
Reference Manual

E-LAB AVRco

Pascal Multi-Tasking for Single Chips

Version for

AVR

© Copyright 1996-2011 by E-LAB Computers



Blaise Pascal Mathematician 1623-1662

The content of this user guide is copyright protected by E-LAB Computers.

Autor Rolf Hofmann
Editor Gunter Baab

E-LAB

Computers

Mikroprozessor-Technik
Industrie-Elektronik
Hard + Software
8-Bit • 16-Bit • 32-Bit

E-LAB Computers
Grombacherstr. 27
D74906 Bad Rappenau
Tel 07268/9124-0
Fax 07268/9124-24
<http://www.e-lab.de>
info@e-lab.de

Important Information

Everybody tries to write Software without bugs. The emphasis is on tries, because everybody knows that the more complex a Software is, the more likely it is to produce bugs.

We have the opinion, that this shouldn't have to be norm, and that we do not have to live with the problems and mistakes (although some Software giants think like that J).

If you should find any errors, we would be thankful for any information. We will try to solve any problems as quickly as possible.

It is also a normal international agreement that the software producer does not accept liability for any costs arising out of errors in software, unless otherwise agreed.

E-LAB Computers do not accept liability for costs resulting out of errors in the software. It is a condition of use of this Software you agree with these terms. If you do not agree, you are not permitted to use the software.

As we have said, before this exclusion of liability is international standard.

This user guide and the software is intellectual property from E-LAB Computers and therefore copyright protected.

This document and the software it relates to are solely for the use of the purchaser. The purchaser is not permitted to give, sell or distribute these products. Distributing copies of these products to a third party is strictly prohibited.

We like to think that you as user of the software can make money from it and therefore also expect maintenance of the product. Illegal copies would make it impossible for us to be able to maintain this service.

As you see it is also in the interest of you, the user, to observe the copyright.

.

That's it the author

1 AVRco Help

1.1 Overview Compiler Switches

Compiler Switches

<u>\$ANALYSIS ON</u>	<u>\$IFEND</u>	<u>\$OVERLAY</u>
<u>\$BDATA</u>	<u>\$IFNDEF</u>	<u>\$PCU</u>
<u>\$BootApplication</u>	<u>\$J</u>	<u>\$PDATA</u>
<u>\$BOOTRST</u>	<u>\$LCDNOINIT</u>	<u>\$PHASE</u>
<u>\$CodeStart</u>	<u>\$LCDNOWAIT</u>	<u>\$Q</u>
<u>\$D</u>	<u>\$MODBUS</u>	<u>\$REUTILIZE</u>
<u>\$DATA</u>	<u>\$NOADDRCHECK</u>	<u>\$SHOWERROR</u>
<u>\$DEBDELAY</u>	<u>\$NOFRAME</u>	<u>\$SHOWWARNING</u>
<u>\$DEFINE</u>	<u>\$NOINIT</u>	<u>\$SL</u>
<u>\$DEPHASE</u>	<u>\$NOOVRCHECK</u>	<u>\$TYPEDCONST</u>
<u>\$DEVICE</u>	<u>\$NORAMCHECK</u>	<u>\$UDATA</u>
<u>\$EEPROM</u>	<u>\$NOREGSAVE</u>	<u>\$UNDEF</u>
<u>\$EEPROM1</u>	<u>\$NORETURNCHECK</u>	<u>\$VALIDATE</u>
<u>\$ELSE</u>	<u>\$NOSAVE</u>	<u>\$VALIDATE ALL</u>
<u>\$ELIF</u>	<u>\$NOSHADOW</u>	<u>\$VALIDATE OFF</u>
<u>\$ELIFDEF</u>	<u>\$NOWATCHDOGAUTO</u>	<u>\$VALIDATE ON</u>
<u>\$ENDIF</u>	<u>\$OPTIMISE</u>	<u>\$VectTab</u>
<u>\$ENUMTOASM</u>	<u>\$OPTI ALLOW INLINE</u>	<u>\$W</u>
<u>\$HEXNAME</u>	<u>\$OPTI BETA OFF</u>	<u>\$WG</u>
<u>\$HEXPATH</u>	<u>\$OPTI CHECK RETURN REGS</u>	<u>\$X</u>
<u>\$I</u>	<u>\$OPTI NO ALLOW INLINE</u>	<u>\$XDATA</u>
<u>\$IDATA</u>	<u>\$OPTI NO CHECK RETURN REGS</u>	<u>\$XIO</u>
<u>\$IDATA1</u>	<u>\$OPTI NO CSE OPT</u>	<u>\$ZEROLOCVARS</u>
<u>\$IF</u>	<u>\$OPTI QUICK</u>	
<u>\$IFDEF</u>	<u>\$OPTI SMARTLINK ONLY</u>	

1.2 Overview Maths

Maths

Abs	Inc	Mirror32	SinInt
ArcTan	IncToLim	Mirror8	SinInt16
BCDtoByte	IncToLimWrap	MulDivByte	SizeOf
ByteToBCD	Int	MulDivInt	SquareDivByte
CalcCheckSum	IntegrateB	MulDivInt8	SquareDivInt
Cos	IntegrateI	MulDivLong	SquareDivInt8
CosD	IntegrateI8	Negate	Sqr
CosInt	IntegrateW	Odd	Sqr
CosInt16	IntToFix64	Ord	Succ
Dec	IsPowOfTwo	Parity	Swap
DecToLim	Lo	Pow	SwapLong
DecToLimWrap	LogN	Pow10	Tan
DegToRad	Log10	Pred	TanD
Even	LoNibble	RadToDeg	Trunc
Exp	Lower	RotatePntI	ValueInRange
Frac	LoWord	Round	ValueInTolerance
Hi	LowPassFW	Sgn	ValueInToleranceP
Higher	Max	Sign	ValueTrimLimit
HiNibble	Min	Sin	Within
HiWord	Mirror16	SinD	WordToBCD

1.3 Overview Fix64

Fix64 type must be imported: **from** system **import** ..., **Fix64**;

Following complex Fix64 functions (higher Math, Trigonometry, Logarithm etc) are contained in an unit that must also be specified if needed: **uses** uFix64;

Less complex functions (like standard maths) are part of the compiler itself and already available in the *Standard Version* of the AVRco.

(see [Overview Maths](#)^[4]).

Fix64ArcCos	Fix64ArcTanh	Fix64Ln	Fix64SinD
Fix64ArcCosD	Fix64Cos	Fix64Log	Fix64Sinh
Fix64ArcCosh	Fix64CosD	Fix64Log10	Fix64Sqrt
Fix64ArcCot	Fix64Cosh	Fix64LogN	Fix64Tan
Fix64ArcCotD	Fix64Cot	Fix64Mod	Fix64TanD
Fix64ArcCsc	Fix64CotD	Fix64ModInt	Fix64Tanh
Fix64ArcCscD	Fix64Csc	Fix64MulInt	Fix64ToFloat
Fix64ArcSec	Fix64CscD	Fix64MulLong	Fix64ToHex
Fix64ArcSecD	Fix64DegToRad	Fix64Odd	Fix64ToInt
Fix64ArcSin	Fix64DegToRadD	Fix64Power	Fix64ToLongInt
Fix64ArcSinD	Fix64DivInt	Fix64PowerInt	Fix64ToStr
Fix64ArcSinh	Fix64DivLong	Fix64Quadrant	Fix64ValueInTolerance
Fix64ArcTan	Fix64Even	Fix64RadToDeg	Fix64ValueInToleranceP
Fix64ArcTanD	Fix64Exp	Fix64Sec	FloatToFix64
Fix64ArcTan2	Fix64Integrate	Fix64SecD	IntToFix64
Fix64ArcTan2D	Fix64IsPowOfTwo	Fix64Sin	StrToFix64

1.4 Overview Misc

Various Functions

<u>AddAVFilter</u>	<u>GetAVfilter</u>	<u>Read</u>	<u>TestDeviceLock</u>
<u>BitCountOf</u>	<u>GetLargestBlock</u>	<u>ReadLn</u>	<u>Trap</u>
<u>DeclAVfilter</u>	<u>GetMem</u>	<u>ResetSysTimer</u>	<u>WatchDogStart</u>
<u>FlashDownLoader</u>	<u>GetMemAvail</u>	<u>SetAVfilter</u>	<u>WatchDogStop</u>
<u>FlashLoaderExit</u>	<u>InterPolX</u>	<u>SetDeviceLock</u>	<u>WatchDogTrig</u>
<u>FlashLoaderInit</u>	<u>InterPolY</u>	<u>SetSysTimer</u>	<u>Write</u>
<u>FlashLoaderRecv</u>	<u>isSysTimerZero</u>	<u>SetSysTimerM</u>	<u>WriteLn</u>
<u>FlashLoaderTransm</u>	<u>I2CexpStat</u>	<u>SizeOf</u>	
<u>FloatAsLong</u>	<u>LongAsFloat</u>	<u>SwapIPaddr</u>	
<u>FreeMem</u>	<u>PresetAVfilter</u>	<u>SwapMACaddr</u>	

1.5 Overview MultiTasking

Multi Tasking

<u>ClearDeviceLock</u>	<u>IsCurProcess</u>	<u>Schedule</u>	<u>Suspend</u>
<u>DecSema</u>	<u>Lock</u>	<u>SchedulerOff</u>	<u>SuspendAll</u>
<u>GetCurProcess</u>	<u>Priority</u>	<u>SchedulerOn</u>	<u>TestDeviceLock</u>
<u>GetPriority</u>	<u>ProcWaitFlag</u>	<u>SemaStat</u>	<u>UnLock</u>
<u>GetProcessState</u>	<u>ResetProcess</u>	<u>SetDeviceLock</u>	<u>WaitPipe</u>
<u>GetProcessID</u>	<u>Resume</u>	<u>SetSema</u>	<u>WaitSema</u>
<u>IncSema</u>	<u>ResumeAll</u>	<u>Start Processes</u>	<u>WaitDeviceFree</u>

1.6 Overview Reserved Words

reserved Keywords

<u>absolute</u>	<u>define_usr</u>	<u>end_try</u>	<u>in</u>	<u>pointer</u>	<u>table</u>
<u>and</u>	<u>device</u>	<u>end_while</u>	<u>inherit</u>	<u>procedure</u>	<u>task</u>
<u>array</u>	<u>devicelock</u>	<u>end_with</u>	<u>inherited</u>	<u>process</u>	<u>then</u>
<u>asm</u>	<u>div</u>	<u>eof</u>	<u>initialization</u>	<u>program</u>	<u>to</u>
<u>assign</u>	<u>do</u>	<u>eoln</u>	<u>integer</u>	<u>record</u>	<u>true</u>
<u>at</u>	<u>downto</u>	<u>except</u>	<u>interface</u>	<u>repeat</u>	<u>try</u>
<u>begin</u>	<u>else</u>	<u>exec</u>	<u>interrupt</u>	<u>return</u>	<u>type</u>
<u>Bit</u>	<u>elsif</u>	<u>exitloop</u>	<u>int64</u>	<u>rol</u>	<u>unit</u>
<u>bitset</u>	<u>els_if</u>	<u>exit_loop</u>	<u>int8</u>	<u>ror</u>	<u>until</u>
<u>boolean</u>	<u>end</u>	<u>export</u>	<u>label</u>	<u>semaphore</u>	<u>userdevice</u>
<u>break</u>	<u>endasm</u>	<u>false</u>	<u>locked</u>	<u>sendsema</u>	<u>uses</u>
<u>by</u>	<u>endcase</u>	<u>file</u>	<u>longint</u>	<u>set</u>	<u>using</u>
<u>byte</u>	<u>endfor</u>	<u>finalization</u>	<u>longword</u>	<u>shl</u>	<u>val</u>
<u>case</u>	<u>endif</u>	<u>float</u>	<u>loop</u>	<u>shla</u>	<u>var</u>
<u>char</u>	<u>endloop</u>	<u>for</u>	<u>mod</u>	<u>shortint</u>	<u>variant</u>
<u>chr</u>	<u>endtry</u>	<u>forward</u>	<u>nil</u>	<u>shr</u>	<u>while</u>
<u>class</u>	<u>endwhile</u>	<u>from</u>	<u>not</u>	<u>shra</u>	<u>with</u>
<u>close</u>	<u>endwith</u>	<u>function</u>	<u>object</u>	<u>str</u>	<u>word</u>
<u>const</u>	<u>end_asm</u>	<u>goto</u>	<u>of</u>	<u>string</u>	<u>word64</u>
<u>continue</u>	<u>end_case</u>	<u>idle</u>	<u>or</u>	<u>structconst</u>	<u>xor</u>
<u>define</u>	<u>end_for</u>	<u>if</u>	<u>override</u>	<u>systimer</u>	
<u>definefrom</u>	<u>end_if</u>	<u>implementation</u>	<u>pidcontrol</u>	<u>systimer32</u>	
<u>define_fuses</u>	<u>end_loop</u>	<u>import</u>	<u>pipe</u>	<u>systimer8</u>	

1.7 Overview Strings

Strings

<u>Append</u>	<u>Fix64ToHex</u>	<u>Long64ToStr</u>	<u>StrToFix64</u>
<u>ArrToStr</u>	<u>Fix64ToStr</u>	<u>LowerCase</u>	<u>StrToFloat</u>
<u>BoolToStr</u>	<u>HexToInt</u>	<u>LowerCase</u>	<u>StrToInt</u>
<u>ByteToBin</u>	<u>Insert</u>	<u>MACToStr</u>	<u>StrToIP</u>
<u>ByteToHex</u>	<u>IntToBin</u>	<u>PadLeft</u>	<u>StrtoMAC</u>
<u>ByteToStr</u>	<u>IntToHex</u>	<u>PadRight</u>	<u>Trim</u>
<u>Copy</u>	<u>IntToStr</u>	<u>Pos</u>	<u>TrimLeft</u>
<u>Delete</u>	<u>IPtoStr</u>	<u>PosN</u>	<u>TrimRight</u>
<u>ExtractFileExt</u>	<u>Length</u>	<u>SetLength</u>	<u>UpCase</u>
<u>ExtractFileName</u>	<u>LongToHex</u>	<u>StrClean</u>	<u>UpperCase</u>
<u>ExtractFilePath</u>	<u>LongToStr</u>	<u>StrReplace</u>	
<u>FloatToStr</u>	<u>Long64ToHex</u>	<u>StrToArr</u>	

1.8 Overview System

System Functions and Procedures

Addr	DecSema	mDelay	RunTimeErr
Bit	DisableInts	NoInts	sDelay
BootRestart	Disable_JTAGport	NOP	SemaStat
Boot_Init	EEPromPtr	OnADCread	SetBit
CalcFlashCheck	EnableInts	OnFAT16_SS	SetSema
CalcFlashCheck_A	EnableIntsX	OnIdleProcess	SetTable
CalcFlashCheck_B	Enable_JTAGport	OnSchedulerEntry	SetVectTabBoot
CalcFlashCheck_S	Excl	OnSchedulerExit	Sleep
CheckFrameValid	FillBlock	OnSysTick	Start_Processes
CheckStackValid	FillRandom	OnTickTimer	System_Init
ClearRunErr	FlashPtr	OnTINA_SS	System_MCUCR_Init
CompareBlock	GetExceptResult	PopAllRegs	System_Reset
CompareIP	GetFrameFree	PopRegs	System_ShutDown
CompareNet	GetStackFree	PushAllRegs	SysTickDisable
CompareMAC	GetSysTimer	PushRegs	SysTickEnable
CopyBlock	GetTable	PowerSave	SysTickRestart
CPUSleep	GetTaskFrameFree	RaiseException	SysTickStop
CRCcheck	GetTaskStackFree	Random	Toggle
CRCstreamAdd	GetWatchDogFlag	RandomRange	uDelay
CRCstreamAddP	Incl	RestoreInts	uDelay_1
CRCstreamInit	IncSema	RunErr	UsrDevPtr

1.9 ADC Port

Import SysTick, ADCPort;

Define

```

ProcClock    = 4000000;           {4Mhz clock }
SysTick      = 10;                {10msec Tick}
ADCchans     = 2, iData;          {use channel 1 and 2}
//ADCchans   = 2, iData, int2;    {2 times integrate}
//ADCchans   = [4, 7], iData;     {channel 4 and 7}
//ADCchans   = [2..5], iData;     {channel 2, 3, 4, 5}
//ADCchans   = [5], iData;        {only channel 5}
ADCpresc     = 16;                {prescaler 16}

```

Attention: AVRco counts the channels 1, 2, 3, ... (corresponding to controller count 0, 1, 2, ...)

Funktionen und Prozeduren

[GetAdc](#) [OnADCread](#) [SetAdcFixed](#)

1.10 AVR CAN

Import SysTick, CAN_AVR ;

From SysTick import SystemTime16; // or SystemTime32 (option)

Define

```

ProcClock    = 16000000;          {16Mhz clock }
CAN_AVR      = 16, 16, iData;     {RxPipe, TxPipe}

```

```

// CAN_AVR          = 16, 16, iData, CAN_SysTime; // RxPipe, TxPipe, memory,
// Systemtime (option)

CAN_AVRbaud        = CAN_Baud1000;

type
  tCANMessage11 = record
    MOBIdx      : byte;           // Message Object = Maibox number
    MsgIdent    : word;           // Identifier ~ Accept mask 11bits
    DLC         : byte;           // data length 0..8
    data        : array [0..7] of byte;
  end;

  tCAN_baud      = (CAN_Baud25, CAN_Baud50, CAN_Baud100, CAN_Baud125,
    CAN_Baud200,
    CAN_Baud250, CAN_Baud500, CAN_Baud1000);
  tAVR_CAN_Stat  = (CAN_ACKError, CAN_FrameError, CAN_CRCError, CAN_StuffError,
    CAN_BitError, CAN_RxOK, CAN_TxOK, CAN_DLCwarn);
  tAVR_CAN_States = BitSet of tAVR_CAN_Stat;

var
  CAN_RxPipe : Pipe [defined] of tCANMessage11;
  CAN_TxPipe : Pipe [defined] of tCANMessage11;

```

Functions and Procedures

<u>AVR_CAN_BaudRate</u>	<u>AVR_CAN_GetStatus</u>	<u>AVR_CAN_SetRxMask</u>
<u>AVR_CAN_Disable</u>	<u>AVR_CAN_Init</u>	<u>AVR_CAN_StartMessage</u>
<u>AVR_CAN_Enable</u>	<u>AVR_CAN_RxErrCount</u>	<u>AVR_CAN_TxErrCount</u>
<u>AVR_CAN_GetError</u>	<u>AVR_CAN_SetRxEMask</u>	

1.11 Banking Port

```
Import SysTick, BankPort;
```

Define

```

ProcClock  = 8000000;    {Hertz}
SysTick    = 10;         {msec}
StackSize  = $0030, iData;
FrameSize  = $0030, iData;
BankPort   = 3;          {Bank Ports}

```

```
{ $BDATA 0 }
```

var

```

bb0 : byte;
ww0 : word;

```

```
...
```

```
UserDevice BankDevIni;      // Bank Device init
```

begin

```

(* is called at System Init Time *)
(* initialize Device Hardware *)

```

end

```
UserDevice BankDevInp (bank : byte; adr : word) : byte;
```

begin

```
...
return(xxx);
end;
```

```
UserDevice BankDevOut (bank : byte; adr : word; arg : byte);
begin
```

```
...
end;
```

Functions and Procedures

[BankDevPtr](#)

[GetBankNum](#)

1.12 BeepPort

```
Import SysTick, BeepPort, ...;
```

Define

```
ProcClock = 8000000;    {Hertz}
SysTick   = 10;         {msec}
StackSize = $0020, iData;
FrameSize = $0030, iData;
BeepPort  = PortB, 0;
```

Functions and Procedures

[BeepChirpH](#)

[BeepClick](#)

[BeepOutErr](#)

[BeepOutLH](#)

[BeepStepHL](#)

[BeepChirpL](#)

[BeepOut](#)

[BeepOutHL](#)

[BeepSiren](#)

[BeepStepLH](#)

1.13 DA Converter (XMega)

```
Import SysTick, DAC_A, DAC_B;
```

Define

```
// The XMegas don't provide any Oscillator fuses.
// So the application must setup the desired values
// possible OSC types: extXTAL, extClock, ext32kHz, int32Khz, int2MHz, int32MHz
```

```
//>> CPU=32MHz, PeripherX4=32MHz, PeripherX2=32MHz
OSCType   = int32MHz, PLLmul=4, prescB=1, prescC=1;
SysTick   = 10;         {msec}
StackSize = $0032, iData;
FrameSize = $0064, iData;
DAC_A     = chan01, REF100;    // DAC_A channel 0 + 1 used
DAC_B     = chan0, REFextB;    // DAC_B channel 0 used
```

Functions and Procedures

[SetDacA](#)

[SetDacB](#)

1.14 DCF-77 Encoder

```
Import SysTick, DCFclock, ...;
```

Define

```
ProcClock = 8000000;    {Hertz}
```

```

SysTick      = 10 , Timer2;      {msec}
DCFclock     = iData;
DCFport      = PinD, 2, negative; {Port, bit number, polarity}
DCFfieldMode = reset;           // decrement, optional "reset"

```

Functions and Procedures

[DCFDayLightSave](#) [DCFfield](#) [DCFready](#) [DCFupdate](#)

1.15 DDS10 Synthesizer

```

Import SysTick, DDS10;           // and SPIDriver if necessary

```

```

Define ProcClock      = 16000000; {16Mhz clock }
      DDS10Timer        = Timer1;   {use a 16bit Timer}
      DDS10port         = PortA;
      // DDS10port      = SPI;
      // DDS10port      = UserPort ; {use DDS10IOS}
      DDS10Tables       = 1;        {use 2 lookup tables}

```

XMega

```

DDS10Timer = Timer_C1;           {Timer_C0,_C1,_D0,_D1,_E0,_E1,_F0,_F1}
DDS10port = SPI_C, SPImode3, SPImsb, PortF, 4; {Mode 0..3, MSB/LSB, SS-Port, SS-Pin}

```

type

```

tdsMode = (dsSine, dsTriaLeft, dsTriaSym, dsTriaRight, dsSquare);

```

Functions and Procedures

[DDS10buildTab](#) [DDS10setTab](#) [DDS10setFrequ](#) [DDS10start](#) [DDS10stop](#)

1.16 FAT16 File System

```

Import SysTick, FAT16, ...;

```

```

From System Import longword, ...;

```

Define

```

ProcClock      = 16000000;      {Hertz}
SysTick        = 10;            {msec}
StackSize      = $0040, iData;
FrameSize      = $0100, iData;
FAT16          = MMC_SPI, iData;
F16_MMCspeed   = standard;      // standard, slow, fast
F16_FileHandles = 4;
F16_DirLevels  = 2;

```

See also [ExtractFileExt](#), [ExtractFileName](#), [ExtractFilePath](#)

Functions and Procedures

F16_BlockRandomWrite	F16_FileAppend	F16_FileReset	F16_GetUsedHandles
F16_BlockRead	F16_FileAssign	F16_FileSeek	F16_PathExist
F16_BlockWrite	F16_FileClose	F16_FileSetAttr	F16_PathExpand
F16_ChangeDir	F16_FileCreate	F16_FileSetDate	F16_RandomWrite
F16_CheckDisk	F16_FileCopy	F16_FileSize	F16_ReadSector
F16_CheckHandle	F16_FileDelete	F16_FileSizeH	F16_RemoveDir
F16_CreateDir	F16_FileExist	F16_FindFirst	F16_StrToDate
F16_DateToStr	F16_FileGetAttr	F16_FindNext	F16_StrToTime
F16_DirGetDate	F16_FileGetDate	F16_GetCurDir	F16_TimeToStr
F16_DiskInit	F16_FilePos	F16_GetDiskError	F16_WriteSector
F16_DiskReset	F16_FileRename	F16_GetDiskFree	
F16_EndOfFile	F16_FileRewrite	F16_GetDiskSize	

1.17 8bit File System

Import SysTick, FileSystem, ...;

From System Import longWord, ...;

Define

```
FileBuffer    = iData;
FileHandles   = 2, iData;
Disk_A        = 1024, readOnly;    {kBytes}
SecTrk_A      = 2;                 {2sect/track = 512bytes/track}
TRKOFFS_A     = 1;                 {1 reserved system track}
Disk_B        = 2048;              {kBytes}
SecTrk_B      = 32;                {32sect/track = 4096bytes/track}
```

Functions and Procedures

ChangeDir	FileClose	FileOpen	FileSize
DiskFree	FileCreate	FilePos	FileSysReset
DiskFormat	FileDelete	FileRead	FileWrite
DiskReset	FileExists	FileRename	GetCurDir
DiskSelect	FileFirst	FileReset	GetCurDisk
EndOfFile	FileGetAttr	FileRewrite	
FileAppend	FileHandleCheck	FileSeek	
FileChangeDir	FileNext	FileSetAttr	

1.18 Flash Writer

Import SysTick, FlashWrite, ...;

Functions and Procedures

FlashClearPage	FlashErasePage	FlashReadFuses	FlashWritePage
FlashCopyF2R	FlashInitPage	FlashReadPage	
FlashCopyR2F	FlashProgPage	FlashWriteFuses	

1.19 Frequency Counter/Timer

Import SysTick, FreqCount [, FreqCount2];

Define

```
ProcClock    = 8000000;    {8Mhz clock }
SysTick      = 2.0;        {2msec Tick}
FreqTimer    = Timer1;     (used 16bit Timer}
FreqTimer2   = Timer3;     (restricted to Timer3}
```

Type

```
tFreqCountMode = ( FreqBaseNone, TFreqBase100Hz, TFreqBase1kHz, TFreqBase10kHz,
TFreqBase100kHz, TFreqBase1MHz, TTimeBase100s, TTimeBase10s,
TTimeBase1s, TTimeBase100ms, TPulseBase100s, TPulseBase10s,
TPulseBase1s, TPulseBase100ms);
```

Functions and Procedures

<u>FreqCountRestart</u>	<u>SetFreqCountMode</u>	<u>GetTimeCounterP</u>
<u>FreqCountRestart2</u>	<u>GetFreqCountMode2</u>	<u>GetTimeCounterP2</u>
<u>GetFreqCounter</u>	<u>GetFreqCountOvrFlow</u>	<u>SetFreqCountMode</u>
<u>GetFreqCounterL</u>	<u>GetFreqCountOvrFlow2</u>	<u>SetFreqCountMode2</u>
<u>GetFreqCounter2</u>	<u>GetTimeCounter</u>	
<u>GetFreqCounter2L</u>	<u>GetTimeCounter2</u>	

1.20 Increment Counter

Import SysTick, IncrPort, ..;

Define

```
ProcClock    = 8000000;    {Hertz}
SysTick      = 10;        {msec}
StackSize    = $0010, iData;
FrameSize    = $0010, iData;
IncrCounter  = 16, 2;     {16 bit integer, 2 phase}
IncrPort     = PinD, $C0; {PinD, Portpin 6 + 7}
```

Functions and Procedures

<u>ClearIncrementVal</u>	<u>GetIncrementRel</u>	<u>GetIncrementVal</u>	<u>SetIncrementVal</u>
--	--	--	--

1.21 Increment Counter 4chan

Import SysTick, IncrPort4, ..;

Define

```
ProcClock    = 16000000;   {Hertz}
StackSize    = $0020, iData;
FrameSize    = $0040, iData;
IncrPort4    = PinA, 2, 32; // pin-reg used, channels, 16 or 32bit integer
IncrScan4    = Timer3, 10; // timer used, scan rate 10kHz (1..100)
```

Functions and Procedures

[ClearIncrAll4](#)
[ClearIncrVal4](#)

[GetIncrRel4](#)
[GetIncrVal4](#)

[IncrCount4start](#)
[IncrCount4stop](#)

[SetIncrVal4](#)

1.22 I2C Disp7

Import I2Cport, I2C_Disp7;

or

Import TWImaster, I2C_Disp7;

or

Import TWInet, I2C_Disp7; // use Master mode

I2Cport:

Define

```
ProcClock    = 8000000;      {8Mhz clock }
I2Cport      = PortC;        {port used}
I2Cdat       = 7;            {bit7-PortC}
I2Cclk       = 6, 4;         {bit6-PortC, optional delay 4}
I2C_Disp7    = I2C_Soft, iData; {use Software I2Cport, buffer loc}
// use 2 Displays
I2C_7sDig1   = 8;            {first display 8digits}
I2C_7sDig2   = 6;            {second display 8digits}
I2C_7Mode    = wrap;
```

TWImaster:

Define

```
ProcClock    = 8000000;      {8Mhz clock }
TWIpresc     = TWI_BR100;    {100kBit/sec alt. TWI_BR400}
I2C_Disp7    = I2C_TWI, xData; {use TWIport, buffer location}
// use 4 Displays
I2C_7sDig1   = 4;            {first display 4digits}
I2C_7sDig2   = 4;            {second display 4digits}
I2C_7sDig3   = 4;            {third display 4digits}
I2C_7sDig4   = 4;            {fourth display 4digits}
I2C_7Mode    = shiftLeft;
```

TWInetMaster:

Define

```
ProcClock    = 8000000;      {8Mhz clock }
TWInode      = 05;           {default address in slave mode}
TWIpresc     = TWI_BR400;    {400kBit/sec alt. TWI_BR100}
TWIframe     = 4, iData;     {buffer/packet size}
TWIframeBC   = 6;            {option broadcast buffer/packet size}
TWInetMode   = Master;
I2C_Disp7    = I2C_TWI, iData; {use TWIport, buffer location}
// use 1 Display
I2C_7sDig1   = 4;            {4digits}
I2C_7Mode    = wrap;
```

Type

```
TI2C_DISP7 = (Disp7_1, Disp7_2, Disp7_3, Disp7_4);
TI2C_Ctrl7 = (Disp7_On, Disp7_Off, Disp7_Test);
```

Var

```
I2C_7Buff1, I2C_7Buff2, I2C_7Buff3, I2C_7Buff4;
```

Functions and Procedures

[I2C_Disp7Clear](#)
[I2C_Disp7CIEOL](#)
[I2C_Disp7Ctrl](#)

[I2C_Disp7DigitBlink](#)
[I2C_Disp7Dim](#)
[I2C_Disp7Get](#)

[I2C_Disp7Init](#)
[I2C_Disp7Out](#)
[I2C_Disp7Pos](#)

[I2C_Disp7Refresh](#)
[I2C_Disp7Set](#)

1.23 I2C Port

Import SysTick, I2Cport;

Define

```
ProcClock = 4000000;    {4Mhz clock }
SysTick   = 10;         {10msec Tick}
I2Cport    = PortB;     {use port B}
I2Cclk     = 0;         {clock-pin = port B bit 0}
//I2Cclk   = 0,NOPs
I2Cdat     = 3;         {data-pin = port B bit 3}
```

Functions and Procedures

[I2Cinp](#) [I2Cout](#) [I2Cstat](#)

1.24 I2Cexpand

Import I2Cport, I2Cexpand;

or

Import TWImaster, I2Cexpand;

or

Import TWInet, I2Cexpand; // use Master mode

or **XMega**

Import TWI_C, I2Cexpand; // use TWI_C, TWI_D, TWI_E or TWI_F

I2Cport:

Define

```
ProcClock = 8000000;    {8Mhz clock }
I2Cport    = PortC;     {port used}
I2Cdat     = 7;         {bit7-PortC}
I2Cclk     = 6, 4;      {bit6-PortC, optional delay 4}
I2Cexpand  = I2C_Soft, $38; {use Software I2Cport, 9554A}
I2CexpPorts = Port0, Port4; {use Port0 and Port4}
```

TWImaster:

Define

```
ProcClock = 8000000;    {8Mhz clock }
TWIpresc  = TWI_BR100;  {100kBit/sec alt. TWI_BR400}
I2Cexpand = I2C_TWI, $38; {use TWIport, 9554A}
I2CexpPorts = Port1, Port2; {use Port1 and Port2}
```

TWInetMaster:

Define

```
ProcClock = 8000000;    {8Mhz clock }
TWInode   = 05;         {default address in slave mode}
TWIpresc  = TWI_BR400;  {400kBit/sec alt. TWI_BR100}
TWIframe  = 4, iData;    {buffer/packet size}
TWIframeBC = 6;         {option broadcast buffer/packet size}
```

```

TWInetMode = Master;
I2Cexpand  = I2C_TWI, $20;      {use TWIport, 9554}
I2CexpPorts = Port7;           {use Port7}

```

XMega

Define

```

OSCtype      = int32MHz, PLLmul=4, prescB=1, prescC=1;
TWIpresc     = TWI_BR100;      {100kBit/sec alt. TWI_BR400}
I2Cexpand    = TWI_C, $38;     {use TWIportC, 9554A}
I2CexpPorts  = Port1, Port2;   {use Port1 and Port2}

```

Var

```

TWI_DevLock : DEVICELOCK;
TWI_DevLockTN : DEVICELOCK; // XMega TN = C, D, E or F

```

Functions and Procedures

I2CexpStat

1.25 I2Cexpand_5

```

Import I2Cport, I2Cexpand_5;
or
Import TWImaster, I2Cexpand_5;
or
Import TWInet, I2Cexpand_5;      // use Master mode
or XMega
Import TWI_C, I2Cexpand_5;      // use TWI_C, TWI_D, TWI_E or TWI_F

```

I2Cport:

Define

```

ProcClock    = 8000000;          {8Mhz clock }
I2Cport      = PortC;            {port used}
I2Cdat       = 7;                {bit7-PortC}
I2Cclk       = 6, 4;             {bit6-PortC, optional delay 4}
I2Cexpand_5  = I2C_Soft, $38;    {use Software I2Cport}
I2CexpPorts_5= Port0, Port4;     {use Port00..04. and Port05..09 = 2x PCA9698}

```

TWImaster:

Define

```

ProcClock    = 8000000;          {8Mhz clock }
TWIpresc     = TWI_BR100;        {100kBit/sec alt. TWI_BR400}
I2Cexpand_5  = I2C_TWI, $18;     {use TWIport}
I2CexpPorts_5= Port00, Port05;   {use Port00..04. and Port05..09 = 2x PCA9698}

```

TWInetMaster:

Define

```

ProcClock    = 8000000;          {8Mhz clock }
TWInode      = 05;               {default address in slave mode}
TWIpresc     = TWI_BR400;        {400kBit/sec alt. TWI_BR100}
TWIframe     = 4, iData;         {buffer/packet size}
TWIframeBC   = 6;                {option broadcast buffer/packet size}
TWInetMode   = Master;
I2Cexpand_5  = I2C_TWI, $20;     {use TWIport}
I2CexpPorts_5= Port00, Port05;   {use Port00..04. and Port05..09 = 2x PCA9698}

```

XMega**Define**

```

OSCtype      = int32MHz, PLLmul=4, prescB=1, prescC=1;
TWIpresc     = TWI_BR100;          {100kBit/sec alt. TWI_BR400}
I2Cexpand_5  = TWI_C, $18;         {use TWIport C}
I2CexpPorts_5= Port00, Port05;     {use Port00..04. and Port05..09 = 2x PCA9698}

```

Var

```

TWI_DevLock : DEVICELOCK;
TWI_DevLockTN : DEVICELOCK; // XMega TN = C, D, E or F for TWI_C...TWI_F

```

Functions and Procedures

[I2CexpStat_5](#)

1.26 IOexpand

Import IOexpand;

Define

```

ProcClock     = 8000000;           {8Mhz clock }
IOexpand      = PortD, 2, iData;    {Port, Port-startbit, memory loc}
IOexplnp      = 4;                 {4x8 = 32 bit input}
IOexpoutp     = 4;                 {4x8 = 32 bit output}

```

Var

```

IOexplnpArr : array [0.. IOexplnp-1] of byte;
IOexplnp0 : byte;
IOexplnp1 : byte;
IOexplnp2 : byte;
...
IOexpoutpArr : array[0.. IOexpoutp-1] of byte;
IOexpoutp0 : byte;
IOexpoutp1 : byte;
IOexpoutp2 : byte;
...

