ATMEGA32HVB*	*ATTINY104*	*ATTINY48*

ATMEGA164PA
ATMEGA165A
ATMEGA165P
ATMEGA165PA
ATMEGA168
ATMEGA168A
ATMEGA168P
ATMEGA168PA
ATMEGA168PB
ATMEGA169A
ATMEGA169P
ATMEGA169PA
ATMEGA16A
ATMEGA16HVA
ATMEGA16HVB
ATMEGA16HVBREVB
ATMEGA16M1
ATMEGA16U2
ATMEGA16U4
ATMEGA32HVBREVB
ATMEGA32M1
ATMEGA32U2
ATMEGA32U4
ATMEGA406
ATMEGA48
ATMEGA4808
ATMEGA4809
ATMEGA48A
ATMEGA48P
ATMEGA48PA
ATMEGA48PB
ATMEGA64
ATMEGA640
ATMEGA644
ATMEGA644A
ATMEGA644P
ATMEGA644PA
ATMEGA644RFR2
ATTINY11
ATTINY12
ATTINY13
ATTINY13A
ATTINY15
ATTINY1604
ATTINY1606
ATTINY1607
ATTINY1614
ATTINY1616
ATTINY1617
ATTINY1624
ATTINY1626
ATTINY1627
ATTINY1634
ATTINY167
ATTINY20
ATTINY202
ATTINY204
ATTINY5
ATTINY804
ATTINY806
ATTINY807
ATTINY814
ATTINY816
ATTINY817
ATTINY828
ATTINY84
ATTINY841
ATTINY84A
ATTINY85
ATTINY861
ATTINY861A
ATTINY87
ATTINY88
ATTINY9