UnoLib documentation

hardwareserial.pas

version 11/10/2025

The hardwareserial.pas module contains implementation of THardwareSerial class, providing the interface for Universal Asynchronous Receiver/Transmitter (UART) serial communication using the dedicated hardware peripherals on the microcontroller. This holds communication speed register setting, communication control register setting, transmit and receive data registers, transmit and receive buffers.

Since FPC for AVR does not allow dynamic object creation, an object of type THardwareSerial declared in the var section is ready to use and does not require the use of a constructor.

THardwareSerial methods
`procedure` **`Start`**`(const aBaud: UInt32);` Initializes the serial port by setting the data rate (baud rate), such as 9600 or 115200. *Parameters* *aBaud* – baud rate number of the serial port initialization settings Note: In Arduino sources the method is called *begin*.
`procedure` **`Stop;`** Disables the serial port. Note: In Arduino sources the method is called *end*.
`function` **`Available:`** `byte;` Returns the number of unread bytes (characters) that have arrived and are currently stored in the serial receive buffer. This function is crucial for non-blocking program logic, allowing the program to perform other tasks in the main loop function without waiting indefinitely for data to arrive.
`function` **`AvailableForWrite:`** `byte;` Determines how much data can be written to the serial port's transmit buffer without blocking the program. It returns the number of bytes of free space in the transmit buffer.
`function` **`PeekChar:`** `char;` Peeks at the first byte in the receive (RX) buffer. It is used to look at the next available byte of incoming serial data without removing it from the serial buffer. Returns the first char (byte) of incoming serial data, or #0 if no data is available. Note: In Arduino sources the method is called *peek*. The original function returns type *int*, and

-1 if no data is available.

```
function ReadChar: char;
```

Reads the oldest incoming byte as char from the receive buffer. The byte is removed from the buffer and cannot be read again. Returns #0 if no data is available.

Note: extra function, not present in Arduino sources.

```
function ReadByte: byte;
```

Reads the oldest incoming byte from the receive buffer. The byte is removed from the buffer and cannot be read again. Returns 0 if no data is available.

Note: extra function, not present in Arduino sources.

```
function Read: Int16;
```

Reads the oldest incoming byte as Int16 from the receive buffer. The byte is removed from the buffer and cannot be read again. Returns -1 if no data is available.

```
function ReadBuff(const aBuff: PUInt8; const aLen: UInt8): boolean;
```

Reads the number of bytes specified by the *aLen* parameter and stores the result in the *aBuff* buffer. Returns true if success, otherwise false.

Note: extra function, not present in Arduino sources.

```
procedure Flush;
```

Waits for the transmission of outgoing data to complete.

```
procedure WriteChar(const aChar: char);
```

Sends the character specified by the *aChar* parameter through the TX pin.

Note: extra procedure, not present in Arduino sources.

```
procedure Write(const aStr: shortstring);
```

Sends the string specified by the *aStr* parameter through the TX pin.

Note: In Arduino sources the method is called *print*. The original procedure passes raw bytes (binary data) while the *aStr* shortstring parameter consumes 256 bytes.

```
procedure WriteLn(const aStr: shortstring);
```

Sends the string specified by the *aStr* parameter through the TX pin appending a newline character (#10) to the end of the data.

Note: In Arduino sources the method is called *println*. The original procedure passes raw bytes (binary data) while the *aStr* shortstring parameter consumes 256 bytes.

```
procedure WriteBuff(const aBuff: PUInt8; const aLen: UInt8);
```

Sends number of bytes specified in *aLen* parameter located in *aBuff* data buffer.

Note: extra function, not present in Arduino sources.

Example

See documentation of float32.pas module.