```

Functions and Procedures

[IOexpUpdate](#)

1.27 KeyBoard

Import SysTick, MatrixPort, ... ;

Option

From MatrixPort **import** MatrixTimer;

Define

```

MatrixRow     = PortA, 4;          {use PortA, start with bit4}
MatrixCol     = PinA, 0;           {use PinA, start with bit0}
MatrixType    = 3, 4              {3 Rows at PortA, 4 Columns at PinA}
Debounce      = 4;                {optional debounce in systick counts}

```

Option

MatrixPipe = PipeLen, FirstRepeat, RepeatRate;

Type

Keys = (Key1, Key2, ..., KeyN);
 KeySet = BitSet **of** Keys

Const

lastMatrixKey : Keys = KeyN;

Option

Var

MatrixKeyPipe: Pipe [PipeLen] **of** Keys;

Functions and Procedures

ClearKeyBoard	KeyBoardEnable	KeyStat	ReadKeyBoard
GetKey	KeyBoardRepeat	KeyStatRaised	
GetKeyRaised	KeyRaised	ReadKey	

1.28 Keyboard8

Import SysTick, KeyPort8, ... ;

Option

From KeyPort8 **import** KeyBoardTimer;

Define

KeyB8Row = PortA, 2; {use PortA, start with bit2}
 KeyB8Col = PinD; {use PinD complete input port}
 KeyB8Type = 4; {4 Rows at PortA, 8 Columns at PinD}
 Debounce = 5; {nn = SysTicks, optional}

Option

KeyB8Pipe = PipeLen, FirstRepeat, RepeatRate;

Type

Keys = (Key1, Key2, ..., KeyN);
 KeySet = BitSet **of** Keys

Const

lastKeyBoardKey : Keys = KeyN;

Option

Var

KeyboardPipe : Pipe [PipeLen] **of** Keys;

Functions and Procedures

ClearKeyBoard8	KeyBoardEnable8	KeyStatRaised8	ReadKey8
GetKey8	KeyBoardRepeat8	KeyStat8	
GetKeyRaised8	KeyRaised8	ReadKeyBoard8	

1.29 LED 7seg Display

Import SysTick, Disp7sPort;

Define

ProcClock = 4000000; {4Mhz clock }
 SysTick = 10; {10msec Tick}

```

Disp7sPort = PortA, Mux;           {Port address, multiplexed}
//Disp7sPort = PortA, NonMux;       {Port address, not multiplexed}
//Disp7sPort = PortA, Mux, startpin; {Port address, multiplexed with Startpin}
DispMode = ShiftLeft;              {shift left}
DispDigits = 4, iData;             {4-Digit Display}
DispMode = ShiftLeft, noBlank;     {loptional}

```

Functions and Procedures

[DispBlink](#) [DispCLEOL](#) [DispOut](#) [Disp7Test](#)
[DispClear](#) [DispDigBlink](#) [DispPos](#)

1.30 LED 14seg Display

Import SysTick, Disp14sPort;

Define

```

ProcClock = 8000000;           {8Mhz clock }
SysTick = 5;                   {5msec}
Disp14sPort = PortB, 0;        {Port, start Portbit}
Disp14Mode = ShiftRight, Blank;
Disp14Digits = 6, iData;

```

Var Disp14Buff : array [0.. Disp14Digits-1] of word;

Functions and Procedures

[Disp14Blink](#) [Disp14ClrEOL](#) [Disp14Out](#) [Disp14Test](#)
[Disp14Clear](#) [Disp14DigBlink](#) [Disp14Pos](#)

1.31 LCD Bargraph

Define

```

ProcClock = 8000000; {8Mhz clock }
...
...
LCDBargraph1 = LCDmultiPort;
LCDBargraph2 = LCDmultiPort;
LCDBargraph3 = LCDmultiPort;
LCDBargraph4 = LCDmultiPort;
// LCDBargraph4 = LCDPort;

```

Functions and Procedures

[LCDbarInit_M](#) [LCDbarOut1](#) [LCDbarOut3](#) [LCDbarSet1](#) [LCDbarSet3](#)
[LCDbarInit_P](#) [LCDbarOut2](#) [LCDbarOut4](#) [LCDbarSet2](#) [LCDbarSet4](#)

1.32 LCD Display

Import SysTick, LCDport;

Define

```

ProcClock = 4000000;   {4Mhz clock }
SysTick = 10;          {10msec Tick}

```

```

LCDtype      = 44780;      {66712,0070, 0073}
LCDport      = PortA;      {Port Address}
LCDRows      = 2;          {2-Line Display}
LCDColumns   = 16;         {16-Column Display}

```

Option 1

```

LCDport = PortC, 2, PortA, 3;          // controlport, bit, Dataport, bit. Splitted ports

```

Option 2

```

LCDtype = 0073; {KS0073}
LCDport = SPI_Soft, PortB, 1, 2, 3, 0; // PortX, SCK, MOSI, MISO, SS

```

Option 3

```

LCDtype = 0073;          {KS0073}
LCDport = SPI;           // Hardware SPI
LCDport = SPI_C, PortB, 1; // Hardware SPI, SS_port, SS_pin X Mega

```

Functions and Procedures

LCDbarInit_P	LCDbarSet1	LCDcharsetP	LCDcursor	LCDoff	LCDupper
LCDbarOut1	LCDbarSet2	LCDclr	LCDgetXY	LCDon	LCDxy
LCDbarOut2	LCDbarSet3	LCDclrEOL	LCDhome	LCDout	
LCDbarOut3	LCDbarSet4	LCDclrLine	LCDinp	LCDsetup	
LCDbarOut4	LCDcharset	LCDctrl	LCDlower	LCDstat	

1.33 LCD Display Multi

```

Import I2Cport, LCDmultiPort;

```

or

```

Import TWImaster, LCDmultiPort;

```

or

```

Import TWInet, LCDmultiPort;          // use Master mode

```

or **X Mega**

```

Import TWI_C, TWI_E, LCDmultiPort; // TWI_C, TWI_D, TWI_E, TWI_F

```

I2Cport:

Define

```

ProcClock      = 8000000; {8Mhz clock }
I2Cport        = PortC;   {port used}
I2Cdat         = 7;       {bit7-PortC}
I2Cclk         = 6, 4;    {bit6-PortC, optional delay 4}
LCDmultiPort   = I2C_Soft; {use Software I2Cport}
LCDTYPE_M      = 66712;   {LCD controller type}
LCDrows_M      = 2;       {2 rows}
LCDcolumns_M   = 20;      {20 chars per line}

```

TWImaster:

Define

```

ProcClock      = 8000000; {8Mhz clock }
TWIpresc       = TWI_BR100; {100kBit/sec alt. TWI_BR400}
LCDmultiPort   = I2C_TWI;  {use TWIport}
LCDTYPE_M      = 44780;    {LCD controller type}
LCDrows_M      = 4;        {4 rows}
LCDcolumns_M   = 16;       {16 chars per line}

```

TWInetMaster:

Define

```

ProcClock      = 8000000;    {8Mhz clock }
TWInode        = 05;        {default address in slave mode}
TWIpresc       = TWI_BR400;  {400kBit/sec alt. TWI_BR100}
TWIframe       = 4, iData;   {buffer/packet size}
TWIframeBC     = 6;         {option broadcast buffer/packet size}
TWInetMode     = Master;
LCDmultiPort   = I2C_TWI;    {use TWIport}
LCDTYPE_M      = 0070;      {LCD controller type}
LCDrows_M      = 1;         {1 row}
LCDcolumns_M   = 12;        {12 chars per line}

```

XMega**Define**

```

OSctype        = int32MHz, PLLmul=4, prescB=1, prescC=1;
TWIprescC      = TWI_BR100; {100kBit/sec alt. TWI_BR400}
LCDmultiPort   = I2C_TWI;   {use TWIport}
LCDTYPE_M      = 44780;     {LCD controller type}
LCDrows_M      = 4;         {4 rows}
LCDcolumns_M   = 16;        {16 chars per line}

```

Type

```

TLCD_num = (LCD_m1,LCD_m2,LCD_m3,LCD_m4,LCD_m5,LCD_m6,LCD_m7,LCD_m8);

```

Var

```

TWI_DevLock : DEVICELOCK;
TWI_DevLockTN : DEVICELOCK; // XMega, TN = C, D, E, F

```

Functions and Procedures

<u>LCDbarInit_M</u>	<u>LCDbarSet2</u>	<u>LCDclrLine_M</u>	<u>LCDhome_M</u>	<u>LCDsetPort_M</u>
<u>LCDbarOut1</u>	<u>LCDbarSet3</u>	<u>LCDclr_M</u>	<u>LCDinp_M</u>	<u>LCDsetup_M</u>
<u>LCDbarOut2</u>	<u>LCDbarSet4</u>	<u>LCDctrl_M</u>	<u>LCDoff_M</u>	<u>LCDstat_M</u>
<u>LCDbarOut3</u>	<u>LCDcharSet_M</u>	<u>LCDcursor_M</u>	<u>LCDon_M</u>	<u>LCDxy_M</u>
<u>LCDbarOut4</u>	<u>LCDcharSet_MP</u>	<u>LCDgetPort_M</u>	<u>LCDout_M</u>	
<u>LCDbarSet1</u>	<u>LCDclrEOL_M</u>	<u>LCDgetXY_M</u>	<u>LCDportInp_M</u>	

1.34 LCD Edit

Uses FEdit;

fixed Constants:

```

EdEditLength   : Byte = 40
EdLabelLength  : Byte = 20
EdTimeDelim    : Char = ':'
EdDateDelim    : Char = '.'
EdIPDelim      : Char = '.'

```

changeable Constants:

```

EdPreClearLine : Boolean = True
EdBooleanTrue  : String= 'AN'
EdBooleanFalse : String= 'AUS'
KBRepeatTrigger : Byte = 100;      // in SysTicks
KBRepeatDelay   : Word = 100;     // in SysTicks

```

Types:

```

Type   tEdArrayLocation = (edRam, edEEProm, edFlash);
Type   tEdLCDType = (edLCDnone, EdLCDStandard, EdLCDMulti);
Type   tEdKeys = (EdKeyNone, EdKeyUp, EdKeyDown, EdKeyLeft, EdKeyRight, EdKeyExit);
Type   TEdActEditor = (edNoneEd, edTimeEd, edDateEd, edByteEd, edBooleanEd, edStringEd,
                        edIntegerEd, edIPAddressEd, edLongIntEd, edWordEd, edLongWordEd, edListEd);
Type   tEdErrorEvent = (edLeftLim, edRightLim, edUPLim, edDownLim, edNoKeyHandler,
                        edNoLCDDefined);
Type   tEdKeyboardHandler =
        Function (ActiveEditor : tEdActEditor; LookKey : tEdKeys) : tEdKeys;
Type   tEdErrorEventHandler =
        Procedure (ActiveEditor : tEdActEditor; ErrorEvent : tEdErrorEvent);
Type   tEdIPAddress =
        Record
            IPOct1,
            IPOct2,
            IPOct3,
            IPOct4 : Byte;
        end;

```

Standard Parametes of the Editores:

EdValue	editable parameter passed to the editor
LeadLabel	optional leading label of the edit field
Postlabel	optional trailing label after the edit field e.g. <i>LeadLabel -> Mass: 12.54 kg <- PostLabel</i> [^] <i>EdValue</i>
X, Y	Position line/column on the LCD display
BlinkCursor	Blinking block cursor on the LCD display at the edit position
VMin,VMax	Minimum and maximum possible edit value
Repeater	AutoRepeater on/off in the string and list editor
Decimal	Visible decimal count of the edit value

Keyboard Handler:

```

Function MyKeyHandler (ActiveEditor : tEdActEditor; ReturnKey : tEDKeys) : tEDKeys;
begin
    case ActiveEditor of
        ...

```

Error Handler:

```

Procedure MyErrorEventHandler (ActiveEditor : tEdActEditor; ErrorCode: tEdErrorEvent);
begin
    case ActiveEditor of
        ...

```

Functions and Procedures

<u>EdBoolean</u>	<u>EdInteger</u>	<u>EdLongInt</u>	<u>EdTime</u>
<u>EdByte</u>	<u>EdIPAddress</u>	<u>EdLongWord</u>	<u>EdWord</u>
<u>EdDate</u>	<u>EdList</u>	<u>EdString</u>	

1.35 LCD Graphic

Import SysTick, LCDGraphic, ...;

// only if strings are used

From LCDGraphic **Import** CharSet; {block CharSet, pixels}

Define

```

LCDGraphic      = 240, 128, 8; {x-pix, y-pix, access width}
GViewports      = 4, iData; {logical ViewPorts, scalings}
LCDgraphMode    = linear, iData; {optional, linear is default}
or
LCDgraphMode    = column, iData; {column oriented controller}
or
LCDgraphMode    = readonly, iData; {linear, readonly controller}
DefCharSet      = 'Graphchars.pchr'; {FileName, stored into Flash}
or
DefCharSet      = RAM; {charset is stored into RAM}
TGraphStr       = 20; {Graphic Text String Length, max 24}

```

Type

```

TGraphString    = String[n]; {max. 24 Zeichen}
TWriteMode      = (wmClrPix, wmSetPix, wmXorPix);
TtxtAlHor       = (alHorLeft, alHorCenter, alHorRight);
TtxtAlVert      = (alVertBottom, alVertCenter, alVertTop);
TtxtRotate      = (TxtRot0, TxtRot90, TxtRot180, TxtRot270);
TTextBkGnd      = (bkNormal, bkTransp, bkInvers);

```

Functions and Procedures

<u>gClearPixel</u>	<u>gDrawString</u>	<u>gGetTextMode</u>	<u>gSetCharSetRAM</u>
<u>gClearView</u>	<u>gDrawStringRel</u>	<u>gMoveTo</u>	<u>gSetLineColor</u>
<u>gClrScr</u>	<u>gFillCircle</u>	<u>gMoveToRel</u>	<u>gSetLineMode</u>
<u>gDispRefresh</u>	<u>gFillRect</u>	<u>gOpenView</u>	<u>gSetPixel</u>
<u>gDrawBitMap</u>	<u>gFrameView</u>	<u>gPntToScale</u>	<u>gSetTextBkGnd</u>
<u>gDrawBitMapN</u>	<u>gGetCurView</u>	<u>gRestoreView</u>	<u>gSetTextColor</u>
<u>gDrawCircle</u>	<u>gGetLineColor</u>	<u>gSaveView</u>	<u>gSetTextJustify</u>
<u>gDrawLine</u>	<u>gGetLineMode</u>	<u>gScaleToPnt</u>	<u>gSetTextMode</u>
<u>gDrawLineTo</u>	<u>gGetTextBkGnd</u>	<u>gScaleView</u>	<u>gSwitchView</u>
<u>gDrawLineToRel</u>	<u>gGetTextColor</u>	<u>gSetBitMapRAM</u>	<u>gXorPixel</u>
<u>gDrawRect</u>	<u>gGetTextJustify</u>	<u>gSetCharSet</u>	

1.36 LED DOT Display

T.B.D.

Functions and Procedures

<u>LEDdotBlink</u>	<u>LEDdotClr</u>	<u>LEDdotDim</u>	<u>LEDdotOn</u>
<u>LEDdotBlinkDigit</u>	<u>LEDdotClrEOL</u>	<u>LEDdotGetXY</u>	<u>LEDdotOff</u>
<u>LEDdotCharset</u>	<u>LEDdotClrLine</u>	<u>LEDdotHome</u>	<u>LEDdotOut</u>
<u>LEDdotCharsetP</u>	<u>LEDdotCursor</u>	<u>LEDdotInit</u>	<u>LEDdotXY</u>

1.37 LPT Port

Import SysTick, LPTport, ..;

Define

```
ProcClock = 8000000;      {Hertz}
SysTick   = 10;           {msec}
StackSize = $0010, iData;
FrameSize = $0010, iData;
LPTport   = PortA, PortB; // DataPort, ControlPort
```

or

Import SysTick, TWImaster, LPTport, ..; // TWInetMaster is also possible

Define

```
ProcClock = 8000000;      {Hertz}
SysTick   = 10;           {msec}
StackSize = $0010, iData;
FrameSize = $0010, iData;
TWIpresc  = TWI_BR400;
LPTport   = TWI_I2C, $24; // $24 = TWIaddr
```

or

Import SysTick, I2Cport, LPTport, ..;

Define

```
ProcClock = 8000000;      {Hertz}
SysTick   = 10;           {msec}
StackSize = $0010, iData;
FrameSize = $0010, iData;
I2Cport   = PortA;
I2Cclk    = 0;             // bit0, porta
I2Cdat    = 1;             // bit1, porta
LPTport   = Soft_I2C, $24; // $24 = I2Caddr
```

or **XMega**

Import SysTick, TWI_C, LPTport, ..; // TWI_D, TWI_E, TWI_F are also possible

Define

```
OSCtype   = int32MHz, PLLmul=4, prescB=1, prescC=1;
SysTick   = 10;           {msec}
StackSize = $0020, iData;
FrameSize = $0040, iData;
TWIprescC = TWI_BR400;
LPTport   = TWI_C, $24;   // $24 = TWIaddr
```

Type

```
tLPTlines = (lpStrobe, lpError, lpInit, lpSelect, lpACK, lpBusy, lpSelected, lpPaper);
tLPTlineSet = BitSet of tLPTlines;
```

Functions and Procedures

[LPTctrl](#) [LPTdir](#) [LPTinit](#) [LPTinp](#) [LPTout](#) [LPTreset](#) [LPTstat](#)

1.38 MIRF24 Port

Import SysTick..., MIRF24port, ..;

Define

```

ProcClock    = 16000000; {Hertz}
SysTick      = 10; {msec}
StackSize    = $0040, iData;
FrameSize    = $0040, iData;

MIRF24port    = SPI_Soft, PortA, 2, 3, 4, 1, 0, 5;
               // SCK, MOSI, MISO, SS, CE, IRQ
// MIRF24port = SPI, PortA, 0, 1, 2;           // standard SPI port
               // SS, CE, IRQ
//XMEga: SPI_C, SPI_D, SPI_E, SPI_F

// MIRF24port = MSPI_2, PortA, 0, 1, 2;         // MSPI_0..MSPI_3
               // SS, CE, IRQ ...
//XMEga: MSPI_C0, MSPI_C1, MSPI_D0, ...

```

uses uMIRF24;

Type

```

tMRFchan      = (mrfChan1, mrfChan2, mrfChan3, mrfChan4, mrfChan5,
                 mrfChan6, mrfChan7, mrfChan8, mrfChan9, mrfChan10,
                 mrfChan11, mrfChan12, mrfChan13, mrfChan14);
tMRFpwr       = (mrfdBm0, mrfdBm6, mrfdBm12, mrfdBm18);
enMRFstat     = (mrfTX_full, mrfRX_pn0, mrfRX_pn1, mrfRX_pn2,
                 mrfMAX_RT, mrfTX_DS, mrfRX_DR);
tMRFstat      = Bitset of enMRFstat;
tMRFpkt       = (mrfPKTnone, mrfPKTdata, mrfPKTbcast);
tMRFrfSpeed   = (mrfRF250, mrfRF1000, mrfRF2000);

```

Functions and Procedures

MRFgetLostPkts	MRFgetState	MRFsetFreq	MRFsetRetryMax
MRFgetRetryCnt	MRFinit	MRFsetLocalAdr	MRFsetRetryTimeOut
MRFgetRxPower	MRFrxPacket	MRFsetPower	MRFsetRFspeed
MRFgetRxType	MRFsetChan	MRFsetPWRdown	MRFtxPacket

1.39 ModBus ASCII

Import SysTick, SerPort, ModBus; //SerPort, SerPort2, SerPort3, SerPort4 are supported

From System **Import** Processes; //driver is implemented in a process

Define

```

//example for MEGA128
ProcClock    = 16000000;           //Hertz
SysTick      = 10;                 //msec
StackSize    = $0020, iData;
FrameSize    = $0050, iData;
Scheduler    = iData;
SerPort      = 19200, Databit7, parEven, Stop1; //ASCII default
SerPortDTR   = PinB, 7, Positive;  //RF usually has busy signal

```

```

SerCtrl      = PortD, 2, Positive;    //control line if RS485 used
RxBuffer     = 255, iData;           //recommended, but may be lower
TxBuffer     = 100, iData;           //recommended, but may be lower
ModBus       = SerPort, 40;          //use port 1/2, capacity in words
ModBusMode   = ASCII;                //what modbus mode to use

```

Uses `ModBusServASCII;` *//modbus logic is in this unit*

Type

```

mb_InpB      = byte;                { !! always as couples }
mb_RdWrB     = byte;                { !! always as couples }
mb_InpW      = word;
mb_RdWrW     = word;                {prefix mb_Inp is used to identify}
mb_InpI      = Integer;             {that tag is read only, usually used}
mb_RdWrI     = Integer;             {for mapping input from some sensor,}
mb_InpW32    = longword;            {and prefix mb_RdWr is for read/write,}
mb_RdWrW32   = longword;            {usually used for physical outputs.}
mb_InpI32    = longInt;
mb_RdWrI32   = longInt;             {sufixes B, W, I, W32, I32 and F are}
mb_InpF      = Float;               {used to identify byte, word, int,}
mb_RdWrF     = Float;               {longword, longint, and float types}

```

Functions and Procedures

<u>mb_GetModBusDevID</u>	<u>mb_SetAfterRegisterRead</u>	<u>mb_SetBeforeRegisterRead</u>
<u>mb_GetModBusTimeout</u>	<u>mb_SetAfterRegisterWrite</u>	<u>mb_SetBeforeRegisterWrite</u>
<u>mb_SetAfterCoilRead</u>	<u>mb_SetBeforeCoilRead</u>	<u>mb_SetModBusDevID</u>
<u>mb_SetAfterCoilWrite</u>	<u>mb_SetBeforeCoilWrite</u>	<u>mb_SetModBusTimeout</u>

1.40 ModBus RTU

Import `SysTick, SerPort, ModBus;` *//SerPort, SerPort2, SerPort3, SerPort4 are supported*

From System Import `Processes;` *//driver is implemented in a process*

Define *//example for MEGA128*

```

ProcClock    = 16000000;             //Hertz
SysTick      = 10;                   //msec
StackSize    = $0020, iData;
FrameSize    = $0050, iData;
Scheduler    = iData;
SerPort      = 19200, Databit7, parEven, Stop1; //ASCII default
SerPortDTR   = PinB, 7, Positive;    //RF usually has busy signal
SerCtrl      = PortD, 2, Positive;    //control line if RS485 used
RxBuffer     = 255, iData;           //recommended, but may be lower
TxBuffer     = 100, iData;           //recommended, but may be lower
ModBus       = SerPort, 40;          //use port 1/2, capacity in words
ModBusMode   = RTU, Timer3;          //MODBUS mode, timer 1..3

```

Uses `ModBusServRTU;` *//MODBUS logic is in this unit*

Type

```

mb_InpB      = byte;                { !! always as couples }
mb_RdWrB     = byte;                { !! always as couples }
mb_InpW      = word;
mb_RdWrW     = word;                {prefix mb_Inp is used to identify}

```

mb_InpI = Integer; {that tag is read only, usually used}
mb_RdWrI = Integer; {for mapping input from some sensor,}
mb_InpW32 = longword; {and prefix *mb_RdWr* is for read/write,}
mb_RdWrW32 = longword; {usually used for physical outputs.}
mb_InpI32 = longInt;
mb_RdWrI32 = longInt; {sufixes B, W, I, W32, I32 and F are}
mb_InpF = Float; {used to identify byte, word, int,}
mb_RdWrF = Float; {longword, longint, and float types}

Functions and Procedures

<u>mb_GetModBusDevID</u>	<u>mb_SetAfterRegisterRead</u>	<u>mb_SetBeforeRegisterWrite</u>
<u>mb_GetModBusExceptionStatus</u>	<u>mb_SetAfterRegisterWrite</u>	<u>mb_SetModBusDevID</u>
<u>mb_GetModBusTimeout</u>	<u>mb_SetBeforeCoilRead</u>	<u>mb_SetModBusExceptionStatus</u>
<u>mb_SetAfterCoilRead</u>	<u>mb_SetBeforeCoilWrite</u>	<u>mb_SetModBusTimeout</u>
<u>mb_SetAfterCoilWrite</u>	<u>mb_SetBeforeRegisterRead</u>	

1.41 Pipes

Import SysTick, ...

From System *Import* pipes, ...

Functions and Procedures

<u>PipeFlush</u>	<u>PipeRecv</u>	<u>PipeSend</u>	<u>WaitPipe</u>
<u>PipeFull</u>	<u>PipeRecv_ND</u>	<u>PipeStat</u>	

1.42 PulseCounter

Import SysTick, PulseCount, {PulseCount2},...;

Define

ProcClock = 8000000; {Hertz}
SysTick = 10; {msec}
StackSize = \$0010, iData;
FrameSize = \$0010, iData;
PulseCount = Timer1; {or Timer3..5 if present}
//PulseCount2 = Timer1; {or Timer3..5 if present}

Functions and Procedures

<u>GetPulseCount</u>	<u>PulseCountClear</u>	<u>PulseCountStart</u>	<u>PulseCountStop</u>
<u>GetPulseCount2</u>	<u>PulseCountClear2</u>	<u>PulseCountStart2</u>	<u>PulseCountStop2</u>

1.43 PWM Port

PWMport1A, PWMport1B, PWMport1C, PWMport2A, -2B,
 PWMport3A, -3B, -3C, PWMport4A, -4B, -4C, PWMport5A, -5B, -5C

Import SysTick, PWMport1A, ..., PWMport3C, ...;

Define *ProcClock* = 8000000; {Hertz}
SysTick = 10; {msec}
StackSize = \$0030, iData;
FrameSize = \$0030, iData;

```

PWMpresc1 = $2;           {prescaler timer1}
PWMres1    = $8;           {resolution timer1}
PWMmode1   = fast, negative {optional define}
PWMpresc2  = $3;           {prescaler timer1}
PWMres2    = $9;           {resolution timer1}
PWMmode2   = slow, negative {optional define}
PWMpresc3  = $4;           {prescaler timer3}
PWMres3    = $10;          {resolution timer3}
PWMmode3   = slow, positive {optional define }
...

```

```

PWMport1A := ...
PWMport3B := ...

```

XMega:

```

PWMport_C0A, _C0B_, C0C, _C0D,   _D0A, _D0B, _D0C, _D0D,
PWMport_E0A, _E0B_, E0C, _E0D,   _F0A, _F0B, _F0C, _F0D,
PWMport_C1A, _C1B,   _D1A, _D1B,   _E1A, _E1B,   _F1A, _F1B

```

Import SysTick, PWM_C0A, PWM_C0B, PWM_C0C, PWM_C0D, PWM_C1A, PWM_C1B;

Define

```

// The XMegas don't provide any Oscillator fuses.
// So the application must setup the desired values
// possible OSC types: extXTAL, extClock, ext32kHz, int32Khz, int2MHz, int32MHz

```

```

//>> CPU=32MHz, PeripherX4=32MHz, PeripherX2=32MHz

```

```

OSCType    = int32MHz,
PLLmul=4,
prescB=1,
prescC=1;
SysTick    = 10; {msec}
StackSize  = $0064, iData;
FrameSize  = $0064, iData;

```

```

PWMpresc_C0 = $2;           // prescaler timerC0
PWMres_C0   = 8;            // pwm resolution timerC0
PWMpol_C0A  = negative;     // output polarity PWM_C0A
PWMpol_C0B  = negative;     // output polarity PWM_C0B
PWMpol_C0C  = positive;     // output polarity PWM_C0C
PWMpol_C0D  = positive;     // output polarity PWM_C0D

```

```

PWMpresc_C1 = $4;           // prescaler timerC1
PWMres_C1   = 16;           // pwm resolution timerC1
PWMpol_C1A  = positive;     // output polarity PWM_C1A
PWMpol_C1B  = negative;     // output polarity PWM_C1A

```

Functions and Procedures XMega PWMPort_C0

```

EnablePWM_C0A  SetPWM_C0A  EnablePWM_C0B  SetPWM_C0B
EnablePWM_C0C  SetPWM_C0C  EnablePWM_C0D  SetPWM_C0D

```

Functions and Procedures

XMega PWMPort_D0A, PWMPort_D0B, PWMPort_D0C, PWMPort_D0D

XMega PWMPort_E0A, PWMPort_E0B, PWMPort_E0C, PWMPort_E0D

XMega PWMPort_F0A, PWMPort_F0B, PWMPort_F0C, PWMPort_F0D

analog to PWMPort_C0A, PWMPort_C0B, PWMPort_C0C, PWMPort_C0D

Functions and Procedures ***XMega*** ***PWMPort_C1A***[EnablePWM_C1A](#)[SetPWM_C1A](#)[EnablePWM_C1B](#)[SetPWM_C1BA](#)**Functions and Procedures*****XMega*** ***PWMPort_D1A, _D1B, PWMPort_E1A, _E1B, PWMPort_F1A, _F1B***
analog to ***PWMPort_C1A, PWMPort_C1B*****1.44 RC5 Driver*****Import*** SysTick, RC5Rxport, ..;

or

Import SysTick, RC5Txport, ..;

or

Import SysTick, RC5Txport, RC5Rxport, ..;

Defines for Receiver

```
// RC5RXPORT      = PinReg, PinNum, polarity    //polarity is optional
RC5mode           = rc_7bit;                    //rc_6bit = default
```

Defines for Transmitter

```
// RC5TXPORT      = Timer1, polarity, carrier;    //Timer and carrier Frequency is optional
RC5TXPORT         = positive;
RC5TXPORT         = negative;
RC5TXPORT         = positive, 38;
RC5TXPORT         = Timer1, positive;
RC5TXPORT         = Timer3, negative, 36;
RC5mode           = rc_7bit;                    //rc_6bit = default
```

Funktionen und Prozeduren[RecvRC5](#)[SendRC5](#)**1.45 RTC Driver*****Import*** SysTick, RTclock, ..;

Option

From RTclock ***Import*** RTctimer, RTCalarm;**Define**

```
ProcClock    = 6000000;           {Hertz}
SysTick      = 10 , Timer2;       {msec}
RTclock      = iData, Time;       {Time or DateTime}
```

Options

```
RTctimer     = 4;                 {1..8 Channels}
RTCsource    = SysTick[, adj];    {optional}
```

Functions and Procedures[RTCalarm](#)[RTCgetHour](#)[RTCsetDay](#)[RTCsetYear](#)[RTctimer_Start](#)[RTCalarm_Date](#)[RTCgetMinute](#)[RTCsetHour](#)[RTctickHour](#)[RTctimer_Stop](#)[RTCalarm_Start](#)[RTCgetMonth](#)[RTCsetMinute](#)[RTctickMinute](#)[RTCalarm_Stop](#)[RTCgetSecond](#)[RTCsetMonth](#)[RTctickSecond](#)[RTCalarm_Time](#)[RTCgetWeekDay](#)[RTCsetSecond](#)[RTctimer](#)[RTCgetDay](#)[RTCgetYear](#)[RTCsetWeekDay](#)[RTctimer_Load](#)

1.46 Serial LAN

Import LANport;

Define

```

LANport      = SerPort;      {SerPort2}
LANctrl      = PortA, 5;     {PortName, bit nummer} optional
LANmode      = Master;      {Master/Slave}
LANbaud      = 57600;       {Baudrate}
LANadr       = 8 [,Mask];   {8 or 16 bits, [masked by LANADRMASK]}
LANframe     = 16, iData;   {Framesize max. 16 bytes in iData}
LANcheck     = ChkSum8;     {ChkSum8, ChkSum16, CRC16}

```

Var

```

LANrxBuff : array [0..LANframe-1] of byte;
LANtxBuff : array [0..LANframe-1] of byte;

```

Functions and Procedures

```

LANrxAutoAck   LANrxStat       LANtxFrame
LANrxClear      LANtxClear       LANtxStat

```

1.47 SerPort

Note:

the old identifiers of the 1. SerPort, eg. SerPort, SerPortDTR, SerInp etc. were renamed by adding the digit "1": SerPort1, SerPortDTR1, SerInp1 etc. The old names are still applicable.

Import SysTick, SerPort1, SerPort2, ..;

XMega:

Import SysTick, SerPortC0, SerPortC1, ..;

Option

From SerPort **import** SerPortSelect;

Define

```

ProcClock    = 4000000;      {4Mhz clock }
SysTick      = 10;           {10msec Tick}
SerPort1     = 9600, Stop1;   {9600 Baud, 1Stopbit}
//SerPort1   = 9600, parEven; {9600 Baud, gerade parity}
//SerPort1   = 9600, parOdd;  {9600 Baud, odd parity}
//SerPort1   = 9600, Stop2, timeout; // Stop2 and timeout are optional
TxBuffer1    = 8;            {8 Byte Buffer and Int}
RxBuffer1    = 8, iData;     {8 Byte Buffer and Int}

```

the Defines of the other serial ports are analog:

```

SerPort2     = 9600, Stop1;   {9600 Baud, 1Stopbit}
...

```

XMega

Define

```

ProcClock    = 4000000;      {4Mhz clock }
SysTick      = 10;           {10msec Tick}
SerPortC0    = 9600, Stop1;   {9600 Baud, 1Stopbit}
//SerPortC0  = 9600, parEven; {9600 Baud, gerade Parität}
//SerPorC0   = 9600, parOdd;  {9600 Baud, ungerade Parität}

```

```
TxBufferC0 = 8;           {8 Byte Buffer und Int}
RxBufferC0 = 8, iData;    {8 Byte Buffer und Int}
```

the Defines of the other serial ports are analog:

```
SerPortC1 = 9600, Stop1;  {9600 Baud, 1Stopbit}
...
```

Options

```
SerPortDTR1 = PinB, 2, Positive; // first serport
SerPortDTR2 = PinB, 3, Negative; // second serport if available
...
SerPortDTRC0 = PortC, 2, Positive; // X Mega first serport
SerPortDTRC1 = PinC, 3, Negative; // X Mega second serport if available
...
```

```
SerPortDSR1 = PortC, 2, Positive; // first serport
SerPortDSR2 = PortC, 3, Negative; // second serport if available
...
SerPortDSRC0 = PortC, 2, Positive; // X Mega first serport
SerPortDSRC1 = PortC, 3, Negative; // X Mega second serport if available
...
```

```
SerCtrl1 = PortD, 2, positive; { control line for RS485 driver }
SerCtrl2 = PortD, 3, positive; { control line for RS485 driver }
...
SerCtrlC0 = PortD, 2, positive; {X Mega control line for RS485 driver}
//SerCtrlC1 = PortD, 2, positive; {X Mega control line for RS485 driver}
```

SerPortSelect

```
SerPortSelect := 3; // switch to serport4
SerPortSelect := 2; // switch to serport3
SerPortSelect := 1; // switch to serport2
SerPortSelect := 0; // switch to serport1
```

X Mega

the variable *SerPortSelect* is not a Byte but of type *tUSARTenum*

```
SerPortSelect := UsartC1; // switch to serportC1
SerOut ('x'); // write 'x' to serportC1
if SerStat then // check serportC1
  ch:= SerInp; // read from SerportC1
endif;
```

```
SerPortSelect := UsartD0; // switch to serportD0
SerOut ('x'); // write 'x' to serportD0
...
```

Type

```
tParity = (parNone, parEven, parOdd);
tDataBits = (DataBit5, DataBit6, DataBit7, DataBit8);
tStopBits = (StopBit1, StopBit2);
```

X Mega:

```
tUSARTenum = (UsartC0, UsartC1, UsartD0, UsartD1, UsartE0, UsartE1, UsartF0, UsartF1);
```

Functions and Procedures ATmega Port1

FlushBuffer	SerInp1	SerInp_TO1	SerPort_Send1
OnSerRxResumed1	SerInpBlock1	SerOut1	SerStat1
OnSerRxStopped1	SerInpBlock1_P	SerOutBlock1	SerStopBits1
SerBaud1	SerInpBlockP_TO1	SerOutBlock1_P	
SerDataBits1	SerInpBlock_TO1	SerOutSLIP1	
Ser_Enable1	SerInpSLIP1	SerParity1	

Functions and Procedures ATmega Port2

FlushBuffer	SerInp2	SerInp_TO2	SerStopBits2
OnSerRxResumed2	SerInpBlock2	SerOut2	SerPort_Send2
OnSerRxStopped2	SerInpBlock2_P	SerOutBlock2	SerStat2
SerBaud2	SerInpBlockP_TO2	SerOutBlock2_P	
SerDataBits2	SerInpBlock_TO2	SerOutSLIP2	
Ser_Enable2	SerInpSLIP2	SerParity2	

Functions and Procedures ATmega Port3, Port4: analog to Port1 and Port2 (see above)

Functions and Procedures XMega PortC0

FlushBuffer	SerInpC0	SerInp_TOC0	SerPort_SendC0
OnSerRxResumedC0	SerInpBlockC0	SerOutC0	SerStatC0
OnSerRxStoppedC0	SerInpBlockC0_P	SerOutBlockC0	SerStopBitsC0
SetSerBaud(UsartC0,	SerInpBlockP_TOC0	SerOutBlockC0_P	
SerDataBitsC0	SerInpBlock_TOC0	SerOutSLIPC0	
Ser_EnableC0	SerInpSLIPC0	SerParityC0	

Functions and Procedures XMega PortC1

FlushBuffer	SerInpC1	SerInp_TOC1	SerPort_SendC1
OnSerRxResumedC1	SerInpBlockC1	SerOutC1	SerStatC1
OnSerRxStoppedC1	SerInpBlockC1_P	SerOutBlockC1	SerStopBitsC1
SetSerBaud(UsartC1,	SerInpBlockP_TOC1	SerOutBlockC1_P	
SerDataBitsC1	SerInpBlock_TOC1	SerOutSLIPC1	
Ser_EnableC1	SerInpSLIPC1	SerParityC1	

Functions and Procedures XMega PortD0, PortD1, PortE0, PortE1, PortF0, PortF1: analog to PortC0 and PortC1 (see above)

1.48 Servo Driver

Import ServoPort;

Define

```

ProcClock    = 8000000;           {8Mhz clock }
ServoPort    = PortB, 2, iData;   {Port, Startbit im Port, data area}
ServoChans   = 4, Positive;       {4 channels, positive pulse}
ServoNeutral = 1.5, Timer1;       {1.5msec neutral position, use Timer1}
ServoSwing   = 1.5;               {adjustable between 0.5 and 1.0msec, Resolution 100
pts}
Optional resolution of 1000 points (*4*)
ServoSwing   = 1.5, 1000;         {adjustable between 0.5 and 1.0msec, Resolution 1000
pts}

```

Functions and Procedures

[SetServoChan](#) [SetServoOffs](#)

1.49 SHT11 Driver

Import SysTick, SHT11drv;

Define

```
ProcClock    = 8000000;           // 8Mhz clock
SysTick      = 5;                 // 5msec
SHT11drv     = polled [, Delay][, crc]; // polling only
// SHT11drv   = SysTickChecked;    // polling + semaphore
SHT11clk     = PortD, 5;          // port and pin for clock
SHT11dat     = PortD, 6;          // port and pin for data
```

Var

```
SHT11sema : Semaphore;
SHT11crc   : byte;
```

Functions and Procedures

<u>SHT11convState</u>	<u>SHT11getStatus</u>	<u>SHT11startTemp</u>
<u>SHT11getHum</u>	<u>SHT11setStatus</u>	<u>SHT11softReset</u>
<u>SHT11getTemp</u>	<u>SHT11startHum</u>	<u>SHT11synchronize</u>

1.50 SLIPport

Import SysTick..., SLIPport1, ...; // SLIPport2, SLIPport3, SLIPport4

XMega

Import SysTick..., SLIPportC0, ...; // SLIPportC1, SLIPportD0, SLIPportD1, SLIPportE0, SLIPportE1...

Define

```
ProcClock    = 16000000;   {Hertz}
SysTick      = 10;
StackSize    = $0020, iData;
FrameSize    = $0040, iData;
SLIPport1    = 19200;      // SLIPport2, SLIPport3, SLIPport4
// SLIPportCtrl1 = PortA, 0, positive; // optional RS484 line driver control
...
```

XMega

Define

```
ProcClock    = 16000000;   {Hertz}
SysTick      = 10;
StackSize    = $0020, iData;
FrameSize    = $0040, iData;
SLIPportC0   = 19200;      // SLIPportC1, SLIPportD0, SLIPportD1...
// SLIPportCtrlC0 = PortA, 0, positive; // optional RS484 line driver control
...
```

Type

```
tSLIPstate = (SLIPidle, SLIPready, SLIPbusy, SLIPover, SLIPtout, SLIPfrm, SLIPchke);
tSLIPmodeEn = (slpHsk, slpChkS, slpAddr);
tSLIPmode = Bitset of tSLIPmodeEn;
```

Var

```
SlipRxSema1 : semaphore;
SlipRxSema2 : semaphore;
...
```

XMega*SlipRxSemaC0 : semaphore;**SlipRxSemaC1 : semaphore;**SlipRxSemaD0 : semaphore;*

...

Functions and Procedures ATmega Port1[SLIPgetRxCount1](#)[SLIPrxReady1](#)[SLIPsetTimeOut1](#)[SLIPstartTx1](#)[SLIPgetRxState1](#)[SLIPsetMode1](#)[SLIPsetTxAddr1](#)[SLIPstopRx1](#)[SLIPgetTxState1](#)[SLIPsetRxAddr1](#)[SLIPsetTxBuffer1](#)[SLIPwasBC1](#)[SLIPPresumeRx1](#)[SLIPsetRxBuffer1](#)[SLIPstartTx1](#)**Functions and Procedures ATmega Port2**[SLIPgetRxCount2](#)[SLIPrxReady2](#)[SLIPsetTimeOut2](#)[SLIPstartTx2](#)[SLIPgetRxState2](#)[SLIPsetMode2](#)[SLIPsetTxAddr2](#)[SLIPstopRx2](#)[SLIPgetTxState2](#)[SLIPsetRxAddr2](#)[SLIPsetTxBuffer2](#)[SLIPwasBC2](#)[SLIPPresumeRx2](#)[SLIPsetRxBuffer2](#)[SLIPstartTx2](#)**Functions and Procedures ATmega Port3, Port4: analog to Port1 and Port2 (see above)****Functions and Procedures XMega PortC0**[SLIPgetRxCountC0](#)[SLIPrxReadyC0](#)[SLIPsetTimeOutC0](#)[SLIPstartTx C_0](#)[SLIPgetRxStateC0](#)[SLIPsetModeC0](#)[SLIPsetTxAddrC0](#)[SLIPstopRxC0](#)[SLIPgetTxStateC0](#)[SLIPsetRxAddrC0](#)[SLIPsetTxBufferC0](#)[SLIPwasBC C_0](#)[SLIPPresumeRxC0](#)[SLIPsetRxBufferC0](#)[SLIPstartTxC0](#)**Functions and Procedures XMega PortC1**[SLIPgetRxCountC1](#)[SLIPrxReadyC1](#)[SLIPsetTimeOutC1](#)[SLIPstartTx C_1](#)[SLIPgetRxStateC1](#)[SLIPsetModeC1](#)[SLIPsetTxAddrC1](#)[SLIPstopRxC1](#)[SLIPgetTxStateC1](#)[SLIPsetRxAddrC1](#)[SLIPsetTxBufferC1](#)[SLIPwasBC C_1](#)[SLIPPresumeRxC1](#)[SLIPsetRxBufferC1](#)[SLIPstartTxC1](#)**Functions and Procedures XMega PortD0, PortD1, PortE0, PortE1, PortF0, PortF1:
analog to PortC0 and PortC1 (see above)**

1.51 SpeechPort

Import SpeechPort ; {not SPI and **XMega**}

or

Import SpeechPort, SPIdriver ;**Define**

ProcClock = 8000000; {8Mhz clock }

SpeechPort = SPI; // SPIdriver must be imported and defined

or

SpeechPort = UserPort; // uses an user defined driver

or

SpeechPort = PortG, 0; // bit serial using PortG, bit0=DATA, 1=CLK, 2=SEL

or

SpeechPort = PortC; // 8bit parallel output on port C

SpeechTimer = Timer0; // Timer0, Timer1, Timer2, Timer3

or

SpeechTimer = Timer1;

```

or
  SpeechTimer = Timer2;
or
  SpeechTimer = Timer3;
  SPIorder    = LSB;           // SPI define only necessary if SpeechPort = SPI;
  SPIcpol     = 1;
  SPIcpha     = 1;
  SPIpresc    = 0;             // presc = 0..3 -> 4/16/64/128
  SPI_SS      = true;          // use SS pin as chipselect
...
or XMega
  SpeechTimer = Timer_C0;      // _C1, _D0, _D1, ...
  SpeechPort = SPI_C, SPImode3, SPImsb, PortF, 4; // Mode 0..3, MSB/LSB, SS-Port, SS-Pin

```

```

UserDevice SpeechIOS(b : byte);
begin
  ...
end;

```

Funktionen und Prozeduren

[SpeechOutFlash](#) [SpeechOutRAM](#) [SpeechReady](#) [SpeechStop](#)

1.52 Software PWM

Import SysTick, SoftPWM;

Define

```

  ProcClock      = 8000000;      {Hertz}
  SysTick        = 10;           {msec}
  StackSize      = $0032, iData;
  FrameSize      = $0032, iData;
  SoftPWMport     = PortA;        {use PortA for PWM output}
  SoftPWMchans    = 4, 0;         {4 channels, bit0/PortA is the first}
  //SoftPWMchans  = 4, 2, negative; {4 channels, bit2/PortA 1.bit, low pulsed}
  SoftPWMtimer    = timer2, 1;    {use timer2, PWM cycle time 10msec = 200Hz}
  SoftPWMres      = 16;           {PWM resolution is 16 points}

```

Var

SoftPWM1, SoftPWM2, ... : Byte;

Funktions and Procedures

[SoftPWMstart](#) [SoftPWMstop](#)

1.53 SPI Driver MSPI-ATMega

import SysTick, MSPI0, MSPI1, ..;

Define

```

  ProcClock      = 8000000;      {Hertz}
  SysTick        = 10;           {msec}
  StackSize      = $0030, iData;
  FrameSize      = $0030, iData;
  MSPIorder0     = MSB;
  MSPIcpol0      = 1;

```

```

MSPIcpha0 = 1;
MSPIpresc0 = 1;
....
MSPIpresc1 = 1;

```

Functions and Procedures for port MSPI0

<u>MSPIinOut0</u>	<u>MSPIinpByte0</u>	<u>MSPIlout0</u>	<u>MSPIloutWord0</u>	<u>SetMSPIclkPol0</u>
<u>MSPIinOutByte0</u>	<u>MSPIinpLong0</u>	<u>MSPIloutByte0</u>	<u>SetMSPI0mode</u>	<u>SetMSPIorder0</u>
<u>MSPIinp0</u>	<u>MSPIinpWord0</u>	<u>MSPIloutLong0</u>	<u>SetMSPIclkPha0</u>	<u>SetMSPIpresc0</u>

Functions and Procedures for port MSPI1

<u>MSPIinOut1</u>	<u>MSPIinpByte1</u>	<u>MSPIlout1</u>	<u>MSPIloutWord1</u>	<u>SetMSPIclkPol1</u>
<u>MSPIinOutByte1</u>	<u>MSPIinpLong1</u>	<u>MSPIloutByte1</u>	<u>SetMSPI1mode</u>	<u>SetMSPIorder1</u>
<u>MSPIinp1</u>	<u>MSPIinpWord1</u>	<u>MSPIloutLong1</u>	<u>SetMSPIclkPha1</u>	<u>SetMSPIpresc1</u>

Functions and Procedures for Ports MSPI2 and MSPI3: analog to ports MSPI0 and MSPI1

1.54 SPI Driver MSPI-XMega

import SysTick, MSPI_C0, MSPI_C1, ..;

Define

```

// The XMegas don't provide any Oscillator fuses.
// So the application must setup the desired values
// possible OSC types: extXTAL, extClock, ext32kHz, int32Khz, int2MHz, int32MHz
//>> CPU=32MHz, PeripherX4=32MHz, PeripherX2=32MHz
OSCType          =      int32MHz,
                   PLLmul=4,
                   prescB=1,
                   prescC=1;

SysTick           =      10;           {msec}
StackSize         =      $0030, iData;
FrameSize        =      $0030, iData;
MSPIorder_C0     =      MSB;
MSPIMode_C0      =      1;             // 0, 1, 2, 3
MSPIpresc_C0     =      1;             // presc = 1..7
MSPI_SS_C0       =      PortE, 4;
MSPIorder_C1     =      MSB;
MSPIMode_C1      =      0;             // 0, 1, 2, 3
MSPIpresc_C1     =      3;             // presc = 1..7
MSPI_SS_C1       =      none;
....

```

Functions and Procedures for port MSPI_C0

<u>MSPIinOut_C0</u>	<u>MSPIinpLong_C0</u>	<u>MSPIloutLong_C0</u>	<u>SetMSPImode_C0</u>
<u>MSPIinOutByte_C0</u>	<u>MSPIinpWord_C0</u>	<u>MSPIloutWord_C0</u>	<u>SetMSPIorder_C0</u>
<u>MSPIinp_C0</u>	<u>MSPIlout_C0</u>	<u>SetMSPIclkPha_C0</u>	<u>SetMSPIpresc_C0</u>
<u>MSPIinpByte_C0</u>	<u>MSPIloutByte_C0</u>	<u>SetMSPIclkPol_C0</u>	

Functions and Procedures for port MSPI_C1

<u>MSPIinOut_C1</u>	<u>MSPIinpLong_C1</u>	<u>MSPIloutLong_C1</u>	<u>SetMSPImode_C1</u>
<u>MSPIinOutByte_C1</u>	<u>MSPIinpWord_C1</u>	<u>MSPIloutWord_C1</u>	<u>SetMSPIorder_C1</u>
<u>MSPIinp_C1</u>	<u>MSPIlout_C1</u>	<u>SetMSPIclkPha_C1</u>	<u>SetMSPIpresc_C1</u>
<u>MSPIinpByte_C1</u>	<u>MSPIloutByte_C1</u>	<u>SetMSPIclkPol_C1</u>	

*Functions and Procedures for Port MSPI_D0, _D1, _E0, _E1, _F0, _F1:
analog to ports MSPI_C0 and MSPI_C1*

1.55 SPI Hardware Driver

Import SysTick, SPIdriver, ...;

XMega

Import SysTick, SPI_C, ...; // SPI_D, SPI_E, SPI_F

Define

```
ProcClock    = 8000000;    {Hertz}
SysTick      = 10;        {msec}
StackSize    = $0030, iData;
FrameSize    = $0030, iData;
SPIdorder    = MSB;
SPIdcpol     = 1;
SPIdcpol     = 1;
SPIdpresc    = 1;          // presc = 0..3 -> 4/16/64/128
SPIdSS       = false;      // don't use SS pin as chipselect
```

XMega

Define

```
OSCType      = int32MHz, PLLmul=4, prescB=1, prescC=1; // all 32MHz
SysTick      = 10;        {msec}
StackSize    = $0030, iData;
FrameSize    = $0030, iData;
SPIdorderC   = MSB;
SPIdmodeC    = 0;          // Clock Phase and Polarity
SPIdprescC   = 1;          // presc = 0..3 -> 4/16/64/128
SPIdSSC      = PortB, 3;   // use this pin as SS chipselect
```

Functions and Procedures ATmega

<u>SetSPIdclkPha</u>	<u>SetSPIdorder</u>	<u>SPIinOutByte</u>	<u>SPIout</u>
<u>SetSPIdclkPol</u>	<u>SetSPIdpresc</u>	<u>SPIinpByte</u>	<u>SPIoutByte</u>
<u>SetSPIdoubleSpeed</u>	<u>SPIinp</u>	<u>SPIinpLong</u>	<u>SPIoutLong</u>
<u>SetSPIdmode</u>	<u>SPIinOut</u>	<u>SPIinpWord</u>	<u>SPIoutWord</u>

Funktionen und Prozeduren XMega Port_C

<u>SetSPIdclkPhaC</u>	<u>SetSPIdprescC</u>	<u>SPIinpLongC</u>	<u>SPIoutLongC</u>
<u>SetSPIdclkPolC</u>	<u>SPIinpC</u>	<u>SPIinpLong64C</u>	<u>SPIoutLong64C</u>
<u>SetSPIdoubleSpeedC</u>	<u>SPIinOutC</u>	<u>SPIinpWordC</u>	<u>SPIoutWordC</u>
<u>SetSPIdmodeC</u>	<u>SPIinOutByteC</u>	<u>SPIoutC</u>	
<u>SetSPIdorderC</u>	<u>SPIinpByteC</u>	<u>SPIoutByteC</u>	

Funktionen und Prozeduren XMega Port_D

<u>SetSPIdclkPhaD</u>	<u>SetSPIdprescD</u>	<u>SPIinpLongD</u>	<u>SPIoutLongD</u>
<u>SetSPIdclkPolD</u>	<u>SPIinpD</u>	<u>SPIinpLong64D</u>	<u>SPIoutLong64D</u>
<u>SetSPIdoubleSpeedD</u>	<u>SPIinOutD</u>	<u>SPIinpWordD</u>	<u>SPIoutWordD</u>
<u>SetSPIdmodeD</u>	<u>SPIinOutByteD</u>	<u>SPIoutD</u>	
<u>SetSPIdorderD</u>	<u>SPIinpByteD</u>	<u>SPIoutByteD</u>	

Funktionen und Prozeduren XMega Port_D and Port_E:
anlog to XMega Port_C and XMega Port_D

1.56 SPI Network

Import SysTick, SPIport;

Define

```
SPIport      = Master;
or
SPIport      = Slave;
SPIOrder     = MSB; {LSB}
SPICPOL      = 0;    {0/1}
SPICPHA      = 0;    {0/1}
SPIpresc     = 0;    {0/1/2/3}
SPIretry     = 10;
```

Var

```
SpiTxBuff : array [0..SPIbufferLen -1] of byte;
SpiRxBuff : array [0..SPIbufferLen -1] of byte;
SpiRxLen : byte;
SpiTxLen : byte;
```

Functions and Procedures

<u>SPItxClear</u>	<u>SPIout</u>	<u>SPIrxFrame</u>
<u>SPIinp</u>	<u>SPIrxClear</u>	<u>SPIrxStat</u>

1.57 SPI Soft Driver

Import SysTick, SPIdriver1, ...; // or SPIdriver2 or both

Define

```
ProcClock    = 16000000;    {Hertz}
SysTick      = 10;          {msec}
StackSize    = $0030, iData;
FrameSize    = $0040, iData;
SPIdriver1   = PortA, 0, 1, 2, 3; // SCK, MOSI, MISO, SS
SPIorder1    = MSB;
SPICpol1     = 1;
SPICpha1     = 1;
SPIdriver2   = PortC, 7, 4, 1, 5; // SCK, MOSI, MISO, SS
SPIorder2    = MSB;
SPICpol2     = 0;
SPICpha2     = 0;
```

Functions and Procedures

<u>SPIinOutByte1</u>	<u>SPIinpByte1</u>	<u>SPIinpWord1</u>	<u>SPIoutByte1</u>	<u>SPIoutWord1</u>
<u>SPIinOutByte2</u>	<u>SPIinpByte2</u>	<u>SPIinpWord2</u>	<u>SPIoutByte2</u>	<u>SPIoutWord2</u>
<u>SPIinOut1</u>	<u>SPIinpLong1</u>	<u>SPIinp1</u>	<u>SPIoutLong1</u>	<u>SPIout1</u>
<u>SPIinOut2</u>	<u>SPIinpLong2</u>	<u>SPIinp2</u>	<u>SPIoutLong2</u>	<u>SPIout2</u>

1.58 Stepper Driver

Import SysTick, StepPort..

From System **Import** longword;

Option

From StepPort **Import** StepperSema;

Define

```
ProcClock    = 8000000;    {Hertz}
StackSize    = $0030, iData;
FrameSize    = $0010, iData;
StepPort     = PortA;
StepMinFreq  = 100;
StepMaxFreq  = 5000;
StepType     = StepM6;
```

XMega

```
StepTimer    = Timer_C0;  {Timer_C1, _D0, _D1, ...}
```

StepTypes

```
StepFull 2, StepFull 4, SteppHalf 4, StepHalf 6, StepMini 4
StepMini 6, StepMicro 2, StepMicro 8, UserPort
```

Type

```
TStepMode    = (StepStop, StepUp, StepRun, StepDown);
```

Var

```
StepStartFreq    : word;
StepEndFreq      : word;
StepAccValue     : word;
StepCount        : longword;    {max 2^32 Steps}
StepMode         : TStepMode;
```

Funktionen und Prozeduren

<u>StepDestCCW</u>	<u>StepperOff</u>	<u>StepPanicStop</u>	<u>StepRampStop</u>
<u>StepDestCW</u>	<u>StepOneCW</u>	<u>StepRampCW</u>	<u>StepVelocity</u>
<u>StepperOn</u>	<u>StepOneCCW</u>	<u>StepRampCCW</u>	

1.59 Switchport

Import SysTick, SwitchPort1, ..;

or

Import SysTick, SwitchPort2, ..;

or

Import SysTick, SwitchPort_G, ..;

Define

```
ProcClock    = 8000000;    {Hertz}
SysTick      = 10;         {msec}
StackSize    = $0030, iData;
FrameSize    = $0030, iData;
SwitchPort1  = PinB;       {SwitchPort}
```

or

```

SwitchPort1 = PinB, $17;    {SwitchPort, edgemask}
SwitchPort_G = [ResetKey, PinC, 4] [StartInput, PinC, 5] [Sensor, PinB, 1], %00000001;
options
PolarityP1   = %00000110; {Polarity SwitchPort1}
Debounce     = 5;          {debounce every 5 SysTicks}

```

Var

```

PORT_STABLE1   : byte;
PORT_STABLE2   : byte;
PORT_STABLE_G  : byte;

```

Functions and Procedures

Inp_Raise1	Inp_Stable1	SwitchKeyRepeat1	SwitchPort1_Clear
Inp_Raise2	Inp_Stable2	SwitchKeyRepeat2	SwitchPort2_Clear
Inp_Raise_G	Inp_Stable_G	SwitchKeyRepeat_G	SwitchPort_G_Clear

1.60 SysLeds

Import SysTick, SysLEDBlink, ...;

Options

From SysLEDBlink **Import** LEDmessage, FlashOnce; // this is optional

Define

```

ProcClock    = 8000000;           {Hertz}
SysTick       = 10;                {msec}
StackSize     = $0030, iData;
FrameSize     = $0030, iData;
SysLEDBlink   = 30;                {30*SysTick = 300msec}
// alternative
//SysLEDBlink = mSec300;           {10..1000 msec}
SysLEDBlink0 = PortA, 0, high;     {LEDon = high level}
SysLEDBlink1 = PortA, 1, low;      {LEDon = low level}
SysLEDBlink2 = PortA, 2, low;
SysLEDBlink3 = PortA, 3, low;
SysLEDBlink4 = PortA, 4, low;
SysLEDBlink5 = PortA, 5, low;
SysLEDBlink6 = PortA, 6, low;
SysLEDBlink7 = PortA, 7, low;
// alternative
//SysLedPort = @LEDram, $00;       {byte-var, polarity}

```

Functions and Procedures

SetSysBlinkTimer	SysLEDflashAllOn	SysLEDflashOn	SysLEDon
SysLEDallOn	SysLEDflashAllOff	SysLEDflashOnce	SysLEDonOff
SysLEDallOff	SysLEDflashMsg	SysLEDflashOnOff	
SysLEDenable	SysLEDflashOff	SysLEDoff	

1.61 TINA TCP/IP

Import SysTick, TINASTack, ...;

Define

```
ProcClock = 16000000;      // Hertz
StackSize = $0080, iData;  // min size
FrameSize = $00C0, iData;  // min size
xData = $8000, $87ff;      // 2kB optional, only necessary with the xData define below
TINADriver = ENC28J60[, xData]; // TINA hardware, optional buffers in xData
// TINADriver = ENC424J600; // TINA hardware, alternative chip
TINAport = SPI, PortB, 0;   // SPI typ, SS_Port, SS_Pin
// TINAport = MSPI0, PortA, 4; // SPI typ, SS_Port, SS_Pin
TINAtimer = Timer3;        // 1..3
TINAsockets = 4;           // socket count, 1..8
```

XMega

Mit den XMegas werden bis zu vier SPI-ports unterstützt:

SPI_C, SPI_D, SPI_E or SPI_F

Deshalb muss hier der SPI Port definiert werden:

TINAport = SPI_C, PortF, Pin3;

Die XMegas bieten bis zu 8 Timer:

Timer_C0, Timer_C1, Timer_D0, Timer_D1 etc.

Einer von diesen muss benutzt werden:

TINAtimer = Timer_F1;

Type

TMACaddr = **array** [0..5] of byte;

TIPaddr = **array** [0..3] of byte;

TTINASTatus = (TinasNoErrors, TinasInvalidHandle, TinasInitFailed,
TinasNotInitialized, TinasSockClosed, TinasBufferParam,
TinasSendFailed, TinasTimeOutErr, TinasListenFailed,
TinasSockConnected, TinasSockListen, TinasSockCloseWait,
TinasSockClosing, TinasSockUDP, TinasSockRaw);

TTinaPriority = (TinaPrioMedium, TinaPrioLow, TinaPrioHigh, TinaPrioVeryHigh,
TinaPrioVeryLow, TinaPrioAuto, TinaPrioSuspend, TinaPrioResume);

TtinaPacketReceive = **Record**

```
PeerIP      : tIPaddr;
PeerPort    : Word;
RecLen      : Word;
BufferPtr   : Pointer;
BufferLen   : Word;
```

end;

```
TTinaSocketSWS = (NoSillyWindow, SillyWindow);
TTinaSocketNDack = (NoDelayedAck, DelayedAck);
TTinaNDTimeOut = (NoDynamicTimeOut, DynamicTimeOut);
TTinaBroadcast = (NoBroadcast, Broadcast);
TTinaxUDPAKNPort = (xAKNLocalPort, xAKNRemotePort);
```

TTinaSocket = **Record**

```
Protocol      : TProtocolType;
Swindow       : TTinaSocketSWS;
DelayAck      : TTinaSocketNDack;
```

```

        DynTimeOut      : TTinaNDTimeOut;
        Broadcast       : TTinaBroadcast;
        LocalPort       : Word;
        RemoteHost      : tIPAddr;
        RemotePort      : Word;
        TimeOut         : LongWord;
        RetryCount      : Byte;
        AKNPort         : TTinaaxUDPAKNPort;
        PeerTryToDisconnect : Boolean;
        SocketClosed    : Boolean;
        PacketRecInfo   : TTinaPacketReceive;
        ErrorState      : TTINAStatus;
        SocketState     : Byte;
    end;
    tSocketHandle = Pointer to TTinaSocket;

    TTinaCore = Record
        IP              : tIPAddr;
        Mask             : tIPAddr;
        Gateway         : tIPAddr;
        Mac             : TMacAddr;
        TimeOut         : LongWord;
        Retry           : Byte;
        Prio            : TTinaPriority;
        ResponsePing    : Boolean;
        RXChecksumCheck : Boolean;
        SendICMPCtrlMessages : Boolean;
    end;

    var
        TinaCore : TTinaCore;

```

Functions and Procedures

<u>CompareIP</u>	<u>StrToMAC</u>	<u>TINAlinkStat</u>	<u>TINA_Init</u>
<u>CompareNet</u>	<u>SwapIPAddr</u>	<u>TINApacketReceived</u>	<u>TINA_Ping</u>
<u>CompareMAC</u>	<u>SwapMACAddr</u>	<u>TINAresumeReceive</u>	<u>TINA_Start</u>
<u>IPToStr</u>	<u>TINAccreateSocket</u>	<u>TINArxStat</u>	<u>TINA_Stop</u>
<u>MACToStr</u>	<u>TINAFreeSocket</u>	<u>TINAsendPacket</u>	
<u>StrToIP</u>	<u>TINAinitSocket</u>	<u>TINASetPriority</u>	

1.62 Tick Timer, TickTimer2

Import SysTick, TickTimer, ...;

Define

```

    ProcClock   = 16000000;           {Hertz}
    SysTick     = 10;                 {msec}
    StackSize   = $0030, iData;
    FrameSize   = $0030, iData;
    TickTimer   = Timer1;             // use Timer1.COMPA and no PortPin
    //TickTimer = Timer1, pinout;      // use Timer1.COMPA and its PortPin
    //TickTimer = Timer2;             // use Timer2.COMPA and no PortPin
    //TickTimer = Timer2, pinout;      // use Timer2.COMPA and its PortPin
    //TickTimer = Timer3;             // use Timer3.COMPA and no PortPin
    //TickTimer = Timer3, pinout;      // use Timer3.COMPA and its PortPin

```

Var *TickTimerPin* : bit;

xMega:

supports also TickTimer2

Define

```
TickTimer      = Timer_C0;
TickTimer2     = Timer_C1;
oder
TickTimer      = Timer_D0, PortE, 4;
TickTimer2     = Timer_D1, PortE, 5;
```

Functions and Procedures

[TickTimerOutpEnable](#) [TickTimerReload](#) [TickTimerStop](#)
[TickTimerRawVal](#) [TickTimerStart](#) [TickTimerTime](#)

additional Functions and Procedures for xMega

[TickTimer2OutpEnable](#) [TickTimer2Reload](#) [TickTimer2Stop](#)
[TickTimer2RawVal](#) [TickTimer2Start](#) [TickTimer2Time](#)

1.63 TWI (I2C) Port

Import SysTick, TWImaster;

Define

```
ProcClock      = 4000000;    {4Mhz clock }
SysTick        = 10;        {10msec Tick}
TWIpresc       = 32        (TWI speed)
```

or

Import SysTick, TWISlave;

or **xMega**

Import SysTick, TWI_C; // TWI_C, TWI_D, TWI_E, TWI_F

Note:

xMegas do not support MasterSlave und Slave Modes!

Define

```
ProcClock      = 4000000;    {4Mhz clock }
SysTick        = 10;        {10msec Tick}
TWIaddr        = 2;        (TWI slave address}
TWIbuffer      = 8, iData;    (TWI rx/tx buffersize, location}
TWImode        = Handshake;    (TWI handshake or transparent}
```

xMega

Define

```
OSCtype        = int32MHz, PLLmul=4, prescB=1, prescC=1;
SysTick        = 10;        {10msec Tick}
TWIprescC      = TWI_BR400;    (TWI speed)
```

Const

```
TWI_BR100 : byte = nn;        // nn = prescaler value for 100kBits/sec
TWI_BR400 : byte = nn;        // nn = prescaler value for 400kBits/sec
TWI_BR500 : byte = nn;        // nn = prescaler value for 500kBits/sec
TWI_BR600 : byte = nn;        // nn = prescaler value for 600kBits/sec
TWI_BR800 : byte = nn;        // nn = prescaler value for 800kBits/sec
```

Var

```
TWI_DevLock : DEVICELOCK;
TWI_DevLockTN : DEVICELOCK;    // TN = C, D, E, F
```

Statusbytes in TWISlave Mode without handshake: *TWISlvTxCount*, *TWISlvRxCount*.

Functions and Procedures

<u>TWIgetBusy</u>	<u>TWIgetRxStat</u>	<u>TWlinpP</u>	<u>TWloutWP</u>	<u>TWisetRdy</u>
<u>TWIgetCmd</u>	<u>TWIgetTxStat</u>	<u>TWlout</u>	<u>TWisetBusy</u>	<u>TWisetSlaveAddr</u>
<u>TWIgetRdy</u>	<u>TWlinp</u>	<u>TWloutP</u>	<u>TWisetGC</u>	<u>TWlstat</u>

Functions and Procedures für X Mega

<u>TWlinpC</u>	<u>TWlinpD</u>	<u>TWlinpE</u>	<u>TWlinpF</u>
<u>TWlinpPC</u>	<u>TWlinpPD</u>	<u>TWlinpPE</u>	<u>TWlinpPF</u>
<u>TWloutC</u>	<u>TWloutD</u>	<u>TWloutE</u>	<u>TWloutF</u>
<u>TWloutPC</u>	<u>TWloutPD</u>	<u>TWloutPE</u>	<u>TWloutPF</u>
<u>TWloutWPC</u>	<u>TWloutWPD</u>	<u>TWloutWPE</u>	<u>TWloutWPF</u>
<u>TWlstatC</u>	<u>TWlstatD</u>	<u>TWlstatE</u>	<u>TWlstatF</u>

1.64 TWI Network

Import SysTick, TWInet;

Define

```
ProcClock    = 16000000;    {16Mhz clock }
TWIpresc     = TWI_BR100;   {TWI speed 100kBit/sec}
TWInode       = $12;        {Node Addr, always necessary}
TWInetMode    = Master;     {Master, Slave or MasterSlave}
TWIframe      = 16, iData;   {Framesize max. 16 bytes in iData}
Option
  TWIframeBC = 64;          {Broadcast Framesize max. 16 bytes}
```

Type

```
tTWInetmode = (TWInetSlave, TWInetMaster);
tTWIStates  = (TWIRxEmpty, TWIRxBusy, TWIRxFull, TWITxEmpty,
               TWITxBusy, TWITxFull, TWIbcCMD, TWlstatFail);
tTWInetState = set of tTWIStates;
```

Const

```
TWI_BR100 : byte = nn;    // nn = prescaler value for 100kBits/sec
TWI_BR400 : byte = nn;    // nn = prescaler value for 400kBits/sec
TWI_BR500 : byte = nn;    // nn = prescaler value for 500kBits/sec
TWI_BR600 : byte = nn;    // nn = prescaler value for 600kBits/sec
TWI_BR800 : byte = nn;    // nn = prescaler value for 800kBits/sec
```

Var

```
TWlrxStatReg : byte;
TWltxStatReg : byte;
TWlrxAdr      : byte;
TWltxAdr      : byte;
TWlrxLen      : byte|word;
TWltxLen      : byte|word;
TWlrxBuff     : array [0..TWIframe -1] of byte;
TWltxBuff     : array [0..TWIframe -1] of byte;
```

```
TWI_DevLock : DEVICELOCK;
```

Functions and Procedures

[GetTWISlaveStat](#)
[SetTWINodeAddr](#)
[SetTWImode](#)

[TWIrxClear](#)
[TWIrxFram](#)
[TWIrxFram](#)

[TWITxBroadcast](#)
[TWITxClear](#)
[TWITxFram](#)

[TWITxStat](#)

1.65 wizNet TCP/IP

Import SysTick, TWIMaster, wzNet4, ...;

From System Import Tasks, Processes;

Define

```
ProcClock    = 16000000;           // Hertz
SysTick      = 5;                  // msec
StackSize    = $0040, iData;       // min size
FrameSize    = $00C0, iData;       // min size
Scheduler    = iData;
TaskStack    = $0040, iData;       // min size
TaskFrame    = $00C0;              // min size
wzNet4       = I2C_TWI, iData;      // hardware I2C driver, var loc
wzSocks      = 1;                  // socket count, 1..4
TWIpresc     = TWI_BR400;          // max TWI speed
```

Import SysTick, I2Cport, wzNet4, ...;

From System Import Tasks, Processes;

Define

```
ProcClock    = 16000000;           // Hertz
SysTick      = 5;                  // msec
StackSize    = $0040, iData;       // min size
FrameSize    = $00C0, iData;       // min size
Scheduler    = iData;
TaskStack    = $0040, iData;       // min size
TaskFrame    = $00C0;              // min size
wzNet4       = I2C_Soft, iData;     // software I2C driver, var loc
I2Cport      = PortA;
I2Cclk       = 1;
I2Cdat       = 2;
wzSocks      = 1;                  // socket count, 1..4
```

Type

```
TMACaddr     = array [0..5] of byte;
TIPaddr      = array [0..3] of byte;
TwzStatus    = (wzsNoErrors, wzsInvalidHandle, wzsInitFailed,
wzsNotInitialized, wzsSockClosed, wzsBufferParam,
wzsSendFailed, wzsTimeOutErr, wzsListenFailed,
wzsSockConnected, wzsSockListen, wzsSockCloseWait,
wzsSockClosing, wzsSockUDP, wzsSockRaw);
TwzPriority   = (wzPrioLow, wzPrioMedium, wzPrioHigh, WzPrioVeryHigh,
wzPrioAuto, wzPrioSuspend, wzPrioResume);
TwzPacketReceive = Record
    PeerIP      : tIPaddr;
    PeerPort    : Word;
    BufferPtr    : Pointer;
    BufferLen    : Word;
```

```

end;
TwzSocketSWS = (NoSillyWindow, SillyWindow);           // internal use
TwzSocketNDAck = (NoDelayedAck, DelayedAck);           // internal use
TwzNDTimeOut = (NoDynamicTimeOut, DynamicTimeOut);    // internal use
TwzBroadcast = (NoBroadcast, Broadcast);              // internal use
TwzSocketProtocol = (CLOSED, protTCP, protUDP, protIPRAW, protMACRaw);
TwzSocket = Record
    Protocol           : TwzSocketProtocol;
    SWindow            : TwzSocketSWS;                // internal use
    DelayAck            : TwzSocketNDAck;              // internal use
    DynTimeOut          : TwzNDTimeOut;                // internal use
    Broadcast           : TwzBroadcast;                // internal use
    LocalPort           : Word;
    RemoteHost          : tIPAddr;                    // Client mode
    RemotePort          : Word;                       // Client mode
    IPProtocol          : Byte;                       // internal use
    TypeOfService       : Byte;                       // internal use
    MaxSegSize          : Word;                       // internal use
    PeerTryToDisconnect : Boolean;
    SocketClosed        : Boolean;
    PacketRecInfo       : TwzPacketReceive;
    ErrorState          : TwzStatus;                  // internal use
    SocketState         : byte;                       // semaphore
end;
tSocketHandle = Pointer to twzSocket;

var
    TWI_DevLock : DEVICELOCK;

Functions and Procedures
CompareIP           wzCreateSocket       wzReset             wzTelnetClose
CompareNet          wzDisConnect        wzResumeReceive     wzTelnetConnected
CompareMAC         wzDNSQueryHost      wzSendBuffer        wzTelnetCreate
IPtoStr             wzFreeSocket       wzSetDNSserver      wzTelnetEcho
MACtoStr            wzGetLastError     wzSetGatewayAddr   wzTelnetFree
StrToIP             wzGetSocketState   wzSetHWAddr        wzTelnetGetClient
StrToMAC            wzInit             wzSetIPAddr        wzTelnetGetState
SwapIPAddr          wzInitSocket       wzSetPriority       wzTelnetIdleTimeout
SwapMACAddr         wzListen           wzSetRetryCount    wzTelnetListen
wzAcceptConnection wzPacketReceived  wzSetSNTPserver    wzTelnetRead
wzClientConnected wzReceiveBuffer   wzSetTimeOut       wzTelnetWrite
wzConnect           wzReInitSocket    wzSNTPQueryDateTime wzTelnetWriteLn

```

2 Switches

2.1 Overview Switches

Group: [Compiler Switch](#) 

Compiler Switches serve to control the behavior of the compiler. These switches are components of the source.

A switch is started with a { and a following \$ without a space. The switch name must immediately follow the \$, written in upper case. Further parameters follow the usual conventions.

The switches must start in the first column of a line. The same line may not contain any statements.

Syntax: **{\$SWITCH [arg] }**

Each Import and each Device Define is added to the compiler switch list. This means that:

Import *LCDport, ...*

is treated like: {\$DEFINE LCDPORT} and

Define *ProcClock = 8000000;*

is treated like: {\$DEFINE PROCCLOCK}

Conditional Compile

Sometimes it is necessary to generate different, for example hardware dependant versions of an application. The respective behaviour can be controlled with the help of the compiler switch for dependant compilation (Conditional Compile). The used "**Label**" is only a symbolic character. If the result of a switch is "false", then the following source-code is treated as a comment (as if it does not exist) until the switch is "true". All switches in this group can appear at any position within the source.

An "IFxx"-introduction must be finished with an "ENDIF". There can be an "ELSE" in between.

Nested IF's are permitted!

It is also possible to pass one or more DEFINES from the IDE PED32 to the compiler by using the menu item **Project/Project Options** and define a Label (without the \$, parenthesis and DEFINE).

The parameters and labels must be separated with semicolons. These parameters are then treated by the compiler as if they are in the first source line and have the outline *{\$DEFINE Label}*.

2.2 \$ANALYSIS_ON

{\$ANALYSIS_ON}

Group: [CompilerSwitch](#) 

Compiler switch for optimizer.

Enables the creation of the analysis file.

2.3 \$BDATA

{\$BDATA #}

Group: [CompilerSwitch](#) 

Banked External Memory

In order to populate the different banks with any var types, the user must define where (in which

bank) a specific var resides. This must be done with the compiler switch `{$BDATA #}` where # is the desired bank number.

In this banking implementation up to 16 banks with 32kB each can be defined and accessed. This results in an additional memory size of 512kBytes.

2.4 \$BootApplication

`{$BootApplication $nnnn}` // word address, e.g. `{$BootApplication $0F000}`

Group: [CompilerSwitch](#) 

Combination of [\\$CodeStart](#) and [\\$VectTab](#). For special purposes.
Allows to build an application that runs in the Boot Section.
So the Application and Boot program can be independently created.

2.5 \$BOOTRST

`{$BOOTRST nnnn}`

Group: [CompilerSwitch](#) 

This switch informs the Debugger/Simulator E-LAB AVRsim whether the Hardware Reset uses the vector at Addr \$0000 or the reset jumps into the Bootblock. The parameter \$nnnn defines, if activated, the Reset Jump address for the Simulator at reset time. With the real CPU the [Fusebit BOOTRST](#) programmed by the InCircuit Programmer defines the behaviour of the CPU with a hardware reset. This switch is meaningless for the generated program and therefore also has no influence of the working of the CPU itself. The address parameter can be arbitrary, but only makes sense if it is in accordance with to the possible Jump addresses of the used CPU.

2.6 \$CodeStart

`{$CodeStart $nnnn}` // word address, e.g. `{$CodeStart $0F000}`

Group: [CompilerSwitch](#) 

Forces the start of the Code Area to the specified address.
For special purposes e.g. the creation of an application that is placed in the BootBlock.

see also: [\\$BootApplication](#) and [\\$VectTab](#)

2.7 \$D

`{$D+}` / `{$D-}`

Group: [CompilerSwitch](#) 

Debug informations on or off. If off, the following statements are not stepped by the simulator.
This switch does not have any meanings for the generated Hexfile, i.e. it's not necessary to remove it.

2.8 \$DATA

`{ $DATA }`

Group: [CompilerSwitch](#) 

\$DATA assigns the area, which is found in the processor control file (xxx.dsc) within DATA, in all following variable declarations (including **structconst**) are placed. With the AVR this area is from \$04 upto \$1F. The defined variables now are progressively placed from \$04. If another switch of this type occurs later, the following variables are processed analogously. Variables in the area \$DATA are always reachable with very short and fast machine commands.

2.9 \$DEBDELAY

`{ $DEBDELAY }`

Group: [CompilerSwitch](#) 

For the simulator. Shortens the mDelays in the Simulators by 90%.
This switch does not have any meanings for the generated Hexfile, i.e. it's not necessary to remove it.

2.10 \$DEFINE

`{ $DEFINE label }`

Group: [CompilerSwitch](#) 

Sets "label" to true

2.11 \$DEPHASE

`{ $DEPHASE }`

Group: [CompilerSwitch](#) 

Switches to a fixed **WORD**-Address in Flash and back to the standard code page.

`{ $PHASE $1E00 }`; places the following code at addr \$1E00 and up.
`{ $DEPHASE }`; switches the address generation back to the default page.

2.12 \$DEVICE

`{ $DEVICE }`

Group: [CompilerSwitch](#) 

This is only valid in connection with Device driver procedures, which are called by the procedures **Write** or **Read**. In the following Device procedure, which may only have an 8bit parameter; this parameter is passed in a register. A parameter frame is not built, so local variables are not possible.

It only makes sense for fast driver routines written in assembler. Identical with the \$NOFRAME switch.

2.13 \$EEPROM

{\$EEPROM}

Group: [CompilerSwitch](#) 

\$EEPROM assigns the area, which is found in the processor control file (xxx.dsc) within EEprom, to all following variable declarations (including **structconst**). EEprom only can be internal chip memory.

2.14 \$EEPROM1

*{\$EEPROM1} (*4*)*

Group: [CompilerSwitch](#) 

Divides the internal EEPROM into 2 parts.

Define EEprom1 = \$800;

2.15 \$ELSE

{\$ELSE}

Group: [CompilerSwitch](#) 

Reverses the current state. If, for example the preceding source was compiled, then the following source is treated as a comment upto "ENDIF".

2.16 \$ELSIF

{\$ELSIF label}

Group: [CompilerSwitch](#) 

To use a boolean expression consisting of DEFINES. Only the operators "AND" and "OR" are implemented and only arguments previously introduced with {\$DEFINE ..} can be used.

{\$IF ABC AND XYZ}

...

{\$ELSIF HIJK OR OPQ}

...

{\$ENDIF}

2.17 \$ELSIFDEF

`{ $ELSIFDEF label }`

Group: [CompilerSwitch](#) 

If "label" was false, the following source is compiled.

2.18 \$ENDIF

`{ $ENDIF }`
`{ $IFEND }`

Group: [CompilerSwitch](#) 

Closes a conditional block.

2.19 \$ENUMTOASM

`{ $ENUMTOASM }`

Group: [CompilerSwitch](#) 

Enumerations normally are not exported into the assembler file to save computing time with the compiler and assembler and to have a good overview of these files.

If the enum values are needed for inline assembly it is possible to force the export of the values of each enumeration into the asm-file as constants with this compiler switch.

2.20 \$HEXPATH

`{ $HEXPATH 'pathname' }`

Group: [CompilerSwitch](#) 

If "conditional compile" switches generate different firmware from the same source, it makes sense to place the generated hex-files also into different directories.

This compiler switch serves this purpose.

All involved tools which are called from the IDE like Editor, Compiler, Assembler, Programmer recognize this switch.

The argument 'pathname' must be delimited like any string in the system.

If the path or directory doesn't exist it will be created by the system.

Note:

A copy of the Flash and EEprom hex files is also stored in the current project directory so that the incircuit programmer at this point does not handle different locations for the same project.

On the other hand the programmer's project administration must be extended for the different possible projects.

2.21 \$HEXNAME

{\$HEXNAME 'filename'}

Group: [CompilerSwitch](#) 

If "conditional compile" switches generate different firmware out of the same source, it makes sense to save the generated hex-files also with different file names. This compiler switch serves this purpose. All involved tools which are called from the IDE like Editor, Compiler, Assembler, Programmer recognize this switch. The argument 'filename' must be delimited like any string in the system. The switches HexPath and HexName can also be used together.

Note:

A copy of the Flash and EEPROM hex files is also stored in the current project directory with their origin names so that the incircuit programmer at this point does not handle different locations for the same project. On the other hand the programmer's project administration must be extended with the different possible projects.

2.22 \$I

{\$I fname}

Group: [CompilerSwitch](#) 

Reads an include-file, where "filename" also can contain a path. If there is no path the working directory of the application is searched for the file. Thus sources, which are used again and again, can be included. The include-file can contain assembler or Pascal-source as well as both together. The usual conventions are still valid.

2.23 \$IDATA

{\$IDATA}

Group: [CompilerSwitch](#) 

\$IDATA assigns the area, which is found in the processor control file (xxx.dsc) within IDATA, to all following variable declarations (including **structconst**).

2.24 \$IDATA1

*{\$IDATA1} (*4*)*

Group: [CompilerSwitch](#) 

Divides the internal SRAM into 2 parts.

Define iData1 = \$800;

2.25 \$IF

*{**\$IF** cond}*

Group: [CompilerSwitch](#) 

To use a boolean expression consisting of **DEFINES**. Only the operators "**AND**" and "**OR**" are implemented and only arguments previously introduced with **{**\$DEFINE** ..}** can be used.

*{**\$IF** ABC AND XYZ}*

...

*{**\$ELSIF** HIJK OR OPQ}*

...

*{**\$ENDIF**}*

2.26 \$IFDEF

*{**\$IFDEF** label}*

Group: [CompilerSwitch](#) 

If "**label**" is true, the following source is compiled upto "**ELSE**" or "**ENDIF**". If "**label**" is false, the opposite occurs.

*{**\$IFDEF** CPUname}*

This switch makes it possible to build CPU type dependent code.

*{**\$IFDEF** Mega128}*

...

*{**\$ENDIF**}*

2.27 \$IFNDEF

*{**\$IFNDEF** name}*

Group: [CompilerSwitch](#) 

If "**label**" is false, the following source is compiled upto "**ELSE**" or "**ENDIF**". If "**label**" is true, the opposite occurs.

2.28 \$J

*{**\$J** fname}*

Group: [CompilerSwitch](#) 

Reads an include-file, where "**filename**" need not contain a path. The "**Home-Directory**" of the compiler is generally prefixed as path.

This is very advantageous in the case of procedures etc., which occur again and again and it almost replaces a linker in the unit-concept.

2.29 \$LCDNOINIT

`{ $LCDNOINIT }`

Group: [CompilerSwitch](#) 

The LCD controller is not initialized at Reset time. The application must do it with a call to the system procedure "LCDsetup".

2.30 \$LCDNOWAIT

`{ $LCDNOWAIT }`

Group: [CompilerSwitch](#) 

Disables the Busy-Polling of the display driver. Only for debug purposes!

2.31 \$MODBUS

`{ $MODBUS fname }`

Group: [CompilerSwitch](#) 

Create a ModBus Debug File, to use with ModBus Tester Tool.

2.32 \$NOADDRCHECK

`{ $NOADDRCHECK }`

Group: [CompilerSwitch](#) 

For Mega128..256. Disables the Flash Page check for the following constant definition. If the address given is not inside the uppermost flashpage the application must provide very special own access methods for an access to this constant.

const

`{ $NOADDRCHECK }`

`LookUpTab [$20000] : array [1..256] of byte = 'Name.ext'; //mega2561`

2.33 \$NOFRAME

`{ $NOFRAME }`

Group: [CompilerSwitch](#) 

Is only valid in connection with device driver procedures, which are called by the procedures. With the following device procedure, which may only have one 8-bit passing parameter, this parameter is passed within a register. A parameter frame is not made. Thus local variables are not possible. This only makes sense for fast driver routines, written in assembler.

2.34 \$NOINIT

`{ $NOINIT }`

Group: [CompilerSwitch](#) 

Switch for the \$DATA and \$IDATA area. The subsequent variables of this section up to it's end are not initialized, i.e. they are not filled with zeros.

But you should note that the standard initialization is done in a very fast loop over a single block. If you insert NoInit variables in between this process may take much longer. You should group these var definitions to avoid extensive fragmentation.

2.35 \$NOOVRCHECK

`{ $NOOVRCHECK }`

Group: [CompilerSwitch](#) 

Disables the Check for the following Variable Overlay Declaration.

2.36 \$NORAMCHECK

`{ $NORAMCHECK }`

Group: [CompilerSwitch](#) 

The following variable is not checked for a valid RAM area.

```
{ $IDATA }
{ $NORAMCHECK }
var
  Extreme[@$FFFF] : byte;
```

2.37 \$NOREGSAVE

`{ $NOREGSAVE }`

Group: [CompilerSwitch](#) 

Is only valid in connection with Interrupt procedures. With the following interrupt procedure the system does not save any of the working registers. The programmer has to do it himself. It only makes sense for service routines written in assembler.

2.38 \$NORETURNCHECK

`{ $NORETURNCHECK }`

Group: [CompilerSwitch](#) 

Is only valid in conjunction with functions. With the following function there is no error message if

there is no Return statement.

2.39 \$NOSAVE

`{ $NOSAVE }`

Group: [CompilerSwitch](#) 

Is only valid in connection with interrupt procedures. With the following interrupt procedure the system does not save the working register automatically (except the status register and 4 main working registers). The programmer has to do it himself. It only makes sense for fast service - routines, written in assembler.

2.40 \$NOSHADOW

`{ $NOSHADOW }`

Group: [CompilerSwitch](#) 

The definition, if necessary, must be before the device declaration. With non-multitask applications only those working registers are saved which are used by the interrupts within all interrupts. This saves essentially ram, rom, and run-time. This switch is overwritten by the import of tasks and processes.

2.41 \$NOWATCHDOGAUTO

`{ $NOWATCHDOGAUTO }`

Group: [CompilerSwitch](#) 

Only for special purposes. If active, no automatic watchdog retriggers are done in the delays like mDelay etc. and in system caused delays.

2.42 \$OPTIMISE

`{ $OPTIMISE }`

Group: [CompilerSwitch](#) 

Forces a compilation with the use of the Merlin Optimiser.
This switch **must** be located in the first line of the main program!

2.43 \$OPTI ALLOW_INLINE

`{ $OPTI ALLOW_INLINE }`

Group: [CompilerSwitch](#) 

Allows simple functions and procedures (such as simple setters and getters) to be placed inline. Use for debugging if necessary.

see also [\\$OPTI NO_ALLOW_INLINE](#)

2.44 \$OPTI_BETA_OFF

[{\\$OPTI_BETA_OFF}](#)

Group: [CompilerSwitch](#) 

Disables latest optimisations. Don't use this unless you have to, and then please let Merlin know of any problems via pm.

2.45 \$OPTI_CHECK_RETURN_REGS

[{\\$OPTI_CHECK_RETURN_REGS}](#)

Group: [CompilerSwitch](#) 

Should be placed after a function or procedure containing assembly lines where an additional register saving was required by the calling procedure. Particularly intended for use with assembler code to control whether the optimiser uses the .RETURNS directive to determine whether a register is needed by the calling function.

see also [\\$OPTI NO_CHECK_RETURN_REGS](#)

2.46 \$OPTI NO_ALLOW_INLINE

[{\\$OPTI NO_ALLOW_INLINE}](#)

Group: [CompilerSwitch](#) 

Disallows simple functions and procedures (such as simple setters and getters) to be placed inline. Use for debugging if necessary.

see also [\\$OPTI ALLOW_INLINE](#)

2.47 \$OPTI NO_CHECK_RETURN_REGS

[{\\$OPTI NO_CHECK_RETURN_REGS}](#)

Group: [CompilerSwitch](#) 

Should be placed before a function or procedure containing assembly lines where an additional register is required by the calling procedure. Particularly intended for use with assembler code to control whether the optimiser uses the .RETURNS directive to determine whether a register is needed by the calling function.

see also [\\$OPTI CHECK_RETURN_REGS](#)

2.48 \$OPTI NO_CSE_OPT

`{ $OPTI NO_CSE_OPT }`

Group: [CompilerSwitch](#) ⁴⁶

Disables Common sub-expression Exit points optimisation.
This is a global flag that can be anywhere in the code.
Just use for debugging if necessary.

2.49 \$OPTI_QUICK

`{ $OPTI_QUICK }`

Group: [CompilerSwitch](#) ⁴⁶

Performs a quick optimize (less compact, but a littlebit faster).

2.50 \$OPTI SMARTLINK_ONLY

`{ $OPTI SMARTLINK_ONLY }`

Group: [CompilerSwitch](#) ⁴⁶

Removes only unused functions/procedures (dead code).
All other Optimiser functions are disabled.
This switch works global, also for Units and Includes.

2.51 \$OVERLAY

`{ $OVERLAY arg [, arg2] }`

`{ $OverLay @VarName[, NoOvrCheck] }`
`{ $OverLay 0 }`

Group: [CompilerSwitch](#) ⁴⁶

To avoid the separate overlay of each variable with: `yyy[@xxx] : byte;`
To place more than one variable into the referenced var, e.g. an existing Array or buffer.
"VarName" is must be existing variable (@VarName) in the RAM area.
The optional parameter "NoOvrCheck" defines that no overflow checks must take place.

If this switch has a zero parameter "0" then this is the end of this overlay area. All variables which are defined between these two switches now get upcounting addresses starting with the address of "VarName". This means they are placed "over" the basic variable. Basically this variable at least should occupy the same or more memory as the sum of all overlaying ones. If an overflow is created it is displayed with the trailing switch. But this overflow is ignored with if the option "NoOvrCheck" is set with the leading switch.

Instead of a variable as the reference also an absolute address can be defined by
`{ $OverLay $nnnn, NoOvrCheck }`
Then the option "NoOvrCheck" is mandatory.

2.52 \$PCU

`{ $PCU }`

Group: [CompilerSwitch](#) 

The switch located in the IDE "Project/Project Options" controls all Units of the current project. If activated all Units of the project are processed by the precompiler and PCU files are built, dependant of the other "PCU copy" switches.

By the use of this compiler switch in the source area of an Unit the rebuild of a PCU out of this Unit is forced, regardless of the meaning of the global switch in the IDE. However, the "copy PCU" switches of the IDE are still valid and control the destination of this PCU.

2.53 \$PDATA

`{ $PDATA }`

Group: [CompilerSwitch](#) 

\$PDATA assigns the area, which is found in the processor control file (xxx.dsc) within PDATA, to all subsequent variable declarations. With the AVR 8515 this area is from \$20 upto \$5F. The defined variables now are progressively placed from \$20. PDATA is reserved for an IO-area, if it exists. \$PDATA mostly is accessed with special machine commands. Within the definition of variables in this area, the compiler should not allocate the address, but the programmer must/should specify the required address of every variable:

`Var Port1[$35] : byte;`

2.54 \$PHASE

`{ $PHASE nnnn }`

Group: [CompilerSwitch](#) 

Switches to a fixed **WORD**-Address in Flash and back to the standard code page.

`{ $PHASE $1E00 };` places the subsequent code at addr \$1E00 and up.
`{ $DEPHASE };` switches the address generation back to the default page.

2.55 \$Q

`{ $Q- }`

Group: [CompilerSwitch](#) 

disables the need to qualify identifiers within assembler coding.
Only for already existing older programs.

2.56 \$REUTILIZE

{ \$REUTILIZE }

XMega

serves to reuse Timer, SPI and TWI for different drivers.

Valid arguments are TIMER_C0..TIMER_F1, SPI_C..SPI_F, TWI_C..TWI_F

Group: [CompilerSwitch](#) 

2.57 \$SHOWERROR

{ \$SHOWERROR string }

Group: [CompilerSwitch](#) 

raises an error message

{ \$ShowError 'Error with ...' } raises an error message at assembly time.

2.58 \$SHOWWARNING

{ \$SHOWWARNING string }

Group: [CompilerSwitch](#) 

raises a warning

{ \$ShowWarning 'Attention: ...' } raises a warning at assembly time.

2.59 \$SL

{ \$SL+ } / { \$SL- }

temporarily disabled ! Please use Merlin Optimiser instead !

see [\\$OPTIMISE](#) and [\\$OPTI SMARTLINK ONLY](#)

Group: [CompilerSwitch](#) 

Enables or disables the Smart-Linker. Using this switch removing dead code can be enabled (+) or disabled (-). These switches can be placed anywhere and without limitation.

Please note that this switch is always set to "off" at the entry point of an unit. The same is true for the main program file.

{ \$SL ON }

Switches the default mode of the linker at program start to active.

If the switch state is "on" (remove), it can be temporarily set "off" for a specific function or procedure by using the switch *{ \$VALIDATE ProcedureName }*.

But this switch is only valid at locations where the concerned function is already known (defined) by the system.

For validating before a function/procedure declaration use *{ \$VALIDATE \$ }*.

2.60 \$TYPEDCONST

```
{ $TYPEDCONST ON }  
{ $TYPEDCONST OFF }
```

Group: [CompilerSwitch](#) 

For a better readability of programs and to avoid compiler errors.

With "ON" the compiler expects each constant declaration to also have a proper type declaration. Because of this a "0" for example becomes unambiguous a byte, word, integer or a float.

```
const bb : byte = 0;
```

The switch is default "ON".

If existing programs should not be changed and to avoid error messages the typed const feature must be disabled in the main file like this:

```
{ $TYPEDCONST OFF }  
program ProgName;
```

The "OFF" switch should not be used anymore!

2.61 \$UDATA

```
{ $UDATA }
```

Group: [CompilerSwitch](#) 

\$UDATA assigns the area, which is **Defined** in the definitions section with UserData (UserDevice), to all subsequent variable declarations. This data area resides in an external device which is not accessible through the normal CPU-addressing, e.g. a serial EEPROM. The programmer must supply a device driver. See also paragraph **Device Driver** in the *Standard Driver Manual*.

The variables always build up from the lower to the higher address. If the memory areas are changed by a new switch, then it is continued with the actual memory at the **last specified address**.

2.62 \$UNDEF

```
{ $UNDEF label }
```

Group: [CompilerSwitch](#) 

Set "label" to false

2.63 \$VALIDATE

```
{ $VALIDATE name }
```

Group: [CompilerSwitch](#) 

Constants and variables of the system and also of the application in most cases will be removed by the compiler/optimizer if they don't appear in the context. This means they must be included at least once in a statement.

This can lead to problems with inline assembler code in the Pascal sources. The assembler shows an error because the called procedure etc. is not present in the asm source. The switch {\$VALIDATE name} forces the compiler to import or implement the construct "name" and the optimizer is disabled for this item.

ATTENTION:

this option (is the only one that) is not working with the *"Merlin Optimizer"*

`{$VALIDATE $}`

Group: [CompilerSwitch](#) 

Functions and procedures can be excluded from any optimiser remove with this switch. It must be placed before a function/procedure declaration.

2.64 \$VALIDATE_ALL

`{$VALIDATE_ALL}`

Group: [CompilerSwitch](#) 

It tells the compiler to place obviously not used constants of type String, Array and Record into the program code.

The optimization for these type of constants is then disabled completely.

2.65 \$VALIDATE_OFF

`{$VALIDATE_OFF}`

Group: [CompilerSwitch](#) 

To set variable blocks as "used" so the message "possibly unused variable" becomes suppressed for this area.

`{$VALIDATE_ON}`

VAR

...

...

`{$VALIDATE_OFF}`

2.66 \$VALIDATE_ON

`{$VALIDATE_ON}`

Group: [CompilerSwitch](#) 

To set variable blocks as "used" so the message "possibly unused variable" becomes suppressed for this area.

`{$VALIDATE_ON}`

VAR

...

...
`{ $VALIDATE_OFF }`

2.67 \$VectTab

`{ $VectTab $nnnn }` // word address, e.g. `{ $VectTab $0F000 }`

Group: [CompilerSwitch](#) ⁴⁶

Forces the start of the Interrupt Vector Table to the specified address.
 For special purposes e.g. the creation of an application that is placed in the BootBlock.

see also: [\\$BootApplication](#) and [\\$CodeStart](#)

2.68 \$W

`{ $W+ }` / `{ $W- }`

Group: [CompilerSwitch](#) ⁴⁶

Switches the check of variables and procedures separately in main or units on. With obviously unused variables or procedures the compiler produces a warning, which is evaluated by the IDE. (see also optimization and [\\$WG](#) ⁶²)

Default: `{ $W- }`

2.69 \$WG

`{ $WG }`

Group: [CompilerSwitch](#) ⁴⁶

Switches the check of variables and procedures globally for all parts of the application and units on. With obviously unused variables or procedures the compiler produces a warning, which is evaluated by the IDE. (see also optimization and [\\$W](#))

Default: off

2.70 \$X

`{ $X+ }` / `{ $X- }`

Group: [CompilerSwitch](#) ⁴⁶

Disables the execution of a source code area. Useful if the program polls or waits for an external hardware.

This switch does not have any meaning for the generated Hexfile, i.e. it's not necessary to remove it.

2.71 \$XDATA

`{XDATA} {XDATA1} {XDATA2} {XDATA3} {XDATA4}`

Group: [CompilerSwitch](#) 

\$XDATA assigns the area, which is **Defined** with XDATA, to all following variable declarations (including **structconst**). XDATA is external memory, which is only available within the bigger types. If another switch of this type is found, with the following variables are processed analogously. Variables in the area \$XDATA are always accessible with longer and slower machine commands. Often the CPU uses additional Wait states.

2.72 \$XIO

`{XIO+} / {XIO-}`

Group: [CompilerSwitch](#) 

Can be used in the XDATA area. It can tell the Optimizer about a memory mapped IO-area in XDATA. The memory area between these two switches then is treated in a different way by the optimizer. For example redundant accesses are not removed.

2.73 \$ZEROLOCVARS

`{ZEROLOCVARS}`

Group: [CompilerSwitch](#) 

If active all local variables in functions and procedures are cleared to zero if a function or procedure is entered.

3 Reserved Keywords

3.1 absolute

absolute

Variable Overlay.

var

```
abc : byte;
ovr1 : byte absolute abc;           // var abc overlayed
```

Group: [Reserved Words](#) 

3.2 and

and

AND Mask.

```
a := a and $0f;
```

Group: [Reserved Words](#) 

3.3 array

array

Structured type consisting of a fixed number of components which are all of the same type.

1..4 dimensions. One dimension can have up to 61440 (\$F000) members. Total size is limited to about 60kbytes.

Types: bytes, chars, int8, booleans, words, integers, longwords, floats, Fix64, pointers, procedures.

```
type Day = (Mon,Tue,Wed,Thu,Fri,Sat,Sun);
```

var

```
WorkHour : array[1..8] of Integer;
Week     : array [ 1.. 7] of Day;
ArrEn    : array [ Day] of byte;
```

Group: [Reserved Words](#) 

3.4 asm

asm

Used to insert Assembler Statement(s).

```
ASM: PUSH _ACCA;
```

ASM;

```
LDI _ACCA, 040h
ANDI _ACCA, myProg.a;           //a = Pascal var in myProg
ENDASM;
```

Group: [Reserved Words](#) 

3.5 assign

assign

Reserved for future use.

Group: [Reserved Words](#) 

3.6 at

at

Reserved for future use.

Group: [Reserved Words](#) 

3.7 begin

begin

Starting procedure-, function-, process- or task body.

Group: [Reserved Words](#) 

3.8 bitset

bitset

8 bit or 16 bit dependent of the underlying enumeration

type

```
TBitNames      = (one, two, three, four, five, six); // enum  
TBitSet = BitSet of TbitNames; // build a bitset type
```

var

```
BitSet1          : TBitSet;      // build a bitset var  
BitSet2          : TBitSet;      // build a bitset var
```

Group: [Reserved Words](#) 

3.9 boolean

boolean

Predefined type. true = \$FF, false = \$00

var flag : boolean;

Group: [Reserved Words](#) 

3.10 break

break

Leave a FOR-, WHILE-, REPEAT-loop. See [for](#) [repeat](#) [while](#)
for *x* := 1 **to** 9 **do**

```
...
  if a := 0 then Break;
...
endfor;
```

Group: [Reserved Words](#) 

3.11 by

by

Optional in FOR-loops. Sets the increment or decrement value (1..255).

```
for x := 16 downto 0 by 2 do ...
for x := 0 to 16 by 2 do ...
```

Group: [Reserved Words](#) 

3.12 byte

byte

1. Predefined type. Whole number. 0 bis 255. Needs one byte in memory.

2. Typecast. Converts the argument into a byte.

```
Function Byte (a : type) : byte;
  b := byte (x);
```

Group: [Reserved Words](#) 

3.13 case

case

Leading statement of a branch block.

```
Case x of
0      : inc(a);
      |
1, 7   : a := 4;
      |
2..6   : x := x + a;
      | dec(x);
else   : x := 0;
EndCase;
```

Group: [Reserved Words](#) 

3.14 char

char

1. Predefined type. Character of ASCII character set. Needs one byte in memory.

```
const  cd : char      = 'D';  
        Bell : char    = ^G;      {Control G}  
        LF  : char    = #10;     {Line Feed}
```

2. Typecast. Converts the argument into a character.

```
Function Char (a : type) : char;  
ch := char (x);
```

Group: [Reserved Words](#) 

3.15 chr

chr

Reserved for future use.

Group: [Reserved Words](#) 

3.16 class

class

Reserved for future use.

Group: [Reserved Words](#) 

3.17 close

close

Reserved for future use.

Group: [Reserved Words](#) 

3.18 const

const

Begin of a constant declaration.

```
const abc : byte = 1;
```

Group: [Reserved Words](#) 

3.19 continue

continue

Interrupts the normal flow of program execution of a for-, while- or repeat-loop and transfers execution to the beginning of the loop. The following statements until EndFor, EndWhile or Until are ignored.

See [for](#) [repeat](#) [while](#)

```
for x:= 1 to 9 by 2 do
```

```
...
```

```
  if a = 0 then Continue;
```

```
...
```

```
endfor;
```

Group: [Reserved Words](#) 

3.20 define

define

Certain imports require an additional specification, which is specified using Define.

```
Define ProcClock = 4000000;           //4 Mhz
```

Group: [Reserved Words](#) 

3.21 define_fuses

define_fuses

As an option the fuse settings for the programmers can be defined in the source.

Define_Fuses

```
Override_Fuses;           // optional, always replaces fuses in ISPE
COMport = COM1;           // COM2..COM7, USB
Supply = 4.0, 200;        // programmer supplies target, 4.0Volt, 200mA
SPiclk = 1000000;         // optional SPI programming speed
NoteBook = B;             // A ... N
LockBits0 = [];
LockBits1 = [];
FuseBits0 = [CKSEL1];
FuseBits1 = [BOOTRST, BOOTSZ1, SPIEN, OCDEN];
FuseBits2 = [];
ProgMode = SPI;           // SPI, JTAG or OWD
ProgFuses = true;         // or false – program Fuse Bits (* REV4 *)
ProgLock = true;          // or false – program Lock Bits (* REV4 *)
ProgFlash = true;         // or false – program Flash (* REV4 *)
ProgEEProm = true;        // or false – program Eeprom (* REV4 *)
CalByte = 2, $3FFF;       // read/write calibration byte (* REV4 *)
AddApp = 'pathname\projectname';
```

Group: [Reserved Words](#) 

3.22 **define_usr**

define_usr

With "Define_USR" it is possible to define constants which are visible and accessible from every point of the program and also from each Unit.

Define_USR *myConst = 1;*

Group: [Reserved Words](#) 

3.23 **definefrom**

definefrom

With Units imported definitions can be made inside the units.

DefineFrom *unit1; // Definitions of Unit1*

Group: [Reserved Words](#) 

3.24 **device**

device

Processor und Hardware specification.

device = *90S2313;*

Group: [Reserved Words](#) 

3.25 **devicelock**

devicelock

Predefined type. To lock drivers in multi tsaking against other processes/tasks.

Group: [Reserved Words](#) 

3.26 **div**

div

Operator. Division of whole numbers.

Allowed operants: Type byte, int8, boolean, integer, word, longint, longword, word64, int64 or fix64.

*a:= a **div** b;*

Group: [Reserved Words](#) 

3.27 **do**

do

Starts the statement block in **for**-, **while**-, **with**-commands. See [for](#) [while](#) [with](#)

Group: [Reserved Words](#) 

3.28 downto

downto

Specifies the runtime-variable in for-loops to be decremented. See [for](#)

Group: [Reserved Words](#) 

3.29 els_if

els_if

elsif

See [elsif](#)

Group: [Reserved Words](#) 

3.30 else

else

Optional statement in IF-loop.

if a > b then a := b; else b := a; endif;

Group: [Reserved Words](#) 

3.31 elsif

elsif

els_if

Optional statement in IF-loop.

if a > b then ..; elsif b = a then ..; endif;

Group: [Reserved Words](#) 

3.32 end

end

End of a procedure-, function-, task- or process-body.

Group: [Reserved Words](#) 

3.33 end_asm

end_asm

endasm

See [endasm](#)

Group: [Reserved Words](#) 

3.34 end_case

`end_case`
`endcase`

See [endcase](#)

Group: [Reserved Words](#) 

3.35 end_for

`end_for`
`endfor`

See [endfor](#)

Group: [Reserved Words](#) 

3.36 end_if

`end_if`
`endif`

See [endif](#)

Group: [Reserved Words](#) 

3.37 end_loop

`end_loop`
`endloop`

See [endloop](#)

Group: [Reserved Words](#) 

3.38 end_try

`end_try`
`endtry`

See [endtry](#)

Group: [Reserved Words](#) 

3.39 end_while

`end_while`
`endwhile`

See [endwhile](#)

Group: [Reserved Words](#) 

3.40 end_with

end_with
endwith

See [endwith](#)

Group: [Reserved Words](#) 

3.41 endasm

endasm
end_asm

End of an assembler text. See [asm](#)

Group: [Reserved Words](#) 

3.42 endcase

endcase
end_case

End of a CASE statement. See [case](#)

Group: [Reserved Words](#) 

3.43 endfor

endfor
end_for

End of a FOR statement. See [for](#)

Group: [Reserved Words](#) 

3.44 endif

endif
end_if

End of an IF statement. See [if](#)

Group: [Reserved Words](#) 

3.45 endloop

endloop
end_loop

End of a LOOP statement. See [loop](#)

Group: [Reserved Words](#) 

3.46 endtry

endtry
end_try

End of a TRY statement. See [try](#)

Group: [Reserved Words](#) 

3.47 endwhile

endwhile
end_while

End of a WHILE statement. See [while](#)

Group: [Reserved Words](#) 

3.48 endwith

endwith
end_with

End of a WITH statement. See [with](#)

Group: [Reserved Words](#) 

3.49 eof

eof

Reserved for future use.

Group: [Reserved Words](#) 

3.50 eoln

eoln

Reserved for future use.

Group: [Reserved Words](#) 

3.51 **except**

except

Option at the try statement. See [try](#)

Group: [Reserved Words](#) 

3.52 **exec**

exec

Reserved for future use.

Group: [Reserved Words](#) 

3.53 **exit_loop**

exit_loop

exitloop

See [exitloop](#)

Group: [Reserved Words](#) 

3.54 **exitloop**

exitloop

exit_loop

Exits the loop and continues with the next statement after EndLoop. See [loop](#)

Group: [Reserved Words](#) 

3.55 **export**

export

Reserved for future use.

Group: [Reserved Words](#) 

3.56 **false**

false

Value of a boolean constant. Pre-defined = 0.

Group: [Reserved Words](#) 

3.57 file

file

Keyword in filesystem.

Var *ff* : **file of Byte**

Group: [Reserved Words](#) 

3.58 finalization

finalization

Optional section within a unit. See [unit](#)

The statements in this block are processed when the application calls the system procedure "System_ShutDown". See [System_ShutDown](#)

Group: [Reserved Words](#) 

3.59 float

float

1. Predefined type. 32bit, 4 bytes, 6..9 digits, 10E-38..10E38

var *f* : **float**;

2. Typecast. Converts the argument to a float.

Function *Float* (*a* : *type*) : **float**;

f := *Float* (*x*);

Group: [Reserved Words](#) 

3.60 for

for

Leading statement of a program-loop.

for *x* := *a* **to** *v1* **do** {*ramp up*}

PWMport1 := *x*;

endfor;

for *x* := *x* **downto** 0 **do** {*ramp down*}

PWMport1 := *x*;

endfor;

Group: [Reserved Words](#) 

3.61 forward

forward

Forward declarations of pointers, functions, procedures and processes.

Procedure *Test1*; **Forward**;

Group: [Reserved Words](#) 

3.62 from

from

Imports a type, function or procedure from a specific source.

From *System* **Import** *LongInt, LongWord, Float*;

Group: [Reserved Words](#) 

3.63 function

function

Function declaration.

```
Function Test (var par : byte) : byte;
var loc : boolean;                                //local variable
begin
...
end;
```

Group: [Reserved Words](#) 

3.64 goto

goto

Absolute branch statement, which leads to any location (forwards and backwards) in a block.

```
begin
Label: lab1;                                //definition of the label
...
Goto Lab1;
```

Group: [Reserved Words](#) 

3.65 idle

idle

Reserved for future use.

Group: [Reserved Words](#) 

3.66 if

if

Leading statement of a branch.

if *a* > *b* **then**

...

elsif *a* = *b* **then**

...

else {*a* is less than *b*}

...

endif;

Group: [Reserved Words](#) 

3.67 implementation

implementation

1. in main

starting the program. The compiler inserts the reset-code and the initialization at this point.

2. in units

the implementation section contains variables, constants and type declarations local to the unit as well as the procedure and function implementations. See [unit](#)

Group: [Reserved Words](#) 

3.68 import

import

Import of **hardware dependant** system functions.

Group: [Reserved Words](#) 

3.69 in

in

Specifies the storage location of units. See [uses](#)

Group: [Reserved Words](#) 

3.70 inherit

inherit

Import or inheritance of an existion record in a new one.

inherit *RecordName*;

Group: [Reserved Words](#) 

3.71 inherited

inherited

Reserved for future use.

Group: [Reserved Words](#) 

3.72 initialization

initialization

Optional part of an unit. See [unit](#)

Group: [Reserved Words](#) 

3.73 int64

int64

Predefined type. 64bit, 8 Bytes, -9223372036854775808 ... 9223372036854775807

var *i64* : *int64*;

Group: [Reserved Words](#) 

3.74 int8

int8

ShortInt

1. Predefined type. Short Integer.
8bit, 1 byte, -128..127

var *i8* : *int8*;

shint : *ShortInt*;

2. Typecast. Converts the argument to a short integer.

Function *Int8* (*a* : *type*) : *int8*;

i := *Int8* (*x*);

i := *ShortInt* (*x*);

Group: [Reserved Words](#) 

3.75 integer

integer

1. Predefined type. 16bit, 2 bytes, -327678..32767

var *i* : *integer*;

2. Typecast. Converts the argument to an integer.

Function *Integer* (*a* : *type*) : *integer*;

i := *Integer* (*x*);

Group: [Reserved Words](#)

3.76 interface

interface

Declares within an unit constants, types, variables, procedures and functions which are accessible for Clients. See [unit](#)

Group: [Reserved Words](#)

3.77 interrupt

interrupt

Declaration of an interrupt procedure

In addition the I/O control registers of the CPU for the specific interrupt handling must be initialized by the application itself (see the descriptions in the controller manual) !

Interrupt *Int0*;

begin

IncSema (sema0);

end;

Group: [Reserved Words](#)

3.78 label

label

Branch destination of a goto statement. See [goto](#)

Group: [Reserved Words](#)

3.79 locked

locked

Protects global ordinal variables against concurrent accesses of interrupts, processes or tasks.

Var *i : integer, locked*;

Group: [Reserved Words](#)

3.80 longint

longint

1. Predefined type. 32bit, 4 bytes, -2147483648..2147483647

var *li : longint*;

2. Typecast. Coverts the argument to a LongInt.

Function *LongInt (a : type) : longint*;

Li := LongInt (x);

Group: [Reserved Words](#) 

3.81 longword

longword

Predefined type. 32bit, 4 Bytes, 0..4294967295

var lw : longword;

Group: [Reserved Words](#) 

3.82 loop

loop

Starts an endless loop.

begin

loop

...

endloop;

end.

Group: [Reserved Words](#) 

3.83 mod

mod

Operator. Modulo of whole numbers.

res:= a mod 5;

if "a" has a negative value also "res" becomes negative.

Consider using [Abs](#) if you need a positive result.

Group: [Reserved Words](#) 

3.84 nil

nil

Predefined constant of type pointer = \$0000.

p := nil;

Group: [Reserved Words](#) 

3.85 not

not

Operator. Inverting of variables.

Allowed operands: Type byte, int8, boolean, integer, word, longint, longword, word64, int64 or fix64.

a := not a;

Group: [Reserved Words](#) 

3.86 object

object

Reserved for future use.

Group: [Reserved Words](#) 

3.87 of

of

Definition of (if applicable: predefined) types. See [array](#)

Group: [Reserved Words](#) 

3.88 or

or

Operator. OR Mask.

a := a or \$30;

Group: [Reserved Words](#) 

3.89 override

override

Reserved for future use.

Group: [Reserved Words](#) 

3.90 pidcontrol

pidcontrol

Pseudorecord for PID controller.

var Pid1 : PIDcontrol[iLimit, dIntVal];

Group: [Reserved Words](#) 

3.91 pipe

pipe

Predefined type. FIFO memory.

var Pipe1 : Pipe[16] of byte;

Group: [Reserved Words](#) 

3.92 pointer

pointer

1. Predefined type. 16bit, 2 bytes
Address of a variable.

type *tpb : pointer to Byte;*

var *pb : tpb;*

...

pb := tpb (anyPointer);

2. Typecast. Converts the argument to a pointer.

Function *Pointer (a : type) : pointer;*

p := Pointer (x);

Group: [Reserved Words](#) 

3.93 procedure

procedure

Procedure declaration.

Procedure *Test2 (par : byte);*

begin

Group: [Reserved Words](#) 

3.94 process

process

Process declaration.

Process *ProcessName (StackSize, FrameSize : word; DataPage);*

Process *Proc1 (32, 32 : iData);* // default prio=3, autostart

Process *Proc1 (32, 32 : iData; 5);* // priority 5

Process *Proc1 (32, 32 : iData; resumed);* // default prio=3, automatic start

Process *Proc1 (32, 32 : iData; 5, suspended);* // prio= 5, no automatic start

(main process has default priority = 5).

Group: [Reserved Words](#) 

3.95 program

program

Start of the program.

Program *Test;*

Group: [Reserved Words](#) 

3.96 record

record

Structure consisting of a fixed number of components, called *fields*.

```
type    tMonths = (Jan, Feb, Mar, Apr, May, Jun, July, Aug, Sep, Oct, Nov, Dec);  
        Date = record  
            Day : byte;  
            Month : tMonths  
            Year : word;  
        end;
```

Group: [Reserved Words](#) 

3.97 repeat

repeat

Leading statement of a program loop.

```
repeat  
    inc(TCC);  
until TCC > 20;
```

Group: [Reserved Words](#) 

3.98 return

return

Break and exit within a procedure/function.

```
Procedure Test1;  
begin  
    Statement ...;  
    if (a > b) then  
        Return;  
    endif;  
    Statement ...;  
end;
```

Group: [Reserved Words](#) 

3.99 rol

rol

Operator. Rotate left. All bits are retained, but change their positions.

```
a := a rol 4;
```

Group: [Reserved Words](#) 

3.100 ror

ror

Operator. Rotate right. All bits are retained, but change their positions.

a := a ror 4;

Group: [Reserved Words](#) 

3.101 semaphore

semaphore

Predefined type. 8bit, byte. Imported by processes or tasks.

var sema : Semaphore;

Group: [Reserved Words](#) 

3.102 sendsema

sendsema

Reserved for future use.

Group: [Reserved Words](#) 

3.103 set

set

Reserved for future use.

Group: [Reserved Words](#) 

3.104 shl

shl

Operator. Shift left. Shifts all Bits in the operand left, fills the bits from right with 0.

a := a shl 5;

Group: [Reserved Words](#) 

3.105 shla

shla

Operator. Arithmetic shift left. Shifts all bits within the operand to left and fills the bits from right with '0'.

The highest value bit (sign bit) stays unchanged.

a := a shla 5;

Group: [Reserved Words](#) 

3.106 shr

shr

Operator. Shift right. Shifts all Bits in the operand right, fills the bits from left with 0.

a := **a shr** 5;

Group: [Reserved Words](#) 

3.107 shortint

ShortInt

See [int8](#)

3.108 shra

shra

Operator. Arithmetic shift right. Shifts all bits within the operand to right and fills the bits from left with '0'.

The highest value bit (sign bit) stays unchanged.

a := **a shra** 5;

Group: [Reserved Words](#) 

3.109 str

str

Reserved for future use.

Group: [Reserved Words](#) 

3.110 string

string

Predefined type. 0..255 bytes, variable or constant.

var st1 : string[8];

...

st1 := st;

Group: [Reserved Words](#) 

3.111 structconst

structconst

Structured constant.

structconst *//Constant in Rom, copied at startup into Ram*
str : st10 = 'abcde';

Group: [Reserved Words](#) 

3.112 systimer

systimer

16 bit, word. Is imported by SysTick. Variable is decremented by every system tick, if it is > 0.

var *Timer1 : SysTimer;*

Group: [Reserved Words](#) 

3.113 systimer32

systimer32

32bit, LongWord. Is imported by SysTick. Variable is decremented by every system tick, if it is > 0.

from *System* **import** *longword;*
var *Timer3 : SysTimer32;*

Group: [Reserved Words](#) 

3.114 systimer8

systimer8

8bit, Byte. Is imported by SysTick. Variable is decremented by every system tick, if it is > 0.

var *Timer2 : SysTimer8;*

Group: [Reserved Words](#) 

3.115 table

table

1 dimension. Up to 255 members. The table length is limited on power of 2

Types: bytes, chars, int8, booleans, words, integers, longwords, floats, Fix64, pointers, procedures.

var *tb1 : Table[0..15] of char;*
tb2 : Table[0..127] of word;

Group: [Reserved Words](#) 

3.116 task

task

Task declaration.

Task *Name (MemoryArea[; Priority, RunMode]);*

Task *Task1 (iData);* // default prio=5, autostart

Task *Task1 (iData, 8);* // priority 8

Task *Task1 (iData, resumed);* // default prio=5, automatic start

Task *Task1 (iData, 8, suspended);* // prio= 8, no automatic start

Group: [Reserved Words](#)

3.117 then

then

Keyword for the statement block in an IF statement. See [if](#)

Group: [Reserved Words](#)

3.118 to

to

1. used in for loops for increment of the runtime-variable. See [for](#)

2. typecasting of pointers. See [pointer](#)

Group: [Reserved Words](#)

3.119 true

true

Value of a boolean constant. Pre-defined = \$FF.

Group: [Reserved Words](#)

3.120 try

try

Begin of implementation block of exceptions. See [RaiseException](#) [GetExceptResult](#)

Try

StatementE..

StatementE..

StatementE..

except

StatementN..

StatementN..

EndTry;

Group: [Reserved Words](#) 

3.121 type

type

Begin of a type declaration.

type *tpb = pointer to byte;*
 tarr = array[2..7] of byte;

Group: [Reserved Words](#) 

3.122 unit

unit

Definition of a Unit.

unit *Unit1;*

interface

uses *{ List of additional Units }*
 { interface-section }

implementation

{ implementation-section }

initialization

{ initialization-section }

finalization

{ finalization-section }

end.

Group: [Reserved Words](#) 

3.123 until

until

Definition of the end condition in a REPEAT statement. See [repeat](#)

Group: [Reserved Words](#) 

3.124 userdevice

userdevice

Reserved for future use.

Group: [Reserved Words](#) 

3.125 uses

uses

Import of units.

uses *Hello, MyMath in 'C:\MyProg\MyMath', InitUnit;*

Group: [Reserved Words](#) 

3.126 using

using

Reserved for future use.

Group: [Reserved Words](#) 

3.127 val

val

Reserved for future use.

Group: [Reserved Words](#) 

3.128 var

var

Begin of a variable declaration.

{ \$DATA }

Var	<i>TrisA[\$85]</i>	<i>: byte;</i>	<i>//var on fixed adr</i>
	<i>Count</i>	<i>: byte;</i>	<i>//normal dclaration</i>

Group: [Reserved Words](#) 

3.129 variant

variant

Reserved for future use.

Group: [Reserved Words](#) 

3.130 while

while

Leading statement of a program loop.

```
while x < 100 do
  inc (x);
endwhile
```

Group: [Reserved Words](#)

3.131 with

with

Short access to records.

```
with Salary do
  Individual := NewEmployee;           //Salary.Individual
  Cost := StandardRates;              //Salary.Cost
endwith;
```

Group: [Reserved Words](#)

3.132 word

word

1. Predefined type. 16bit, 2 bytes, 0..65535

```
var   w :       word;
      w1[24] : word;           //word at fixed adr 24
```

2. Typecast. Converts the argument to a word

```
Function Word (a : type) : word;
  w:= word (x);
```

Group: [Reserved Words](#)

3.133 word64

word64

Predefined type. 64bit, 8 Bytes, 0..18446744073709551615

```
var w64 : word64;
```

Group: [Reserved Words](#)

3.134 xor

xor

Operator. XOR Mask.

```
a:= a xor 1;
```

Group: [Reserved Words](#) 

4 Functions + Procedures

4.1 Abs

***Function** Abs (i : integer) : integer;*

***Function** Abs (f : float|fix64) : float|fix64;*

Absolute value of an Int8, Integer, Longint, Int64, Fix64 or Float value

Group: [Maths](#)  4

4.2 AddAVFilter

***Procedure** AddAVfilter (var Filter : AVfilter; val : type);*

Replaces the oldest entry with "val". No new average value is calculated and returned.

Group: [Diverse](#)  5

4.3 Addr

***Function** Addr (identifier) : pointer; {Address of the memory location}*

Valid operands are only variables, procedures and functions.

The result is a **typed** pointer.

Group: [System](#)  7

4.4 Append

***Procedure** Append (src : string; var dst : string);*

Appends the string "src" to the end of the string "dst" (concat).

Group: [Strings](#)  6

4.5 ArcTan

***Function** ArcTan (w : float) : float;*

Returns the arctangent in radians.

Group: [Maths](#)  4

4.6 ArrToStr

***Function** ArrToStr (arr : array of char) : string;*

Converts a Null-terminated String to a Pascal String.

Group: [Strings](#)  6

4.7 AVR_CAN_BaudRate

Function *AVR_CAN_BaudRate* (*br : tCAN_baud*) : *boolean*;

Sets the baudrate. A Disable should precede and after it there must be an Enable.

Group: [AVR-CAN](#) 

4.8 AVR_CAN_Disable

Procedure *AVR_CAN_Disable*;

Disables the driver so that the application can make changes.

Group: [AVR-CAN](#) 

4.9 AVR_CAN_Enable

Procedure *AVR_CAN_Enable*;

Re-enables the driver after a disable.

Group: [AVR-CAN](#) 

4.10 AVR_CAN_GetError

Function *AVR_Can_GetError* (*box : byte*) : *boolean*;

Checks the current state of this Box for errors.

Group: [AVR-CAN](#) 

4.11 AVR_CAN_GetStatus

Function *AVR_Can_GetStatus* (*box : byte*) : *tAVR_CAN_States*;

Returns the current state of this Box.

Group: [AVR-CAN](#) 

4.12 AVR_CAN_Init

Function *AVR_CAN_Init* (*RxMOBCount : Byte*) : *boolean*;

Resets the whole CAN hardware and re-inits it.

Attention: this function was changed with Compiler Rev. 4.92.00 and must be changed if applicable.

Group: [AVR-CAN](#) 

4.13 AVR_CAN_RxErrCount

Function *AVR_CAN_RxErrCount* : *byte*;

Returns the Rx Error Count.

Group: [AVR-CAN](#) 

4.14 AVR_CAN_SetRxEMask

Function *AVR_CAN_SetRxEMask* (*box* : byte; *lwiDtag*, *lwiDmask* : longword) : boolean;
Sets the extended ID-TAG and the ID-Mask for a Rx-Box (1..14).

Group: [AVR-CAN](#) 

4.15 AVR_CAN_SetRxMask

Function *AVR_CAN_SetRxMask* (*box* : byte; *wlDtag*, *wlDmask* : word) : boolean;
Sets the standard ID-TAG and the ID-Mask for a Rx-Box (1..14).

Group: [AVR-CAN](#) 

4.16 AVR_CAN_StartMessage

Procedure *AVR_CAN_StartMessage*;
This procedure must be called in order to start the transmission.

Group: [AVR-CAN](#) 

4.17 AVR_CAN_TxErrCount

Function *AVR_CAN_TxErrCount* : byte;
Returns the Tx Error Count.

Group: [AVR-CAN](#) 

4.18 BankDevPtr

Function *BankDevPtr* (*b*:byte; *p*:pointer): pointer;
Redirects the access to the Banked Device.

Group: [Banking Port](#) 

4.19 BCDtoByte

Function *BCDtoByte* (*b* : byte) : byte;
Converts a packed BCD value to a decimal Byte.

Group: [Maths](#) 

4.20 BeepChirpH

Procedure *BeepChirpH* (*repTimes* : byte);
Generates a short declining high sound sequence (chirp).

Group: [BeepPort](#) 

4.21 BeepChirpL

Procedure *BeepChirpL* (*repTimes* : *byte*);

Generates a short declining lower sound sequence (chirp).

Group: [BeepPort](#) 

4.22 BeepClick

Procedure *BeepClick*;

Generates a short click.

Group: [BeepPort](#) 

4.23 BeepOut

Procedure *BeepOut* (*Freq* : *word*; *ticks* : *byte*);

Generates a sound with the frequency passed in parameter Frequency.

Group: [BeepPort](#) 

4.24 BeepOutErr

Procedure *BeepOutErr*;

Generates a short creaking sound.

Group: [BeepPort](#) 

4.25 BeepOutHL

Procedure *BeepOutHL*;

Generates a short declining sound sequence with 3 or 5 tones.

Group: [BeepPort](#) 

4.26 BeepOutLH

Procedure *BeepOutLH*;

Generates a short inclining sound sequence with 3 or 5 tones.

Group: [BeepPort](#) 

4.27 BeepSiren

Procedure *BeepSiren* (**const** *mode* : *byte*; *repTimes* : *byte*);

Generates one siren cycle.

Group: [BeepPort](#) 

4.28 BeepStepHL

Procedure BeepStepHL;

Generate a short declining sound sequence with 3 or 5 tones.

Group: [BeepPort](#) 

4.29 BeepStepLH

Procedure BeepStepLH;

Generate a short inclining sound sequence with 3 or 5 tones.

Group: [BeepPort](#) 

4.30 Bit

1. Predefined type. Required memory: 1bit, true..false, 0..1

```
const LedBit2 = 3;
var port6[6] : byte;
    Led2[@port6, Led2Bit] : bit;
```

Group: [Reserved Words](#) 

2. Test Bit for true/false.

```
Function Bit (a, 0 : byte) : boolean;
Function Bit (b : bit) : boolean;
```

Group: [System](#) 

4.31 BitCountOf

Function BitCountOf (x : ordinal) : byte;

Returns the count of set bits.

Group: [Diverse](#) 

4.32 BoolToStr

Function BoolToStr (bool : boolean) : string;

Function BoolToStr (bool : boolean; TrueStr, FalseStr : string) : string;

Converts a boolean value to it's corresponding string value.

Group: [Strings](#) 

4.33 Boot_Init

Procedure Boot_Init;

Provides the stack and frame as defined.

Group: [System](#) 

4.34 BootRestart

Procedure *BootRestart*;
Execute a boot operation.

Group: [System](#) 

4.35 ByteToBCD

Function *ByteToBCD* (*b* : *byte*) : *byte*;
Converts a byte value into the BCD format.

Group: [Maths](#) 

4.36 ByteToBin

Function *ByteToBin* (*value* : *byte|int8*) : *string*;
The result is the representation of the bits of the arguments by '0' or '1' in the resulting string.

Group: [Strings](#) 

4.37 ByteToHex

Function *ByteToHex* (*b* : *byte|int8*) : *string*;
Converts a numeric 8bit value into a hex-string.

Group: [Strings](#) 

4.38 ByteToStr

Function *ByteToStr* (*b* : *byte|int8*) : *string*;
Converts a numeric 8bit value into a string.

Group: [Strings](#) 

4.39 CalcChecksum

Function *CalcChecksum* (**const** *start*, *end* : *pointer*) : *word*;
Calculates the sum of all bytes in the passed EEPROM area.

Group: [Maths](#) 

4.40 CalcFlashCheck

Function *CalcFlashCheck* : *boolean*;
generates the checksum at runtime and compares it with the value stored in the Flash.

Group: [System](#) 

4.41 CalcFlashCheck_A

Function *CalcFlashCheck_A* (*count* : *word*) : *byte*;
tests the application checksum out of the boot area.

The function parameter defines the amount of bytes to be checked.
The function returns a zero if the end is not reached yet.
If finished a non-zero result is returned:
1 = check finished and ok
2 = check finished but failed.

must be imported with:

From System Import *FlashCheck_A*;

Group: [System](#) 

4.42 CalcFlashCheck_B

Function *CalcFlashCheck_B* : *boolean*;
Executes an extra Flash check over the entire boot area.

must be imported with:

From System Import *FlashCheck_B*;

Group: [System](#) 

4.43 CalcFlashCheck_S

Function *CalcFlashCheck_S*(*count* : *word*) : *byte*;

must be imported with:

From System Import *FlashCheck_S*;

sequential (partial) flash check. Must be called repeatedly until the result is <> 0.

Group: [System](#) 

4.44 ChangeDir

Function *ChangeDir* (**const** *dir* : *char*) : *boolean*; // '0'..'9'
Sets the specified directory to DefaultDirectory for all drives.

Group: [8bit FileSystem](#) 

4.45 CheckFrameValid

Function *CheckFrameValid* (*p* : *Process|Task*) : *integer*;
Checks a frame overflow.

Group: [System](#) 

4.46 CheckStackValid

Function *CheckStackValid* (*p* : *Process|Task*) : *integer*;
Checks a stack overflow.

Group: [System](#) 

4.47 ClearDeviceLock

Function *ClearDeviceLock* (*d* : *DeviceLock*) : *boolean*;
Frees the device.

Group: [MultiTasking](#) 

4.48 ClearIncrAll4

Procedure *ClearIncrAll4*;
Resets all absolute and relative internal counters.

Group: [Increment Counter 4chan](#) 

4.49 ClearIncrementVal

Procedure *ClearIncrementVal*;
Resets the absolute and relative internal counter.

Group: [Increment Counter](#) 

4.50 ClearIncrVal4

Procedure *ClearIncrVal4* (*chan* : *byte*);
Resets the absolute and relative internal counter of *chan*.

Group: [Increment Counter 4chan](#) 

4.51 ClearKeyBoard

Procedure *ClearKeyBoard*;
Clears the entire keyboard.

Group: [KeyBoard 4x4](#) 

4.52 ClearKeyBoard8

Procedure *ClearKeyBoard8*;
Clears the entire keyboard.

Group: [KeyBoard 8x8](#) 

4.53 ClearRunErr

Procedure *ClearRunErr*;
Resets a RunTimeError.

Group: [System](#) 

4.54 CompareBlock

Function *CompareBlock* (**const** *addr1*, *addr2* : *pointer*; **const** *len* : *word*) : *boolean*;
Compares two memory areas.

Group: [System](#) 

4.55 CompareIP

Function *CompareIP* (*ip1*, *ip2* : *TIPAddr*) : *boolean*;
Compares two IP-address arrays.

Group: [System](#)  [TINA TCP/IP](#)  [WizNet TCP/IP](#) 

4.56 CompareMAC

Function *CompareMAC* (*mac1*, *mac2* : *TMACAddr*) : *boolean*;
Compares two MAC-address arrays.

Group: [System](#)  [TINA TCP/IP](#)  [WizNet TCP/IP](#) 

4.57 CompareNet

Function *CompareNet* (*a1*, *a2*, *mask* : *TIPAddr*) : *boolean*;
Compares two IP addresses with the help of the subnet mask.

Group: [System](#)  [TINA TCP/IP](#)  [WizNet TCP/IP](#) 

4.58 Copy

Function *Copy* (*st* : *string*; *pos*, *count* : *byte*) : *string*;
Returns a string which is a substring from the source.

Group: [Strings](#) 

4.59 CopyBlock

Procedure *CopyBlock* (*src* : *pointer*; *dst* : *pointer*; *len* : *word*);
Copies a memory area to another.

Group: [System](#) 

4.60 Cos

Function *Cos* (*w* : *float*) : *float*;
Returns the cosine of the argument.

Group: [Maths](#) 

4.61 CosD

Function *CosD* (*w* : *float*) : *float*;
Returns the cosine of the angle *w* in degrees.

Group: [Maths](#) 

4.62 CosInt

Function *CosInt* (*angle*, *v* : *integer*) : *integer*;
Returns the Cosine of the angle multiplied by the Integer argument.

Group: [Maths](#) 

4.63 CosInt16

Function *CosInt16* (*angle* : *integer*) : *integer*; // *angle in 0.1deg*
Calculates the cosine of an angle and multiplies the result by 10000.

Group: [Maths](#) 

4.64 CPUsleep

Procedure *CPUSleep* (*sleepcmd* : *byte*);
The CPU goes to sleep, i.e. the program is stopped.

Group: [System](#) 

4.65 CRCcheck

Function *CRCcheck* (*p* : *pointer*; *count* : *word*) : *word*;
Checks a block of data with the appended CRC-word.

Group: [System](#) 

4.66 CRCstreamAdd

Function *CRCstreamAdd* (*value* : *byte*) : *word*;
Adds bytes (value) continuously to the checksum.

Group: [System](#) 

4.67 CRCstreamAddP

Function *CRCstreamAddP* (*ptr* : *pointer*; *count* : *word*) : *word*;
Adds a memory block (RAM) to the checksum.

Group: [System](#) 

4.68 CRCstreamInit

Procedure *CRCstreamInit* (*seed* : *word*);
Clears the CRC start value to 0.

Group: [System](#) 

4.69 DCFdayLightSave

Function *DCFdayLightSave* : *boolean*;
Returns the current daylight saving. With summer time the result = true.

Group: [DCF-77 Encoder](#) 

4.70 DCFfield

Function *DCFfield* : *byte*;
Represents the quality of the receiving level.

Group: [DCF-77 Encoder](#) 

4.71 DCFready

Function *DCFready* : *boolean*;
Becomes true if the DCF77 driver received a complete error free telegram.

Group: [DCF-77 Encoder](#) 

4.72 DCFupdate

Procedure *DCFupdate*; // *Callback Procedure*
If the DCF77 driver can't locate the import of the RTC,
it calls the procedure „DCFupdate" every minute.

Group: [DCF-77 Encoder](#) 

4.73 DDS10buildTab

Procedure *DDS10buildTab* (*DDStab* : byte; *dsMode* : tdsMode);
Initializes a DDS10 Lookup Tabelle.

Group: [DDS10 Synthesizer](#)¹⁰

4.74 DDS10setFrequ

Procedure *DDS10setFrequ* (*freq* : float);
Selects the current working frequency.

Group: [DDS10 Synthesizer](#)¹⁰

4.75 DDS10setTab

Procedure *DDS10setTab* (*DDStab* : byte);
Switches a runtime between existing tables.

Group: [DDS10 Synthesizer](#)¹⁰

4.76 DDS10start

Procedure *DDS10start*;
Starts synthesis.

Group: [DDS10 Synthesizer](#)¹⁰

4.77 DDS10stop

Procedure *DDS10stop*;
Stops synthesis.

Group: [DDS10 Synthesizer](#)¹⁰

4.78 Dec

Procedure *Dec* (*var v* [, *step*] : type);
Decrements a variable.

Group: [Maths](#)⁴

4.79 DeclAVfilter

Function *DeclAVfilter* (*var Filter* : AVfilter) : type;
Calculates the gradient between the oldest and the youngest entry.

Group: [Diverse](#)⁵

4.80 DecSema

Function *DecSema* (*s : semaphore*) : *boolean*;

A semaphore is decremented by one.

Group: [System](#) 

4.81 DecToLim

Function *DecToLim* (**var** *v : ordinal* [, *limit : ordinal*] ; *val : ordinal*) : *boolean*;

The argument "v" is decremented by val, provided that "v" has not reached it's natural limit.

Group: [Maths](#) 

4.82 DecToLimWrap

Function *DecToLimWrap* (**var** *value, lim, pres : type*) : *boolean*;

The variable "value" is decremented by one.

Group: [Maths](#) 

4.83 DegToRad

Function *DegToRad* (*w : float*) : *float*;

Turns an angular value, which is in degrees, into radians.

Group: [Maths](#) 

4.84 Delete

Procedure *Delete* (**var** *s : string*; *pos, count : byte*);

Deletes a number of characters within a string.

Group: [Strings](#) 

4.85 Disable_JTAGport

Procedure *Disable_JTAGport*;

Disables the JTAG port at runtime.

Group: [System](#) 

4.86 DisableInts

Procedure *DisableInts*;

Disables the global interrupt.

Group: [System](#) 

4.87 DiskFormat

Function *DiskFormat* (**const** drive : char) : boolean; // 'A'..'D'

Initializes the directory part of the drive. Already existent file will be erased.

Group: [8bit FileSystem](#) 

4.88 DiskFree

Function *DiskFree* (**const** dir : char) : word; // 'A'..'D'

Returns the unused (free) disk space in kBytes.

Group: [8bit FileSystem](#) 

4.89 DiskReset

Function *DiskReset* (**const** drive : char) : boolean; // 'A'..'D'

Resets the specified drive.

Group: [8bit FileSystem](#) 

4.90 DiskSelect

Function *DiskSelect* (**const** drive : char) : boolean; // 'A'..'D'

Selects the DefaultDrive.

Group: [8bit FileSystem](#) 

4.91 Disp14Blink

Procedure *Disp14Blink* (**const** blink : boolean);

The whole display blinks or the blinking is switched OFF.

Group: [LED14seg Display](#) 

4.92 Disp14Clear

Procedure *Disp14Clear*;

The whole display is deleted.

Group: [LED14seg Display](#) 

4.93 Disp14ClrEOL

Procedure *Disp14ClrEOL*;

The display is deleted from the current position of the cursor upto the end of line.

Group: [LED14seg Display](#) 

4.94 Disp14DigBlink

Procedure Disp14DigBlink (digit: byte; blink: boolean);
A single digit blinks or blinking is disabled.

Group: [LED14seg Display](#)¹⁸

4.95 Disp14Out

Procedure Disp14Out (const ch : char);
Write into the display buffer.

Group: [LED14seg Display](#)¹⁸

4.96 Disp14Pos

Procedure Disp14Pos (const digit : byte);
The writing cursor is positioned to a certain location.

Group: [LED14seg Display](#)¹⁸

4.97 Disp14Test

Procedure Disp14Test;
Testfunction. All segments of the display are switched ON.

Group: [LED14seg Display](#)¹⁸

4.98 Disp7Test

Procedure Disp7Test;
Test function. Switches ON all segments of the display.

Group: [LED7seg Display](#)¹⁷

4.99 DispBlink

Procedure DispBlink (true);
The whole display blinks or the blinking is switched off.

Group: [LED7seg Display](#)¹⁷

4.100 DispClear

Procedure DispClear;
The whole display is deleted.

Group: [LED7seg Display](#)¹⁷

4.101 DispCIEOL

Procedure *DispCIEOL*;

The display is deleted from the current position of the cursor upto the end of the line.

Group: [LED7seg Display](#)¹⁷

4.102 DispDigBlink

Procedure *DispDigBlink* (*x* : byte);

A single digit blinks or the blinking is switched off.

Group: [LED7seg Display](#)¹⁷

4.103 DispOut

Procedure *DispOut* (*ch* : char);

Write into the display buffer.

Group: [LED7seg Display](#)¹⁷

4.104 DispPos

Procedure *DispPos* (*x* : byte);

The writing cursor is positioned to a certain location.

Group: [LED7seg Display](#)¹⁷

4.105 EdBoolean

Function *EdBoolean* (*EdValue* : Boolean; *LeadLabel*, *PostLabel* : String[EdLabelLength];
X, *Y* : Byte; *BlinkCursor* : Boolean) : Boolean;

Edit a Boolean.

Parameters: see Group [LCD Edit](#)²⁰

4.106 EdByte

Function *EdByte* (*EdValue* : Byte; *LeadLabel*, *PostLabel* : String[EdLabelLength];
X, *Y* : Byte; *BlinkCursor* : Boolean; *VMin*, *VMax* : Byte) : Byte;

Edit a Byte.

Parameters: see Group [LCD Edit](#)²⁰

4.107 EdDate

Function *EdDate* (*EdValue* : String[EdDateLength]; *LeadLabel* : String[EdLabelLength];
X, *Y* : Byte; *BlinkCursor* : Boolean) : String[EdDateLength];

Edit a Date.

Parameters: see Group [LCD Edit](#) 

4.108 EdInteger

Function *EdInteger* (*EdValue* : Integer; *LeadLabel*, *PostLabel* : String[EdLabelLength];
X, *Y* : Byte; *BlinkCursor* : Boolean; *VMin*, *VMax* : Integer;
Decimal : Byte) : Integer;

Edit an Integer.

Parameters: see Group [LCD Edit](#) 

4.109 EdIPAddress

Function *EdIPAddress* (*EdValue* : tEDIPAddress; *LeadLabel*, *PostLabel* : String[EdLabelLength];
X, *Y* : Byte; *BlinkCursor* : Boolean;
IPMin, *IPMax* : tEdIpAddress) : tEDIPAddress;

Edit an IP address.

Parameters: see Group [LCD Edit](#) 

4.110 EdList

Function *EdList* (*EdValue* : Pointer; *Location* : tEdArrayLocation;
LeadLabel, *PostLabel* : String[EdLabelLength]; *X*, *Y* : Byte;
BlinkCursor : Boolean; *StrLen*, *Count*, *Default* : Byte) : Byte;

Display string lists, step up and down in the list/array. On exit the selected index is returned. Please note that the array must start with the index 0.

EdValue	is a pointer to an array of strings (see the demo program)
Location	defines the source (RAM, ROM, EEPROM) of the array
StrLen	defines the maximum length of a string in the array (be careful here. Pointers are used)
Count	defines the entry count of the array (be careful here. Pointers are used)

Group: [LCD Edit](#) 

4.111 EdLongInt

Function *EdLongInt* (*EdValue* : LongInt; *LeadLabel*, *PostLabel* : String[EdLabelLength];
X, *Y* : Byte; *BlinkCursor* : Boolean; *VMin*, *VMax* : LongInt;
Decimal : Byte) : LongInt;

Edit a LongInt.

Parameters: see Group [LCD Edit](#) 

4.112 EdLongWord

Function *EdLongWord* (*EdValue* : LongWord; *LeadLabel*, *PostLabel* : String[EdLabelLength];
X, *Y* : Byte; *BlinkCursor* : Boolean; *VMin*, *VMax* : LongWord;
Decimal : Byte) : LongWord;

Edit a LongWord.

Parameters: see Group [LCD Edit](#) 

4.113 EdString

Function *EdString* (*EdValue* : String[EdStringLength];
LeadLabel, *PostLabel* String[EdLabelLength];
X, *Y* : Byte; *BlinkCursor* : Boolean; *MaxLen* : Byte;
MinChar, *MaxChar* : Byte;
Repeater : Boolean) : String[EdStringLength];

Edit a String.

MaxLen defines the maximum string length which can be edited

Standard Parameters: see Group [LCD Edit](#) 

4.114 EdTime

Function *EdTime* (*EdValue* : String[EdTimeLength]; *LeadLabel* : String[EdLabelLength];
X, *Y* : Byte; *BlinkCursor* : Boolean;
EditSeconds : Boolean) : String[EdTimeLength];

Edit a Time.

EditSeconds defines whether the seconds can be edited.

Standard Parameters: see Group [LCD Edit](#) 

4.115 EdWord

Function *EdWord* (*EdValue* : Word; *LeadLabel*, *PostLabel* : String[EdLabelLength]; *X*, *Y* : Byte;
BlinkCursor : Boolean; *VMin*, *VMax* : Word; *Decimal* : Byte) : Word;

Edit a Word.

Parameters: see Group [LCD Edit](#) 

4.116 EEPromPtr

Function *EEPromPtr* (*p:pointer*): pointer;
Turns the pointer to an EEProm pointer.

Group: [System](#) 

4.117 Enable_JTAGport

Procedure Enable_JTAGport;

Enables the JTAG port at runtime.

Group: [System](#)⁷

4.118 EnableInts

Procedure EnableInts;

Enables the global interrupt.

Xmega

extended with a parameter which defines the enabled/disabled Interrupt Levels.

Procedure EnableInts (level : byte);

Group: [System](#)⁷

4.119 EnableIntsX

Procedure EnableIntsX;

Xmega

enables the global Xmega Interrupt without changing the Levels.

Group: [System](#)⁷

4.120 EnablePWM

procedure EnablePWM_C0A (ena : boolean);

procedure EnablePWM_C0B (ena : boolean);

procedure EnablePWM_C0C (ena : boolean);

...

only XMega

enables or disables the corresponding PWM Port.

Group: [PWM Port](#)²⁶

4.121 EndOfFile

Function EndOfFile (const f : file) : boolean;

Checks whether the file pointer reached the file end.

Group: [8bit FileSystem](#)¹¹

4.122 Even

Function Even (x : type) : boolean;

Tests the value for even.

Group: [Maths](#)⁴

4.123 Excl

Procedure *Excl* (*v, b : byte*);
Resets a bit in a byte.

Procedure *Excl* (*SrcDest : BitSet; op : BitSet*);
Resets a bit in a BitSet.

Group: [System](#) 

4.124 Exp

Function *Exp* (*x : float*) : *float*;
Returns the power of x.

Group: [Maths](#) 

4.125 ExtractFileExt

Function *ExtractFileExt* (*FName : string*) : *string*;
Returns the extension part of a file name.

Group: [Strings](#) 

4.126 ExtractFileName

Function *ExtractFileName* (*FName : string*) : *string*;
Returns the name part of a file name.

Group: [Strings](#) 

4.127 ExtractFilePath

Function *ExtractFilePath* (*FName : string*) : *string*;
Returns the path part of a file name.

Group: [Strings](#) 

4.128 F16_BlockRandomWrite

Function *F16_BlockRandomWrite* (*f : File; pt : pointer; Count : word; var res : Word*) : *boolean*;
Writes a datablock into a file.

Group: [FAT16 FileSystem](#) 

4.129 F16_BlockRead

Function *F16_BlockRead* (*f : File; pt : pointer; count : word; var res : word*) : *boolean*;
Reads a datablock from a file.

Group: [FAT16 FileSystem](#) 

4.130 F16_BlockWrite

Function *F16_BlockWrite* (*f* : File; *pt* : pointer; *count*: word; **var** *res*: word) : boolean;
Writes a datablock into a file.

Group: [FAT16 FileSystem](#) 

4.131 F16_ChangeDir

Function *F16_ChangeDir* (*path* : TPathStr) : boolean;
Presets the DefaultDirectory/Path for all file operations.

Group: [FAT16 FileSystem](#) 

4.132 F16_CheckDisk

Function *F16_CheckDisk* : boolean;
Checks the state of the medium in the drive.

Group: [FAT16 FileSystem](#) 

4.133 F16_CheckHandle

Function *F16_CheckHandle* (*f* : File) : tFileAccess;
Checks a FileHandle for its validity.

Group: [FAT16 FileSystem](#) 

4.134 F16_CreateDir

Function *F16_CreateDir* (*path* : TPathStr; *DirName* : TFileName; *aTime*, *aDate* : word): boolean;
Creates a new directory.

Group: [FAT16 FileSystem](#) 

4.135 F16_DateToStr

Function *F16_DateToStr* (*FileDate* : word) : tF16DateStr;
Converts a DOS FileDate into a string. Format = "yy.mm.dd".

Group: [FAT16 FileSystem](#) 

4.136 F16_DirGetDate

Function *F16_DirGetDate* (*path* : TPathStr; *FDName* : TFileName; **var** *aTime*, *aDate* : word) : boolean;
Returns the creation date of the directory.

Group: [FAT16 FileSystem](#) 

4.137 F16_DiskInit

Function *F16_DiskInit* : *boolean*;
Initializes the driver hardware.

Group: [FAT16 FileSystem](#) 

4.138 F16_DiskReset

Function *F16_DiskReset* : *boolean*;
Initializes all internal buffers.

Group: [FAT16 FileSystem](#) 

4.139 F16_EndOfFile

Function *F16_EndOfFile* (*f* : *File*) : *boolean*;
Returns a true if the end of the file is reached with a read access.

Group: [FAT16 FileSystem](#) 

4.140 F16_FileAppend

Function *F16_FileAppend* (*f* : *File*) : *boolean*;
Opens an existing file for write accesses.

Group: [FAT16 FileSystem](#) 

4.141 F16_FileAssign

Function *F16_FileAssign* (**var** *f* : *File*; *path* : *TPathStr*; *fn* : *TFileName*) : *boolean*;
Builds the FileHandle which must be used with all further operations.

Group: [FAT16 FileSystem](#) 

4.142 F16_FileClose

Function *F16_FileClose* (**var** *f* : *File*) : *boolean*;
Releases a FileHandle.

Group: [FAT16 FileSystem](#) 

4.143 F16_FileCopy

Function *F16_FileCopy* (*srcPath* : *TPathStr*; *srcFn* : *TfileName*;
 dstPath : *TPathStr*; *dstFn* : *TFileName*) : *boolean*;
Copies a file into another directory/name.

Group: [FAT16 FileSystem](#) 

4.144 F16_FileCreate

Function *F16_FileCreate* (*Path* : *TPathStr*; *FName* : *TFileName*; *aAttr* : *tFAttr*;
aTime, *aDate* : *Word*; *Size* : *LongWord*) : *boolean*;

Creates a new file and also defines the file size of it.

Group: [FAT16 FileSystem](#) 

4.145 F16_FileDelete

Function *F16_FileDelete* (*path* : *TPathStr*; *fn* : *TFileName*) : *boolean*;

Deletes an existing file.

Group: [FAT16 FileSystem](#) 

4.146 F16_FileExist

Function *F16_FileExist* (*path* : *TPathStr*; *fn* : *TFileName*; *attr* : *tFAttr*) : *boolean*;

Checks the existence of a file.

Group: [FAT16 FileSystem](#) 

4.147 F16_FileGetAttr

Function *F16_FileGetAttr* (*path* : *TPathStr*; *fn* : *TFileName*; **var** *attr* : *tFAttr*) : *boolean*;

Returns the attributes of an existing file.

Group: [FAT16 FileSystem](#) 

4.148 F16_FileGetDate

Function *F16_FileGetDate* (*path* : *TPathStr*; *fn* : *TFileName*; **var** *aTime*, *aDate* : *word*) : *boolean*;

Returns the filedate and time of an existing file.

Group: [FAT16 FileSystem](#) 

4.149 F16_FilePos

Function *F16_FilePos* (*f* : *File*) : *longword*;

Returns the current read/write pointer of a file.

Group: [FAT16 FileSystem](#) 

4.150 F16_FileRename

Function *F16_FileRename* (*path* : *TPathStr*; *fn*, *fnNew* : *TFileName*) : *boolean*;

Renames an existing file.

Group: [FAT16 FileSystem](#) 

4.151 F16_FileReset

Function *F16_FileReset* (*f* : *File*) : *boolean*;
Opens an existing file for reading.

Group: [FAT16 FileSystem](#) 

4.152 F16_FileRewrite

Function *F16_FileRewrite* (*f* : *File*; *attr* : *tFAttr*; *aTime*, *aDate* : *word*) : *boolean*;
Opens a file for write accesses.

Group: [FAT16 FileSystem](#) 

4.153 F16_FileSeek

Function *F16_FileSeek* (*f* : *File*; *p* : *longword*) : *longword*;
Moves the read/write pointer.

Group: [FAT16 FileSystem](#) 

4.154 F16_FileSetAttr

Function *F16_FileSetAttr* (*path* : *TPathStr*; *fn* : *TFileName*; *attr* : *tFAttr*) : *boolean*;
Changes the attributes of an existing file.

Group: [FAT16 FileSystem](#) 

4.155 F16_FileSetDate

Function *F16_FileSetDate* (*path* : *TPathStr*; *fn* : *TFileName*; *aTime*, *aDate* : *word*) : *boolean*;
Changes the filedate and time of an existing file.

Group: [FAT16 FileSystem](#) 

4.156 F16_FileSize

Function *F16_FileSize* (*path* : *TPathStr*; *fn* : *TFileName*; **var** *size* : *longword*) : *boolean*;
Calculates the file size (bytes) of an existing file.

Group: [FAT16 FileSystem](#) 

4.157 F16_FileSizeH

Function *F16_FileSizeH* (*f* : *File*) : *longword*;
Returns the current filesize in records.

Group: [FAT16 FileSystem](#) 

4.158 F16_FindFirst

Function *F16_FindFirst* (path : TPathStr; fn : TFileName; attr : tFAttr; **var** sr : TSearchRec) : boolean;

Opens the search for certain files.

Group: [FAT16 FileSystem](#) 

4.159 F16_FindNext

Function *F16_FindNext* (**var** sr : TSearchRec) : boolean;

Continue a search after open it with the function FindFirst.

Group: [FAT16 FileSystem](#) 

4.160 F16_GetCurDir

Function *F16_GetCurDir* : TPathStr;

Returns the current DefaultDirectory/Path.

Group: [FAT16 FileSystem](#) 

4.161 F16_GetDiskError

Function *F16_GetDiskError* : tDiskError;

Returns the state of the last disk operation.

Group: [FAT16 FileSystem](#) 

4.162 F16_GetDiskFree

Function *F16_GetDiskFree* : LongWord;

Returns the unused disk space (in bytes).

Group: [FAT16 FileSystem](#) 

4.163 F16_GetDiskSize

Function *F16_GetDiskSize* : longword;

Returns the capacity (in bytes) of the current media.

Group: [FAT16 FileSystem](#) 

4.164 F16_GetUsedHandles

Function *F16_GetUsedHandles* : byte;

Returns the count of FileHandles in use.

Group: [FAT16 FileSystem](#) 

4.165 F16_PathExist

Function *F16_PathExist* (*path* : *TPathStr*) : *boolean*;
Checks whether the supplied path exists.

Group: [FAT16 FileSystem](#) 

4.166 F16_PathExpand

Function *F16_PathExpand* (*path* : *TPathStr*; **var** *ExpandedPath* : *TPathStr*) : *boolean*;
Expands a relative path into an absolute one.

Group: [FAT16 FileSystem](#) 

4.167 F16_RandomWrite

Function *F16_RandomWrite* (*f*: *File*): *boolean*;
Opens a file for write accesses.

Group: [FAT16 FileSystem](#) 

4.168 F16_ReadSector

Function *F16_ReadSector* (*SectNum* : *longword*; *pt* : *Pointer*) : *Boolean*;
Reads a physical 512 byte sector.

Group: [FAT16 FileSystem](#) 

4.169 F16_RemoveDir

Function *F16_RemoveDir* (*path* : *TPathStr*; *DirName* : *TFileName*) : *boolean*;
Erases an existing directory.

Group: [FAT16 FileSystem](#) 

4.170 F16_StrToDate

Function *F16_StrToDate* (*strDate* : *tF16DateStr*) : *word*;
Converts a string into a DOS FileDate word.

Group: [FAT16 FileSystem](#) 

4.171 F16_StrToTime

Function *F16_StrToTime* (*strTime* : *tF16TimeStr*) : *word*;
Converts a string into a DOS FileTime word.

Group: [FAT16 FileSystem](#) 

4.172 F16_TimeToStr

Function *F16_TimeToStr* (*FileTime* : word) : tF16TimeStr;
Converts a DOS FileTime into a string. Format = "hh.mm".

Group: [FAT16 FileSystem](#) 

4.173 F16_WriteSector

Function *F16_WriteSector* (*SectNum* : longword; *pt* : Pointer) : Boolean;
Writes a physical 512 byte sector.

Group: [FAT16 FileSystem](#) 

4.174 FileAppend

Function *FileAppend* (**const** *f* : file; **const** *Buff*; **const** *Count*: word]) : word;
Appends records to the end of the file.

Group: [8bit FileSystem](#) 

4.175 FileChangeDir

Function *FileChangeDir* (**const** *fn* : tFName; **const** *dir* : char) : boolean;
Moves a file to another directory.

Group: [8bit FileSystem](#) 

4.176 FileClose

Function *FileClose* (**var** *f* : file) : boolean;
Closes the opened file.

Group: [8bit FileSystem](#) 

4.177 FileCreate

Function *FileCreate* (**const** *fn* : tFName) : boolean;
Creates an empty file.

Group: [8bit FileSystem](#) 

4.178 FileDelete

Function *FileDelete* (**const** *fn* : tFName) : boolean;
Deletes the file *fn*.

Group: [8bit FileSystem](#) 

4.179 FileExists

Function *FileExists* (**const** *fn* : *tFName*) : *boolean*;
Checks whether the file *fn* exists.

Group: [8bit FileSystem](#) 

4.180 FileFirst

Function *FileFirst* (**var** *st* : *tFName*; **const** *fn* : *tFName*) : *boolean*;
Opens the search for certain files.

Group: [8bit FileSystem](#) 

4.181 FileGetAttr

Function *FileGetAttr* (**const** *fn* : *tFName*) : *TFileAttributes*;
Returns the attributes of a file.

Group: [8bit FileSystem](#) 

4.182 FileHandleCheck

Function *FileHandleCheck* (**const** *f* : *file*) : *boolean*;
Checks the validity of a FileHandle variable.

Group: [8bit FileSystem](#) 

4.183 FileNext

Function *FileNext* (**var** *st* : *tFName*) : *boolean*;
Continue a search after open it with the function FindFirst.

Group: [8bit FileSystem](#) 

4.184 FileOpen

Function *FileOpen* (**var** *f* : *file*; **const** *fn* : *tFName*) : *boolean*;
Opens an existing file for read or write operations.

Group: [8bit FileSystem](#) 

4.185 FilePos

Function *FilePos* (**const** *f* : *file*) : *longword*;
Returns the current position of the read/write pointer of this file.

Group: [8bit FileSystem](#) 

4.186 FileRead

Function *FileRead* (**const** *f* : file; **var** *Buf* [; **const** *Count*: word]) : word;
Reads *count* records from the current position of the read/write pointer.

Group: [8bit FileSystem](#) 

4.187 FileRename

Function *FileRename* (**const** *fn*, *fnNew* : tFName) : boolean;
Renames a file.

Group: [8bit FileSystem](#) 

4.188 FileReset

Function *FileReset* (**const** *f* : file) : boolean;
Restores the ReadWrite pointer of this file to the file start position.

Group: [8bit FileSystem](#) 

4.189 FileRewrite

Function *FileRewrite* (**const** *f* : file) : boolean;
Closes this file, deletes it and then creates a new empty file.

Group: [8bit FileSystem](#) 

4.190 FileSeek

Function *FileSeek* (**const** *f* : file; **const** *p* : longword) : longword;
Adjust the read/write pointer.

Group: [8bit FileSystem](#) 

4.191 FileSetAttr

Function *FileSetAttr* (**const** *fn* : tFName; *attr* : TFileAttributes) : boolean;
Changes the attributes of an existing file.

Group: [8bit FileSystem](#) 

4.192 FileSize

Function *FileSize* (**const** *fn* : tFName [, *f* : fileType|type]) : longword;
Returns the current filesize of *fn* in records.

Group: [8bit FileSystem](#) 

4.193 FileSysReset

Procedure *FileSysReset*;
Closes all internal buffer.

Group: [8bit FileSystem](#) 

4.194 FileWrite

Function *FileWrite* (**const** *f* : *file*; **const** *Buf* [: **const** *Count*: *word*]) : *word*;
Writes *count* records beginning at the current position of the read/write pointer.

Group: [8bit FileSystem](#) 

4.195 FillBlock

Procedure *FillBlock* (*start*, *size* : *word*; *fill* : *byte*);
Fills a memory area with a byte or char.

Group: [System](#) 

4.196 FillRandom

Procedure *FillRandom* (*p* : *pointer*; *cnt* : *word*);
Fills a memory area with random values.

Random must be imported:
From *System* **Import** *Random*;

Group: [System](#) 

4.197 Fix64ArcCos

Function *Fix64ArcCos* (**const** *cosine*: *fix64*): *fix64*;
returns the angle in radians for given cosine of that angle.
see [Fix64ArcCosD](#)

Group: [Fix64](#) 

4.198 Fix64ArcCosD

Function *Fix64ArcCosD* (**const** *cosine*: *fix64*): *fix64*;
returns the angle in degrees for given cosine of that angle.
see [Fix64ArcCos](#)

Group: [Fix64](#) 

4.199 Fix64ArcCosh

Function *Fix64ArcCosh* (const a: fix64): fix64;

returns the angle in radians of a given hyperbolic sine of that angle.

Group: [Fix64](#) 

4.200 Fix64ArcCot

Function *Fix64ArcCot* (const cotangent: fix64): fix64;

returns the angle in radians for given cotangent of that angle.

see [Fix64ArcCotD](#)

Group: [Fix64](#) 

4.201 Fix64ArcCotD

Function *Fix64ArcCotD* (const cotangent: fix64): fix64;

returns the angle in degrees for given cotangent of that angle.

see [Fix64ArcCot](#)

Group: [Fix64](#) 

4.202 Fix64ArcCsc

Function *Fix64ArcCsc* (const cosecant: fix64): fix64;

returns the angle in radians of a given cosecant of that angle.

see [Fix64ArcCscD](#)

Group: [Fix64](#) 

4.203 Fix64ArcCscD

Function *Fix64ArcCscD* (const cosecant: fix64): fix64;

returns the angle in degrees of a given cosecant of that angle.

see [Fix64ArcCsc](#)

Group: [Fix64](#) 

4.204 Fix64ArcSec

Function *Fix64ArcSec* (const secant: fix64): fix64;

returns the angle in radians of a given secant of that angle.

see [Fix64ArcSecD](#)

Group: [Fix64](#) 

4.205 Fix64ArcSecD

Function *Fix64ArcSecD (const secant: fix64): fix64;*
returns the angle in degrees of a given secant of that angle.
see [Fix64ArcSec](#)

Group: [Fix64](#) 

4.206 Fix64ArcSin

Function *Fix64ArcSin (const sine: fix64): fix64;*
returns the angle in radians for given sine of that angle.
see [Fix64ArcSinD](#)

Group: [Fix64](#) 

4.207 Fix64ArcSinD

Function *Fix64ArcSinD (const sine: fix64): fix64;*
returns the angle in degrees for given sine of that angle.
see [Fix64ArcSin](#)

Group: [Fix64](#) 

4.208 Fix64ArcSinh

Function *Fix64ArcSinh (const a: fix64): fix64;*
returns the angle in radians of a given hyperbolic sine of that angle.

Group: [Fix64](#) 

4.209 Fix64ArcTan

Function *Fix64ArcTan (const tangent: fix64): fix64;*
returns the angle in radians for given tangent of that angle.
see [Fix64ArcTanD](#), [Fix64ArcTan2](#), [Fix64ArcTan2D](#)

Group: [Fix64](#) 

4.210 Fix64ArcTan2

Function *Fix64ArcTan2 (const y, x: fix64): fix64;*
variation of the arctangent function. For any real arguments x and y not both equal to zero, $\text{arctan2}(x,y)$ is the angle in radians between the positive x-axis of a plane and the point given by the coordinates (x,y) on it.
see [Fix64ArcTan2D](#), [Fix64ArcTan](#), [Fix64ArcTanD](#)

Group: [Fix64](#) 

4.211 Fix64ArcTan2D

Function *Fix64ArcTan2D* (const y, x: fix64): fix64;
variation of the arctangent function. For any real arguments x and y not both equal to zero, arctan2(x,y) is the angle in degrees between the positive x-axis of a plane and the point given by the coordinates (x,y) on it.
see [Fix64ArcTanD](#), [Fix64ArcTan](#), [Fix64ArcTan2](#)

Group: [Fix64](#) 

4.212 Fix64ArcTanD

Function *Fix64ArcTanD* (const tangent: fix64): fix64;
returns the angle in degrees for given tangent of that angle.
see [Fix64ArcTan2D](#), [Fix64ArcTan](#), [Fix64ArcTan2](#)

Group: [Fix64](#) 

4.213 Fix64ArcTanh

Function *Fix64ArcTanh* (const a: fix64): fix64;
returns the angle in radians of a given hyperbolic tangent of that angle.

Group: [Fix64](#) 

4.214 Fix64Cos

Function *Fix64Cos* (const radians: fix64): fix64;
returns the cosine of the argument (argument=angle in radians).
see [Fix64CosD](#)

Group: [Fix64](#) 

4.215 Fix64CosD

Function *Fix64CosD* (const degrees: fix64): fix64;
returns the cosine of the argument (argument=angle in degrees).
see [Fix64Cos](#)

Group: [Fix64](#) 

4.216 Fix64Cosh

Function *Fix64Cosh* (const a: fix64): fix64;
returns the hyperbolic cosine of a.

Group: [Fix64](#) 

4.217 Fix64Cot

Function *Fix64Cot (const radians: fix64): fix64;*

returns the cotangent of the argument (argument=angle in radians).

see [Fix64CotD](#)

Group: [Fix64](#) 

4.218 Fix64CotD

Function *Fix64CotD (const degrees: fix64): fix64;*

returns the cotangent of the argument (argument=angle in degrees).

see [Fix64Cot](#)

Group: [Fix64](#) 

4.219 Fix64Csc

Function *Fix64Csc (const radians: fix64): fix64;*

returns the cosecant of the argument (argument=angle in radians).

see [Fix64CscD](#)

Group: [Fix64](#) 

4.220 Fix64CscD

Function *Fix64CscD (const degrees: fix64): fix64;*

returns the cosecant of the argument (argument=angle in degrees).

see [Fix64Csc](#)

Group: [Fix64](#) 

4.221 Fix64DegToRad

Function *Fix64DegToRad (const degrees: fix64): fix64;*

returns the radians value of the argument (argument in degrees).

see [Fix64RadToDeg](#)

Group: [Fix64](#) 

4.222 Fix64DegToRadD

Function *Fix64DegToRadD (const degrees: fix64): fix64;*

returns the radians value of the argument (argument in degrees).

see [Fix64DegToRad](#)

Group: [Fix64](#) 

4.223 Fix64DivInt

Function *Fix64DivInt* (const a: fix64; const b: integer): fix64;
returns the quotient of a fix64 divided by an integer.
see [Fix64DivLong](#)

Group: [Fix64](#) 

4.224 Fix64DivLong

Function *Fix64DivLong* (const a: fix64; const b: longint): fix64;
returns the quotient of a fix64 divided by a long integer.
see [Fix64DivInt](#)

Group: [Fix64](#) 

4.225 Fix64Even

Function *Fix64Even* (const a: fix64): boolean;
returns *true* if the argument a is even.
see [Fix64Odd](#)

Group: [Fix64](#) 

4.226 Fix64Exp

Function *Fix64Exp* (const a: fix64): fix64;
returns "e" raised to the "a" power, "e" is the base of the natural logarithm (2.71828...).

Group: [Fix64](#) 

4.227 Fix64Integrate

Function *Fix64Integrate* (const aold, anew: fix64; const factor: byte): fix64;
Integrate a new value with an existing.
Calculation: *result* := ((aold * factor) + anew) **div** (fact + 1);
An overflow in the multiplication can happen so be careful with big numbers!

Group: [Fix64](#) 

4.228 Fix64IsPowOfTwo

Function *Fix64IsPowOfTwo* (const a: fix64): boolean;
returns *true* if "a" is a power of 2.

Group: [Fix64](#) 

4.229 Fix64Ln

Function *Fix64Ln* (const a: fix64): fix64;

returns the base e logarithm of a, "e" is the base of the natural logarithm ((2.71828...)).
Is the same as [Fix64LogN](#). See [Fix64Log](#) und [Fix64Log10](#)

Group: [Fix64](#) 

4.230 Fix64Log

Function *Fix64Log* (const a, base: fix64): fix64;

returns the base "base" logarithm of a.
see [Fix64Log10](#) und [Fix64Ln](#)

Group: [Fix64](#) 

4.231 Fix64Log10

Function *Fix64Log10* (const a: fix64): fix64;

returns the base 10 logarithm of a.
see [Fix64Log](#) und [Fix64Ln](#)

Group: [Fix64](#) 

4.232 Fix64LogN

Function *Fix64LogN* (const a: fix64): fix64;

returns the base e logarithm of a, "e" is the base of the natural logarithm ((2.71828...)).
Is the same as [Fix64Ln](#). See [Fix64Log](#) und [Fix64Log10](#). see [Fix64Log](#) und [Fix64Log10](#)

Group: [Fix64](#) 

4.233 Fix64Mod

Function *Fix64Mod* (const a, modulus: fix64): fix64;

returns a mod "modulus". Both arguments are fix64.
see: [Fix64ModInt](#)

Group: [Fix64](#) 

4.234 Fix64ModInt

Function *Fix64ModInt* (const a: fix64; modulus: integer): fix64;

returns a mod "modulus". modulus is here an integer.
see: [Fix64Mod](#)

Group: [Fix64](#) 

4.235 Fix64MullInt

Function *Fix64MullInt(const a: fix64; const b: integer): fix64;*
returns the product of a fix64 and an integer.
see: [Fix64MulLong](#)

Group: [Fix64](#) 

4.236 Fix64MulLong

Function *Fix64MulLong(const a: fix64; const b: longint): fix64;*
returns the product of a fix64 and a long integer.
see: [Fix64MullInt](#)

Group: [Fix64](#) 

4.237 Fix64Odd

Function *Fix64Odd(const a: fix64): boolean;*
returns *true* if the argument a is odd.
see [Fix64Even](#)

Group: [Fix64](#) 

4.238 Fix64Power

Function *Fix64Power (const base, exponent: fix64): fix64;*
returns the power $base \wedge exponent$. "base" and "exponent" are fix64 types.
see: [Fix64PowerInt](#)

Group: [Fix64](#) 

4.239 Fix64PowerInt

Function *Fix64PowerInt (const base: fix64; const exponent: integer): fix64;*
returns the power $base \wedge exponent$. "base" is a fix64 type, "exponent" is an integer type.
see: [Fix64Power](#)

Group: [Fix64](#) 

4.240 Fix64Quadrant

Function *Fix64Quadrant (const radians: fix64): byte;*
returns the quadrant (1..4) of an angle in radians.

Group: [Fix64](#) 

4.241 Fix64RadToDeg

Function *Fix64RadToDeg (const radians: fix64): fix64;*
returns the degrees value of the argument (argument in radians).
see [Fix64DegToRad](#)

Group: [Fix64](#) 

4.242 Fix64Sec

Function *Fix64Sec (const radians: fix64): fix64;*
returns the secant of the argument (argument=angle in radians).

Group: [Fix64](#) 

4.243 Fix64SecD

Function *Fix64SecD (const degrees: fix64): fix64;*
returns the secant of the argument (argument=angle in degrees).
see [Fix64Sec](#)

Group: [Fix64](#) 

4.244 Fix64Sin

Function *Fix64Sin (const radians: fix64): fix64;*
returns the sine of the argument (argument=angle in radians).

Group: [Fix64](#) 

4.245 Fix64SinD

Function *Fix64SinD (const degrees: fix64): fix64;*
returns the sine of the argument (argument=angle in degrees).
see [Fix64Sin](#)

Group: [Fix64](#) 

4.246 Fix64Sinh

Function *Fix64Sinh (const a: fix64): fix64;*
returns the hyperbolic sine of a.

Group: [Fix64](#) 

4.247 Fix64Sqrt

Function *Fix64Sqrt (f : fix64) : fix64;*
Returns the square root of a fix64-argument.

Note:
Fix64Sqrt needs 64 Bytes in Frame!

Info:

Fix64Sqrt needs 2.2msec @16MHz, precision: 9 frac digits
(only in AVRco Profi version available)

[Sqrt](#) (fix64) needs 600usec @16MHz, precision: 5 frac digits

Group: [Fix64](#) 

4.248 Fix64Tan

Function *Fix64Tan* (const radians: fix64): fix64;

returns the tangent of the argument (argument=angle in radians).

see [Fix64TanD](#)

Group: [Fix64](#) 

4.249 Fix64TanD

Function *Fix64TanD* (const degrees: fix64): fix64;

returns the tangent of the argument (argument=angle in degrees).

see [Fix64Tan](#)

Group: [Fix64](#) 

4.250 Fix64Tanh

Function *Fix64Tanh* (const a: fix64): fix64;

returns the hyperbolic tangent of a.

Group: [Fix64](#) 

4.251 Fix64ToFloat

Function *Fix64ToFloat* (const a: fix64): float;

type cast: converts a fix64 to a float.

Group: [Fix64](#) 

4.252 Fix64ToHex

Function *Fix64ToHex* (const f: Fix64): string;

type cast: converts a Fix64 value into a hex-string.

Groups: [Strings](#) , [Fix64](#) 

4.253 Fix64ToInt

Function *Fix64ToInt* (const a: fix64): integer;

type cast: converts a fix64 to an integer.

Group: [Fix64](#) 

4.254 Fix64ToLongInt

Function *Fix64ToLongInt* (const a: fix64): longint;
type cast: converts a fix64 to a long integer.

Group: [Fix64](#)

4.255 Fix64ToStr

Function *Fix64ToStr*(f : Fix64[:int : byte[:frac : char] [:space : char]]) : string;
Converts a Fix64 value into a string. The optional parameters int, frac and space format the string.

Groups: [Strings](#), [Fix64](#)

4.256 Fix64ValueInTolerance

Function *Fix64ValueInTolerance* (const a, aref, atol: fix64): boolean;
compares the content of "a" to the limit "vmin" which is build from (aref - atol) and "vmax", build from (aref + atol). If one of the two limits is exceeded the result returns false. Otherwise it returns true.
see [Fix64ValueInToleranceP](#)

Group: [Fix64](#)

4.257 Fix64ValueInToleranceP

Function *Fix64ValueInToleranceP* (const a, aref, atol: fix64): boolean;
compares the content of "a" with the limit "vmin" which is build from (aref - (aref div 100) * atol) and "vmax", build from (aref + (aref div 100) * tol). If one of the two limits is exceeded the result is false. Otherwise it is a true.
The value "atol" must be in the range of 0..100 because it is used as "percentage".
This function is the same as the function [Fix64ValueInTolerance](#) except that here the tolerance is not absolute but relative in percent.

Group: [Fix64](#)

4.258 FlashClearPage

Procedure *FlashClearPage*;
Erases the temporary buffer of the CPU with \$FF.

Group: [Flash Writer](#)

4.259 FlashCopyF2R

Procedure *FlashCopyF2R* (p : pointer);
Copies the flash page where FLASH_ADDR points to into the buffer in RAM.

Group: [Flash Writer](#)

4.260 FlashCopyR2F

Procedure *FlashCopyR2F* (*p : pointer*);

Copies flash page previously saved with *FlashCopyF2R* into the temporary buffer of the CPU.

Group: [Flash Writer](#) 

4.261 FlashDownloader

Procedure *FlashDownloader*;

Calls the Download Monitor.

Group: [Diverse](#) 

4.262 FlashErasePage

Procedure *FlashErasePage*;

Erases the page in the Flash where *FLASH_ADDR* points to.

Group: [Flash Writer](#) 

4.263 FlashInitPage

Procedure *FlashInitPage* (**const** *addr : LongWord*);

Must be called each time before a new page is opened.

Group: [Flash Writer](#) 

4.264 FlashLoaderExit

Procedure *FlashLoaderExit*;

Callback function of the Flash Downloader.

Group: [Diverse](#) 

4.265 FlashLoaderInit

Procedure *FlashLoaderInit*;

Callback function of the Flash Downloader.

Group: [Diverse](#) 

4.266 FlashLoaderRecv

Procedure *FlashLoaderRecv*;

Callback function of the Flash Downloader.

Group: [Diverse](#) 

4.267 FlashLoaderTransm

Procedure *FlashLoaderTransm*;

Callback function of the Flash Downloader.

Group: [Diverse](#) 

4.268 FlashProgPage

Procedure *FlashProgPage*;

The temporary buffer of the CPU will be programmed into the Flash memory.

Group: [Flash Writer](#) 

4.269 FlashPtr

Function *FlashPtr* (*p:pointer*): *pointer*;

Redirects the pointer into Flash.

Group: [System](#) 

4.270 FlashReadFuses

Function *FlashReadFuses* (*FuseGroup : byte*) : *byte*;

The Fusebits of an AVR can be read back at runtime.

Group: [Flash Writer](#) 

4.271 FlashReadPage

Procedure *FlashReadPage*;

Reads the Flashpage, where *FLASH_ADDR* points to, from ROM.

Group: [Flash Writer](#) 

4.272 FlashWriteFuses

Procedure *FlashWriteFuses* (*FuseGroup, fsBits : byte*);

The Fusebits of an AVR can be written at runtime.

Group: [Flash Writer](#) 

4.273 FlashWritePage

Procedure *FlashWritePage* (**const** *parm : word*);

Writes a word into the temporary buffer of the CPU.

Group: [Flash Writer](#) 

4.274 FloatAsLong

Function *FloatAsLong* (*f : float*) : *longword*;
Converts the argument into a LongWord.
There is **no type conversion**!

Group: [Diverse](#)  5

4.275 FloatToFix64

Function *FloatToFix64* (*const a: float*): *fix64*;
Type Cast: konvertiert einen Float in einen Fix64.

Group: [Fix64](#)  4

4.276 FloatToStr

Function *FloatToStr* (*f : float*) : *string*;
Converts a Floating Point value into a string.

Group: [Strings](#)  6

4.277 FlushBuffer

Procedure *FlushBuffer* (*Buffer : tBuffer*);

```
FlushBuffer (RxBuffer1);  
FlushBuffer (RxBuffer2);  
...  
FlushBuffer (TxBuffer1);  
FlushBuffer (TxBuffer2);  
...
```

XMega:

```
FlushBuffer (RxBufferC0);  
FlushBuffer (RxBufferC1);  
FlushBuffer (RxBufferD0);  
...  
FlushBuffer (TxBufferC0);  
FlushBuffer (TxBufferC1);  
FlushBuffer (TxBufferD0);  
...
```

Clears a buffer of the serial interfaces.

Group: [SerPorts](#)  29

4.278 Frac

Function *Frac* (*f* : float|fix64) : float|fix64;
Returns the fractional part of the argument.

Group: [Maths](#) 

4.279 FreeMem

Function *FreeMem* (*var ptr* : pointer) : boolean;
Frees Heap memory.

Group: [Diverse](#) 

4.280 FreqCountRestart

Procedure *FreqCountRestart*;
Resets the frequency counter.

Group: [Frequency Counter/Timer](#) 

4.281 FreqCountRestart2

Procedure *FreqCountRestart2*;
Resets the frequency counter.

Group: [Frequency Counter/Timer](#) 

4.282 gClearPixel

Procedure *gClearPixel* (*Px, Py* : integer);
Clears a pixel at the position „Px,Py" in the current ViewPort.

Group: [LCD Graphic](#) 

4.283 gClearView

Procedure *gClearView* (*ClearMode* : TWriteMode);
Erases the current ViewPort.

Group: [LCD Graphic](#) 

4.284 gClrScr

Procedure *gClrScr* (*pattern* : byte);
Clears the whole display with the byte „pattern".

Group: [LCD Graphic](#) 

4.285 gDispRefresh

Procedure *gDispRefresh*;

Writes the whole internal refresh buffer into the display.

Group: [LCD Graphic](#) 

4.286 gDrawBitMap

Procedure *gDrawBitMap* (*Xs, Ys : integer; source : pointer; DrawMode : TWriteMode*);

Copies a BitMap from ROM/RAM into the current ViewPort.

Group: [LCD Graphic](#) 

4.287 gDrawBitMapN

Procedure *gDrawBitMapN* (*Xs, Ys : integer; source : pointer; DrawMode : TWriteMode*);

Copies a BitMap from ROM/RAM into the current ViewPort.

Group: [LCD Graphic](#) 

4.288 gDrawCircle

Procedure *gDrawCircle* (*Xc, Yc, R : integer; pattern : byte*);

Draws a circle into the current ViewPort.

Group: [LCD Graphic](#) 

4.289 gDrawLine

Procedure *gDrawLine* (*Xs, Ys, Xe, Ye : integer; pattern : byte*);

Draws a line into the current ViewPort.

Group: [LCD Graphic](#) 

4.290 gDrawLineTo

Procedure *gDrawLineTo* (*Xd, Yd : integer; pattern : byte*);

Draws a line into the current ViewPort.

Group: [LCD Graphic](#) 

4.291 gDrawLineToRel

Procedure *gDrawLineToRel* (*Xr, Yr : integer; pattern : byte*);

Draws a line into the current ViewPort.

Group: [LCD Graphic](#) 

4.292 gDrawRect

Procedure *gDrawRect* (*Xs, Ys, Xe, Ye* : integer; *pattern* : byte);

Draws a rectangle into the current ViewPort.

Group: [LCD Graphic](#) 

4.293 gDrawString

Procedure *gDrawString* (*X,Y*: integer; *zx,zy*: byte; *rot*: TTxtRotate; *str*: TGraphString);

Draws the string „str“ at the logical ViewPort position „X,Y“.

Group: [LCD Graphic](#) 

4.294 gDrawStringRel

Procedure *gDrawStringRel* (*zx, zy* : byte; *rot* : TTxtRotate; *str* : TGraphString);

Draws the string „str“ at the logical ViewPort position „X,Y“.

Group: [LCD Graphic](#) 

4.295 GetADC

Function *GetADC* (*chan* : byte) : word;

Returns data of the ADC. Function to indirectly read the ADC-channels.

Attention: AVRco counts the channels 1, 2, 3, ... (corresponding to controller count 0, 1, 2, ...)

or

Function *GetADC* : word;

used with only one active channel

Group: [ADC](#) 

4.296 GetAVfilter

Function *GetAVfilter* (*var Filter* : AVfilter) : type;

Computes the current averages without changing the content of the filter.

Group: [Diverse](#) 

4.297 GetBankNum

Function *BankNum* (bankedVar) : byte;

Returns the bank number of the banked variable "bankedVar".

Group: [Banking Port](#) 

4.298 GetCurDir

Function *GetCurDir* : char; // '0'..'9'

Returns the current DefaultDirectory.

Group: [8bit FileSystem](#) 

4.299 GetCurDisk

Function *GetCurDisk* : char; // 'A'..'D'

Returns the current DefaultDrive.

Group: [8bit FileSystem](#) 

4.300 GetCurProcess

Function *GetCurProcess* : byte;

The process ID of the current process/task is interrogated.

Group: [MultiTasking](#) 

4.301 GetExceptResult

Function *GetExceptResult* : byte;

Parameter of the Exception. See [try](#)

Group: [System](#) 

4.302 GetFrameFree

Function *GetFrameFree* (*p* : Process) : word;

Get the Frame usage of processes at runtime.

Group: [System](#) 

4.303 GetFreqCounter

Function *GetFreqCounter*: word;

Returns the result of the last frequency measure cycle.

Group: [Frequency Counter/Timer](#) 

4.304 GetFreqCounter2

Function *GetFreqCounter2*: word;

Returns the result of the last frequency measure cycle.

Group: [Frequency Counter/Timer](#) 

4.305 GetFreqCounter2L

Function *GetFreqCounter2L*: *longword*;

Returns the result of the last frequency measure cycle.

Group: [Frequency Counter/Timer](#)¹²

4.306 GetFreqCounterL

Function *GetFreqCounterL*: *longword*;

Returns the result of the last frequency measure cycle.

Group: [Frequency Counter/Timer](#)¹²

4.307 GetFreqCountMode

Function *GetFreqCountMode* : *tFreqCountMode*;

Returns the current mode of the counter.

Group: [Frequency Counter/Timer](#)¹²

4.308 GetFreqCountMode2

Function *GetFreqCountMode2* : *tFreqCountMode*;

Returns the current mode of the counter.

Group: [Frequency Counter/Timer](#)¹²

4.309 GetFreqCountOvrFlow

Function *GetFreqCountOvrFlow*: *byte*;

Returns the overflow, if any, of the last conversion cycle.

Group: [Frequency Counter/Timer](#)¹²

4.310 GetFreqCountOvrFlow2

Function *GetFreqCountOvrFlow2*: *byte*;

Returns the overflow, if any, of the last conversion cycle.

Group: [Frequency Counter/Timer](#)¹²

4.311 GetIncrementRel

Function *GetIncrementRel* : *integer* [*longint*];

Returns the current internal relative counter value.

Group: [Increment Counter](#)¹²

4.312 GetIncrementVal

Function *GetIncrementVal* : integer [longint];
Returns the current internal counter value.

Group: [Increment Counter](#)^[12]

4.313 GetIncrRel4

Function *GetIncrRel4* (chan : byte) : integer [longint];
Returns the current internal relative counter value of *chan* .

Group: [Increment Counter 4chan](#)^[12]

4.314 GetIncrVal4

Function *GetIncrVal4* (chan : byte) : integer [longint];
Returns the current internal counter value of *chan*.

Group: [Increment Counter 4chan](#)^[12]

4.315 GetKey

Function *GetKey* : Keys;
The first active key found is returned.

Group: [KeyBoard 4x4](#)^[16]

4.316 GetKey8

Function *GetKey8* : Keys;
The first active key found is returned.

Group: [KeyBoard 8x8](#)^[17]

4.317 GetKeyRaised

Function *GetKeyRaised* : Keys;
The first active and latched key found is returned.

Group: [KeyBoard 4x4](#)^[16]

4.318 GetKeyRaised8

Function *GetKeyRaised8* : Keys;
The first active and latched key found is returned.

Group: [KeyBoard 8x8](#)^[17]

4.319 GetLargestBlock

Function *GetLargestBlock*: *word*;

Returns the value of the largest one-piece unfragmented memory block.

Group: [Diverse](#) 

4.320 GetMem

Function *GetMem* (*var ptr* : *pointer* [*const size* : *word*]) : *boolean*;

Requests memory from heap.

Group: [Diverse](#) 

4.321 GetMemAvail

Function *GetMemAvail* : *word*;

Returns the total amount of free memory.

Group: [Diverse](#) 

4.322 GetPriority

Function *GetPriority* (*prcs* : *process|task*) : *byte*;

Returns the current priority of a process/task.

Group: [MultiTasking](#) 

4.323 GetProcessID

Function *GetProcessID* (*ProcName*) : *byte*;

The process ID of a process is interrogated.

Group: [MultiTasking](#) 

4.324 GetProcessState

Function *GetProcessState* (*name* : *process|Task*) : *tProcessState*;

The state of a process or task is interrogated.

Group: [MultiTasking](#) 

4.325 GetPulseCount

Function *GetPulseCount* : *longword*;

Returns the actual internal counter value.

Group: [Pulse Counter](#) 

4.326 GetPulseCount2

Function *GetPulseCount2* : longword;
Returns the actual internal counter value.

Group: [Pulse Counter](#)^[26]

4.327 GetStackFree

Function *GetStackFree* (*p* : Process) : word;
Get the Stack usage of processes at runtime.

Group: [System](#)^[7]

4.328 GetSysTimer

Function *GetSysTimer* (*tm* : tSysTimer) : byte|word;
Returns the current value of a systimer.

Group: [System](#)^[7]

4.329 GetTable

Function *GetTable* (*t* : Table; *index* : byte) : type;
Returns a member of a LookUp-Table.

Group: [System](#)^[7]

4.330 GetTaskFrameFree

Function *GetTaskFrameFree* : word;
Get the Frame usage of tasks at runtime.
The function returns a word with the minimum count of unused bytes in the Task Frame so far.

Group: [System](#)^[7]

4.331 GetTaskStackFree

Function *GetTaskStackFree* : word;
Get the Stack usage of tasks at runtime.
The function returns a word with the minimum count of unused bytes in the Task Stack so far.

Group: [System](#)^[7]

4.332 GetTimeCounter

Function *GetTimeCounter* : word;
Returns the result of the last pulse time measure cycle.

Group: [Frequency Counter/Timer](#)^[12]

4.333 GetTimeCounter2

Function *GetTimeCounter2* : word;

Returns the result of the last pulse time measure cycle.

Group: [Frequency Counter/Timer](#)¹²

4.334 GetTimeCounterP

Function *GetTimeCounterP* (var Count1, Count2: word) : boolean;

Returns the counter values of the last pulse time measure cycle.

Count1: high, Count2: low. False at overflow.

Group: [Frequency Counter/Timer](#)¹²

4.335 GetTimeCounterP2

Function *GetTimeCounterP2* (var Count1, Count2: word) : boolean;

Returns the counter values of the last pulse time measure cycle.

Count1: high, Count2: low. False at overflow.

Group: [Frequency Counter/Timer](#)¹²

4.336 GetTWISlaveStat

Function *GetTWISlaveStat* (node : byte) : tTWINetState;

Retrieves the state of the local Rx and Tx buffers.

Group: [TWI Network](#)⁴³

4.337 GetWatchDogFlag

Function *GetWatchDogFlag* : byte;

Returns a byte which is the copy of the "MCUSR" register content.

Group: [System](#)⁷

4.338 gFillCircle

Procedure *gFillCircle* (Xc, Yc, R : integer; pattern : byte);

Fills a circle in the current ViewPort.

Group: [LCD Graphic](#)²²

4.339 gFillRect

Procedure *gFillRect* (Xs, Ys, Xe, Ye : integer; pattern : byte);

Fills a rectangle in the current ViewPort.

Group: [LCD Graphic](#)²²

4.340 gFrameView

Procedure *gFrameView* (*ViewPort*: byte);
Draws a frame around the ViewPort.

Group: [LCD Graphic](#) 

4.341 gGetCurView

Function *gGetCurView* : byte;
Returns the current ViewPort.

Group: [LCD Graphic](#) 

4.342 gGetLineColor

Function *gGetLineColor* : byte;
Returns the color of the actual ViewPort.

Color must be imported:

From *LCDGraphic* **Import** *GraphColor*;

Group: [LCD Graphic](#) 

4.343 gGetLineMode

Function *gGetLineMode* : *TWriteMode*;
Returns the current line write mode of the actual ViewPort.

Group: [LCD Graphic](#) 

4.344 gGetTextBkGnd

Function *gGetTextBkGnd* : *TTextBkGnd*;
Returns the current text background mode for the actual ViewPort.

Group: [LCD Graphic](#) 

4.345 gGetTextColor

Function *gGetTextColor* : byte;
Returns the color of the actual ViewPort.

Color must be imported:

From *LCDGraphic* **Import** *GraphColor*;

Group: [LCD Graphic](#) 

4.346 gGetTextJustify

Function *gGetTextJustify* (**var** Horiz : TtxtAlHor; **var** Vert : TtxtAlVert);

Returns the current text alignments for the actual ViewPort.

Group: [LCD Graphic](#) 

4.347 gGetTextMode

Function *gGetTextMode* : TWriteMode;

Returns the current write mode of the actual ViewPort.

Group: [LCD Graphic](#) 

4.348 gMoveTo

Procedure *gMoveTo* (Xd, Yd : integer);

Moves the virtual line draw cursor to the coordinate „Xd, Yd“.

Group: [LCD Graphic](#) 

4.349 gMoveToRel

Procedure *gMoveToRel* (Xr, Yr : integer);

Moves the virtual line draw cursor relative to it's previous position.

Group: [LCD Graphic](#) 

4.350 gOpenView

Procedure *gOpenView* (ViewPort : byte; Xs, Ys, Xe, Ye : integer) : boolean;

Defines the position and size of a ViewPort in physical pixel count.

Group: [LCD Graphic](#) 

4.351 gPntToScale

Procedure *gPntToScale* (ViewPort: byte; Xp, Yp : integer; **var** XL, YL : integer);

Converts a display absolute coordinate into a logical ViewPort coordinate.

Group: [LCD Graphic](#) 

4.352 gRestoreView

Procedure *gRestoreView* (source : pointer);

Overwrites the content of the current ViewPort with the content of the memory.

Group: [LCD Graphic](#) 

4.353 gSaveView

Procedure *gSaveView (dest : pointer);*

Stores the content of the current ViewPort into the RAM.

Group: [LCD Graphic](#) 

4.354 gScaleToPnt

Procedure *gScaleToPnt (ViewPort: byte; XL, YL : integer; var Xp, Yp : integer);*

Converts a logical ViewPort coordinate into a display absolute coordinate.

Group: [LCD Graphic](#) 

4.355 gScaleView

Function *gScaleView (ViewPort : byte; Xs, Ys, Xe, Ye : integer) : boolean;*

Defines the internal scalings of the ViewPort.

Group: [LCD Graphic](#) 

4.356 gSetBitMapRAM

Procedure *gSetBitMapRAM (RAM: boolean);*

Switches at runtime between BitMaps in ROM and RAM.

Group: [LCD Graphic](#) 

4.357 gSetCharSet

Procedure *gSetCharSet (source : pointer);*

Determines the current character set (5x7).

Group: [LCD Graphic](#) 

4.358 gSetCharSetRAM

Procedure *gSetCharSetRAM (RAM : boolean);*

Switches at runtime between ROM and RAM based char sets.

Group: [LCD Graphic](#) 

4.359 gSetLineColor

Procedure *gSetLineColor (c : byte);*

Determines the color for the current ViewPort.

Color must be imported:

From *LCDGraphic* **Import** *GraphColor;*

Group: [LCD Graphic](#) 

4.360 gSetLineMode

Procedure *gSetLineMode (LineWriteMode : TWriteMode);*

Determines the current line write mode for the actual ViewPort.

Group: [LCD Graphic](#) 

4.361 gSetPixel

Procedure *gSetPixel (Px, Py : integer);*

Sets a pixel to one (1) at the position "Px,Py" in the current ViewPort.

Group: [LCD Graphic](#) 

4.362 gSetTextBkGnd

Procedure *gSetTextBkGnd (backgnd : TTextBkGnd);*

Determines the text background mode for the current ViewPort.

Group: [LCD Graphic](#) 

4.363 gSetTextColor

Procedure *gSetTextColor (c : byte);*

Determines the color for the current ViewPort.

Color must be imported:

From *LCDGraphic Import* *GraphColor;*

Group: [LCD Graphic](#) 

4.364 gSetTextJustify

Procedure *gSetTextJustify (Horiz : TtxtAlHor; Vert : TTxtAlVert);*

Determines the text alignment of the current ViewPort.

Group: [LCD Graphic](#) 

4.365 gSetTextMode

Procedure *gSetTextMode (TextWriteMode : TWriteMode);*

Determines the text write mode for the current ViewPort.

Group: [LCD Graphic](#) 

4.366 gSwitchView

Procedure *gSwitchView (ViewPort: byte) : boolean;*

Changes the current ViewPort.

Group: [LCD Graphic](#) 

4.367 gXorPixel

Procedure gXorPixel (Px, Py : integer);

Inverts a pixel at the position "Px,Py" in the current ViewPort.

Group: [LCD Graphic](#) 

4.368 HexToInt

Function HexToInt (st : string) : integer [int8, byte, word, longint, longword];

Converts a Hex-string without a leading "\$".

Group: [Strings](#) 

4.369 Hi

Function Hi (w : word|integer) : byte;

Returns most significant byte of a 16bit value.

Group: [Maths](#) 

4.370 Higher

Function Higher (x, y : type) : type;

Returns the higher of two values. Types of both arguments must be the same.
type: byte, int8, word, integer, longint, longword, int64, word64, fix64, float.

Group: [Maths](#) 

4.371 HiNibble

Function HiNibble (w : byte|int8) : byte|int8;

Returns the High-nibble of a byte (higher 4bits).

Group: [Maths](#) 

4.372 HiWord

Function HiWord (ww : Longword|longint) : word|integer;

Returns the High word of a 32bit value (LongInt oder LongWord).

Group: [Maths](#) 

4.373 I2C_Dis7Clear

Procedure I2C_Dis7Clear (const Disp : TI2C_Dis7);

The whole display is deleted.

Group: [I2C Disp7](#) 

4.374 I2C_Disp7cIEOL

Procedure *I2C_Disp7cIEOL* (**const** Disp : TI2C_Disp7);

The display is deleted from the current position of the cursor upto the end of line.

Group: [I2C_Disp7](#) 

4.375 I2C_Disp7Ctrl

Function *I2C_Disp7Ctrl* (**const** Disp : TI2C_Disp7; **const** ctrl : TI2C_Ctrl7);

Switches all segments on/off.

Group: [I2C_Disp7](#) 

4.376 I2C_Disp7DigitBlink

Procedure *I2C_Disp7DigitBlink* (Disp : TI2C_Disp7; digit: byte; blink: boolean);

A single digit blinks or blinking is disabled.

Group: [I2C_Disp7](#) 

4.377 I2C_Disp7Dim

Procedure *I2C_Disp7Dim* (**const** Disp : TI2C_Disp7; **const** dim : byte);

Dims the display.

Group: [I2C_Disp7](#) 

4.378 I2C_Disp7Get

Function *I2C_Disp7Get* : TI2C_Disp7;

Returns the current selected display.

Group: [I2C_Disp7](#) 

4.379 I2C_Disp7Init

Function *I2C_Disp7Init* (**const** Disp : TI2C_Disp7; BlinkRate, DutyCycle : byte);

Initializes the display.

Group: [I2C_Disp7](#) 

4.380 I2C_Disp7Out

Procedure *I2C_Disp7Out* (**const** ch : char);

Writes into the display buffer.

Group: [I2C_Disp7](#) 

4.381 I2C_Disp7Pos

Procedure *I2C_Disp7Pos* (**const** *Disp* : *TI2C_Disp7*; **const** *digit* : *byte*);
The writing cursor is positioned to a specific location.

Group: [I2C Disp7](#) 

4.382 I2C_Disp7Refresh

Procedure *I2C_Disp7Refresh* (**const** *Disp* : *TI2C_Disp7*);
Forces an update of the display.

Group: [I2C Disp7](#) 

4.383 I2C_Disp7Set

Procedure *I2C_Disp7Set* (**const** *disp* : *TI2C_Disp7*);
Optional, switches the driver to a specific display.

Group: [I2C Disp7](#) 

4.384 I2CexpStat

Function *I2CexpStat* (*Port*: *TI2Cport*) : *boolean*;
Checks the state of an I2C-PortSlave.

Group: [I2C Expand](#) 

4.385 I2CexpStat_5

Function *I2CexpStat* (*Port*: *TI2Cport*) : *boolean*;
Checks the state of an I2C-PortSlave.

Group: [I2Cexpand_5](#) 

4.386 I2Cinp

Function *I2Cinp* (*SlaveAdr* : *byte*; **var** *Data*) : *boolean*;
Reads at least one byte from the chosen slave.

Group: [I2C Port](#) 

4.387 I2Cout

Function *I2Cout* (**const** *SlaveAdr* : *byte*, *Command* : *byte|word* [*Data*]) : *boolean*;
Writes at least one byte to the choosen slave.

Group: [I2C Port](#) 

4.388 I2Cstat

Function *I2Cstat* (*SlaveAdr* : *byte*) : *boolean*;
Checks the state of an I2C-Slave.

Group: [I2C Port](#)¹⁴

4.389 Inc

Procedure *Inc* (*var* *v* [, *step*] : *type*);
Increments a variable.

Group: [Maths](#)⁴

4.390 Incl

Procedure *Incl* (*v* : *byte*; *num* : *byte*);
Sets a bit in a byte.

Procedure *Incl* (*SrcDest* : *BitSet*; *op* : *BitSet*);
Sets a bit in a BitSet.

Group: [System](#)⁷

4.391 IncrCount4start

Procedure *IncrCount4start*;
Starts the Scan Timer.

Group: [Increment Counter 4chan](#)¹²

4.392 IncrCount4stop

Procedure *IncrCount4stop*;
Stops the Scan Timer.

Group: [Increment Counter 4chan](#)¹²

4.393 IncSema

Procedure *IncSema* (*s* : *semaphore*);
Increments the Semaphore by one.

Group: [System](#)⁷

4.394 IncToLim

Function *IncToLim* (*var* *v* : *ordinal* [, *limit* : *ordinal*] ; *val* : *ordinal*]) : *boolean*;
The argument "v" is incremented, provided that "v" has not reached it's natural limit.

Group: [Maths](#)⁴

4.395 IncToLimWrap

Function *IncToLimWrap* (**var** value, lim, pres : type) : boolean;
Increments the variable "value" by one.

Group: [Maths](#)  4

4.396 Inp_Raise

Function *Inp_Raise1* (bit : byte) : boolean;
Function *Inp_Raise2* (bit : byte) : boolean;
Function *Inp_Raise_G* (bit : byte) : boolean;
Returns true, if pin "bit" is has been activated.

Group: [SwitchPorts](#)  38

4.397 Inp_Stable

Function *Inp_Stable1* (bit : byte) : boolean;
Function *Inp_Stable2* (bit : byte) : boolean;
Function *Inp_Stable_G* (bit : byte) : boolean;
Returns true, if pin "bit" is active.

Group: [SwitchPorts](#)  38

4.398 Insert

Procedure *Insert* (src : string; **var** dst : string; p : byte);
Inserts a string into a string.

Group: [Strings](#)  6

4.399 Int

Function *Int* (f : float|fix64) : float|fix64;
Returns the integer part of the argument.

Group: [Maths](#)  4

4.400 IntegrateB

Function *IntegrateB* (oldVal, newVal, fact : byte) : byte;
Integrate a new value with an existing.

Group: [Maths](#)  4

4.401 Integratel

Function *Integratel* (*oldVal*, *newVal* : integer; *fact* : byte) : integer;
Integrate a new value with an existing.

Group: [Maths](#) 

4.402 Integratel8

Function *Integratel8* (*oldVal*, *newVal* : int8; *fact* : byte) : int8;
Integrate a new value with an existing.

Group: [Maths](#) 

4.403 IntegrateW

Function *IntegrateW* (*oldVal*, *newVal* : word; *fact* : byte) : word;
Integrate a new value with an existing.

Group: [Maths](#) 

4.404 InterPolX

Function *InterPolX* (**const** *LookUp* : pointer; *x*: type; **var** *y*: type) : boolean;
Interpolates a value with a table.

Group: [Diverse](#) 

4.405 InterPolY

Function *InterPolY* (**const** *LookUp* : pointer; *y*: type; **var** *x*: type) : boolean;
Interpolates a value with a table.

Group: [Diverse](#) 

4.406 IntToBin

Function *IntToBin* (*value* : word|integer) : string;
Returns the representation of the bits of the arguments in a string.

Group: [Strings](#) 

4.407 IntToFix64

Function *IntToFix64* (*i* : ordinal) : fix64;
converts an ordinal value (Byte...LongInt) into a Fix64

Groups: [Maths](#) , [Fix64](#) 

4.408 IntToHex

Function *IntToHex* (*i* : integer) : string;

Converts a numeric 16bit value into a hex-string.

Group: [Strings](#) 

4.409 IntToStr

Function *IntToStr* (*i* : word) : string;

Converts a numeric 16bit value into a string.

Group: [Strings](#) 

4.410 IOexpUpdate

Procedure *IOexpUpdate*;

Reads all input shift registers and stores them into the input array.

Group: [IOexpand](#) 

4.411 IPtoStr

Function *IPtoStr* (*IP* : TIPAddr) : String[15];

Converts an IP address array to a sting "aaa:bbb:ccc:ddd".

Group: [Strings](#)  [TINA TCP/IP](#)  [WizNet TCP/IP](#) 

4.412 IsCurProcess

Function *IsCurProcess* (*ID* : byte|*Name* : ProcName) : boolean;

Interrogation if the current process has the ID or the Name.

Group: [MultiTasking](#) 

4.413 IsPowOfTwo

Function *IsPowOfTwo* (*n* : type) : boolean;

Checks the number "n" whether is a power of 2.

Group: [Maths](#) 

4.414 IsSysTimerZero

Function *IsSysTimerZero* (*tm* : tSysTimer) : boolean;

Returns true if the timer has a value of zero.

Group: [Diverse](#) 

4.415 KeyBoardEnable

Procedure *KeyBoardEnable* (*ena* : *boolean*);
Enables or disables the entire KeyBoard.

Group: [KeyBoard 4x4](#) 

4.416 KeyBoardEnable8

Procedure *KeyBoardEnable8* (*ena* : *boolean*);
Enables or disables the entire KeyBoard.

Group: [KeyBoard 8x8](#) 

4.417 KeyBoardRepeat

Procedure *KeyBoardRepeat* (*rept* : *boolean*);
Enables or disables the repeat function.

Group: [KeyBoard 4x4](#) 

4.418 KeyBoardRepeat8

Procedure *KeyBoardRepeat8* (*rept* : *boolean*);
Enables or disables the repeat function.

Group: [KeyBoard 8x8](#) 

4.419 KeyRaised

Function *KeyRaised* (**const** *key* : *Keys*) : *boolean*;
The state of a key is returned by a true or false.

Group: [KeyBoard 4x4](#) 

4.420 KeyRaised8

Function *KeyRaised8* (**const** *key* : *Keys*) : *boolean*;
The state of a key is returned by a true or false.

Group: [KeyBoard 8x8](#) 

4.421 KeyStat

Function *KeyStat* : *boolean*;
The state of the entire KeyBoards is returned.

Group: [KeyBoard 4x4](#) 

4.422 KeyStat8

Function *KeyStat8* : *boolean*;

The state of the entire KeyBoards is returned.

Group: [KeyBoard 8x8](#) 

4.423 KeyStatRaised

Function *KeyStatRaised* : *boolean*;

The state of the KeyBoard is returned.

Group: [KeyBoard 4x4](#) 

4.424 KeyStatRaised8

Function *KeyStatRaised8* : *boolean*;

The state of the KeyBoard is returned.

Group: [KeyBoard 8x8](#) 

4.425 LANrxAutoAck

Procedure *LANrxAutoAck* (*const OnOff* : *boolean*);

Controls the Acknowledge mode.

Group: [Serial LAN](#) 

4.426 LANrxClear

Procedure *LANrxClear*;

The statusbyte of the RxBuffer is reset.

Group: [Serial LAN](#) 

4.427 LANrxStat

Function *LANrxStat* : *boolean*;

Returns a true if a frame has been received.

Group: [Serial LAN](#) 

4.428 LANtxClear

Procedure *LANtxClear*;

The status byte of the TxBuffer is reset.

Group: [Serial LAN](#) 

4.429 LANtxFrame

Function *LANtxFrame* (*node* : byte[word]; *len* : byte[word]) : boolean;
Sends a Buffer Frame.

Group: [Serial LAN](#)^[29]

4.430 LANtxStat

Function *LANtxStat* : boolean;
Checks whether a frame is already send or not.

Group: [Serial LAN](#)^[29]

4.431 LCDbarInit_M

Procedure *LCDbarInit_M*;
Initializes the BarGraph driver.

Group: [LCD Bargraph](#)^[18] [LCD Display Multi](#)^[19]

4.432 LCDbarInit_P

Procedure *LCDbarInit_P*;
Initializes the BarGraph driver.

Group: [LCD Bargraph](#)^[18] [LCD Display](#)^[18]

4.433 LCDbarOut

Procedure *LCDbarOut1* (*const b* : byte);
Procedure *LCDbarOut2* (*const b* : byte);
Procedure *LCDbarOut3* (*const b* : byte);
Procedure *LCDbarOut4* (*const b* : byte);
Changes the BarGraph to the new value and displays it.

Groups: [LCD Bargraph](#)^[18] [LCD Display](#)^[18] [LCD Display Multi](#)^[19]

4.434 LCDbarSet

Procedure *LCDbarSet1* (*const Line*, *PosA*, *Len*, *Scal* : byte);
Procedure *LCDbarSet2* (*const Line*, *PosA*, *Len*, *Scal* : byte);
Procedure *LCDbarSet3* (*const Line*, *PosA*, *Len*, *Scal* : byte);
Procedure *LCDbarSet4* (*const Line*, *PosA*, *Len*, *Scal* : byte);
Initializes the coordinate system of a BarGraph.

Groups: [LCD Bargraph](#)^[18] [LCD Display](#)^[18] [LCD Display Multi](#)^[19]

4.435 LCDcharset

Procedure *LCDcharset* (*loc* : *char*; *b1*, *b2*, *b3*, *b4*, *b5*, *b6*, *b7*, *b8* : *byte*);

Stores a program defined graphic character into the Character Generator Ram.

Group: [LCD Display](#)¹⁸

4.436 LCDcharset_M

Procedure *LCDcharset_M* (*LCD_num* : *TLCD_num*; *loc* : *char*; *c1*, *c2*, *c3*, *c4*, *c5*, *c6*, *c7*, *c8* : *byte*);

Stores a program defined graphic character into the Character Generator Ram.

Group: [LCD Display Multi](#)¹⁹

4.437 LCDcharset_MP

Procedure *LCDcharset_MP* (*LCD_num* : *TLCD_num*; *loc* : *char*; *srcArea* : *byte*; *ptr* : *pointer*);

Stores a program defined graphic character into the Character Generator Ram.

Group: [LCD Display Multi](#)¹⁹

4.438 LCDcharsetP

Procedure *LCDcharsetP* (**const** *loc* : *char*; *srcArea* : *byte*; *ptr* : *pointer*);

Stores a program defined graphic character into the Character Generator Ram.

Group: [LCD Display](#)¹⁸

4.439 LCDclr

Procedure *LCDclr*;

Deletes the contents of the display and moves the Cursor to position 0,0.

Group: [LCD Display](#)¹⁸

4.440 LCDclr_M

Procedure *LCDclr_M* (**const** *LCD_num* : *TLCD_num*);

Deletes the contents of the display and moves the Cursor to position 0,0.

Group: [LCD Display Multi](#)¹⁹

4.441 LCDclrEOL

Procedure *LCDclrEOL*;

Clears the current line from cursor position to line end.

Group: [LCD Display](#)¹⁸

4.442 LCDclrEOL_M

Procedure *LCDclrEOL_M* (**const** *LCD_num* : *TLCD_num*);

Clears the current line from cursor position to line end.

Group: [LCD Display Multi](#) ¹⁹

4.443 LCDclrLine

Procedure *LCDclrLine* (*Line* : *byte*);

Clears the desired Line(0..n) and places the Cursor to the line start.

Group: [LCD Display](#) ¹⁸

4.444 LCDclrLine_M

Procedure *LCDclrLine_M* (**const** *LCD_num* : *TLCD_num*; **const** *line* : *byte*);

Deletes the passed line number (0..n) and moves the cursor to the line start position.

Group: [LCD Display Multi](#) ¹⁹

4.445 LCDctrl

Procedure *LCDctrl* (**const** *b* : *byte*);

Writes on the LCD-ControlPort with *RS=0* and *RW=0*.

Group: [LCD Display](#) ¹⁸

4.446 LCDctrl_M

Procedure *LCDctrl_M* (**const** *LCD_num* : *TLCD_num*; **const** *b* : *byte*);

Writes on the LCD-ControlPort with *RS=0* and *RW=0*.

Group: [LCD Display Multi](#) ¹⁹

4.447 LCDcursor

Procedure *LCDcursor* (*on, blink* : *boolean*);

Switches the display ON and setup of the cursor mode.

Group: [LCD Display](#) ¹⁸

4.448 LCDcursor_M

Procedure *LCDcursor_M* (**const** *LCD_num* : *TLCD_num*; *on, blink* : *boolean*);

Switches the display ON and setup of the cursor mode.

Group: [LCD Display Multi](#) ¹⁹

4.449 LCDgetPort_M

Function *LCDgetPort_M* : *TLCD_num*;
Returns the current selected LCD-Port.

Group: [LCD Display Multi](#)¹⁹

4.450 LCDgetXY

Function *LCDgetXY* : *word*;
Reads the current Cursor Position.

Group: [LCD Display](#)¹⁸

4.451 LCDgetXY_M

Function *LCDgetXY_M* (**const** *LCD_num* : *TLCD_num*) : *word*;
Reads the current Cursor Position.

Group: [LCD Display Multi](#)¹⁹

4.452 LCDhome

Procedure *LCDhome*;
Moves the Cursor to position 0,0.

Group: [LCD Display](#)¹⁸

4.453 LCDhome_M

Procedure *LCDhome_M* (**const** *LCD_num* : *TLCD_num*);
Moves the Cursor to position 0,0.

Group: [LCD Display Multi](#)¹⁹

4.454 LCDinp

Function *LCDinp* : *byte*;
Reads the LCD DD-Ram or CG-Ram with *RS=1* and *RW=1*.

Group: [LCD Display](#)¹⁸

4.455 LCDinp_M

Function *LCDINP_M* (**const** *LCD_num* : *TLCD_num*) : *byte*;
Reads the LCD DD-Ram or CG-Ram with *RS=1* and *RW=1*.

Group: [LCD Display Multi](#)¹⁹

4.456 LCDlower

Procedure LCDlower;

Selects between two display controllers.

Group: [LCD Display](#)¹⁸

4.457 LCDoff

Procedure LCDoff;

Sets the display off, contents, positions and mode stay unchanged.

Group: [LCD Display](#)¹⁸

4.458 LCDoff_M

Procedure LCDoff_M (const LCD_num : TLCD_num);

Sets the display off, contents, positions and mode stay unchanged.

Group: [LCD Display Multi](#)¹⁹

4.459 LCDon

Procedure LCDon;

Sets the display on and sets cursor to ON and BLINK.

Group: [LCD Display](#)¹⁸

4.460 LCDon_M

Procedure LCDon_M (const LCD_num : TLCD_num);

Sets the display on and sets cursor to ON and BLINK.

Group: [LCD Display Multi](#)¹⁹

4.461 LCDout

Procedure LCDout (b : byte|char);

Writes into LCD display ram.

Group: [LCD Display](#)¹⁸

4.462 LCDout_M

Procedure LCDout_M (const c : char|byte);

Writes into LCD display ram.

Group: [LCD Display Multi](#)¹⁹

4.463 LCDportInp_M

Function *LCDportInp_M* (**const** *LCD_num* : *TLCD_num*) : *byte*;
Reads the upper 5 input pins of the control port.

Group: [LCD Display Multi](#) ¹⁹

4.464 LCDsetPort_M

Procedure *LCDsetPort_M* (**const** *LCD_num* : *TLCD_num*);
Selects a specific LCD.

Group: [LCD Display Multi](#) ¹⁹

4.465 LCDsetup

Procedure *LCDsetup*;
Initializes the LCD-controller.

Group: [LCD Display](#) ¹⁸

4.466 LCDsetup_M

Function *LCDsetup_M* (**const** *LCD_num* : *TLCD_num*) : *boolean*;
Initializes the LCD-controller.

Group: [LCD Display Multi](#) ¹⁹

4.467 LCDstat

Function *LCDstat* : *byte*;
Reads the LCD StatusPort with *RS=0* and *RW=1*.

Group: [LCD Display](#) ¹⁸

4.468 LCDstat_M

Function *LCDstat_M* (**const** *LCD_num* : *TLCD_num*) : *byte*;
Reads the LCD StatusPort with *RS=0* and *RW=1*.

Group: [LCD Display Multi](#) ¹⁹

4.469 LCDupper

Procedure *LCDupper*;
Selects between two display controllers.

Group: [LCD Display](#) ¹⁸

4.470 LCDxy

Procedure LCDxy (column, row : byte);

Positions the Cursors to column[x] and row[y].

Group: [LCD Display](#)^[18]

4.471 LCDxy_M

Procedure LCDxy_M (const LCD_num : TLCD_num; x, y : byte);

Positions the Cursors to column[x] and row[y].

Group: [LCD Display Multi](#)^[19]

4.472 LEDdotBlink

LEDdotBlink

T.B.D.

Group: [LED DOT Display](#)^[22]

4.473 LEDdotBlinkDigit

LEDdotBlinkDigit

T.B.D.

Group: [LED DOT Display](#)^[22]

4.474 LEDdotCharset

LEDdotCharset

T.B.D.

Group: [LED DOT Display](#)^[22]

4.475 LEDdotCharsetP

LEDdotCharsetP

T.B.D.

Group: [LED DOT Display](#)^[22]

4.476 LEDdotClr

LEDdotClr

T.B.D.

Group: [LED DOT Display](#)^[22]

4.477 LEDdotClrEOL

LEDdotClrEOL

T.B.D.Group: [LED DOT Display](#) **4.478 LEDdotClrLine**

LEDdotClrLine

T.B.D.Group: [LED DOT Display](#) **4.479 LEDdotCursor**

LEDdotCursor

T.B.D.Group: [LED DOT Display](#) **4.480 LEDdotDim**

LEDdotDim

T.B.D.Group: [LED DOT Display](#) **4.481 LEDdotGetXY**

LEDdotGetXY

T.B.D.Group: [LED DOT Display](#) **4.482 LEDdotHome**

LEDdotHome

T.B.D.Group: [LED DOT Display](#) **4.483 LEDdotInit**

LEDdotInit

T.B.D.Group: [LED DOT Display](#) 

4.484 LEDdotOff

LEDdotOff

T.B.D.

Group: [LED DOT Display](#) 

4.485 LEDdotOn

LEDdotOn

T.B.D.

Group: [LED DOT Display](#) 

4.486 LEDdotOut

LEDdotOut

T.B.D.

Group: [LED DOT Display](#) 

4.487 LEDdotXY

LEDdotXY

T.B.D.

Group: [LED DOT Display](#) 

4.488 Length

***Function** Length (s : string) : byte;*

Returns the current length of a string.

Group: [Strings](#) 

4.489 Lo

***Function** Lo (w : word) : byte;*

Returns least significant byte of a 16bit value.

Group: [Maths](#) 

4.490 Lock

***Procedure** Lock (p : process);*

The whole run-time is placed at disposal of a process.

Group: [MultiTasking](#) 

4.491 Log10

Function *Log10* (*f : float*) : *float*;
Returns the logarithm to base 10.

Group: [Maths](#) 

4.492 LogN

Function *LogN*(*f : float*) : *float*;
Returns the logarithm naturalis. Euler number $e = 2.71\dots$

Group: [Maths](#) 

4.493 Long64ToHex

Function *Long64ToHex* (**const** *ii : Int64|Word64*) : *string*;
Converts a numeric 64bit value into a hex-string.

Group: [Strings](#) 

4.494 Long64ToStr

Function *Long64ToStr* (**const** *ii : Int64|Word64*: **const** *len : byte*: **const** *space : char*]) : *string*;
Converts a numeric 64bit value into a string.

Group: [Strings](#) 

4.495 LongAsFloat

Function *LongAsFloat* (*L : LongWord*) : *float*;
Converts the argument into a Float.
There is **no type conversion!**

Group: [Diverse](#) 

4.496 LongToHex

Function *LongToHex* (*w : longword*) : *string*;
Converts a numeric 32bit value into a hex-string.

Group: [Strings](#) 

4.497 LongToStr

Function *LongToStr* (*ii : longword*) : *string*;
Converts a numeric 32bit value into a string.

Group: [Strings](#) 

4.498 LoNibble

Function *LoNibble* (*w* : *byte|int8*) : *byte|int8*;
Returns the Low-nibble of a byte (lower 4bits).

Group: [Maths](#) 

4.499 LowCase

Function *LowCase* (*ch* : *char*) : *char*;
Converts a char into lowercase letter.

Group: [Strings](#) 

4.500 Lower

Function *Lower* (*x, y* : *type*) : *type*;
Returns the lower of two values. Types of both arguments must be the same.
type: byte, int8, word, integer, longint, longword, int64, word64, fix64, float.

Group: [Maths](#) 

4.501 LowerCase

Function *LowerCase* (*st* : *string*) : *string*;
Converts a string into lowercase letters.

Group: [Strings](#) 

4.502 LoWord

Function *LoWord* (*v* : *Longword|longint*) : *word [integer]*;
Returns the Low word of a 32bit value (LongInt oder LongWord).

Group: [Maths](#) 

4.503 LowPassFW

LowPassFW

T.B.D.

Group: [Maths](#) 

4.504 LPTctrl

Procedure *LPTctrl* (*ctrl* : *tLPTlineSet*);
Controls the control port and the control lines to the printer.

Group: [LPT Printer](#) 

4.505 LPTdir

Procedure LPTdir (inp : boolean);

Controls the data direction of the data port. True = output, false = input.

Group: [LPT Printer](#)^[23]

4.506 LPTinit

Procedure LPTinit;

Initialises the printer by the Init line and the Select line.

Group: [LPT Printer](#)^[23]

4.507 LPTinp

Function LPTinp : byte;

Reads the data port.

Group: [LPT Printer](#)^[23]

4.508 LPTout

Procedure LPTout (dat : byte);

This is the printing function.

Group: [LPT Printer](#)^[23]

4.509 LPTreset

Procedure LPTreset;

Generates a short pulse on the Init line.

Group: [LPT Printer](#)^[23]

4.510 LPTstat

Function LPTstat : tLPTlineset;

Returns the state of the control port.

Group: [LPT Printer](#)^[23]

4.511 MACtoStr

Function MACtoStr (MAC : TMACAddr) : String[17];

Converts a MAC-address array to a string "aa:bb:cc:dd:ee:ff".

Group: [Strings](#)^[6] [TINA TCP/IP](#)^[40] [WizNet TCP/IP](#)^[44]

4.512 Max

Function *Max* (*x*: type) : type;

Returns the highest possible value of the type.

Group: [Maths](#)^[4]

4.513 mb_GetModBusDevID

Function *mb_GetModBusDevID*: byte;

Returns the ID of the device.

Group: [ModBus ASCII](#)^[24] [ModBus RTU](#)^[25]

4.514 mb_GetModBusExceptionStatus

Function *mb_GetModBusExceptionStatus*: byte;

Gets the ExceptionStatus.

Group: [ModBus RTU](#)^[25]

4.515 mb_GetModBusTimeout

Function *mb_GetModBusTimeout*: word;

Gets the Time-Out between two Rx-Bytes (msec).

Group: [ModBus ASCII](#)^[24] [ModBus RTU](#)^[25]

4.516 mb_SetAfterCoilRead

procedure *mb_SetAfterCoilRead* (proc: tAfterCoilRead);

Callback Function. Change coils after they are read from the client.

Group: [ModBus ASCII](#)^[24] [ModBus RTU](#)^[25]

4.517 mb_SetAfterCoilWrite

procedure *mb_SetAfterCoilWrite* (proc: tAfterCoilWrite);

Callback Function. Change coils after they are received from the client.

Group: [ModBus ASCII](#)^[24] [ModBus RTU](#)^[25]

4.518 mb_SetAfterRegisterRead

procedure *mb_SetAfterRegisterRead* (proc: tAfterRegisterRead);

Callback Function. Change registers after they are sent to the client.

Group: [ModBus ASCII](#)^[24] [ModBus RTU](#)^[25]

4.519 mb_SetAfterRegisterWrite

procedure mb_SetAfterRegisterWrite (proc: tAfterRegisterWrite);

Callback Function. Change registers after they are received from the client.

Group: [ModBus ASCII](#) ^[24] [ModBus RTU](#) ^[25]

4.520 mb_SetBeforeCoilRead

procedure mb_SetBeforeCoilRead (proc: tBeforeCoilRead);

Callback Function. Change coils before they are sent to the client.

Group: [ModBus ASCII](#) ^[24] [ModBus RTU](#) ^[25]

4.521 mb_SetBeforeCoilWrite

procedure mb_SetBeforeCoilWrite (proc: tBeforeCoilWrite);

Callback Function. Change coils before they are received from the client.

Group: [ModBus ASCII](#) ^[24] [ModBus RTU](#) ^[25]

4.522 mb_SetBeforeRegisterRead

procedure mb_SetBeforeRegisterRead (proc: tBeforeRegisterRead);

Callback Function. Change registers before they are sent to the client.

Group: [ModBus ASCII](#) ^[24] [ModBus RTU](#) ^[25]

4.523 mb_SetBeforeRegisterWrite

procedure mb_SetBeforeRegisterWrite (proc: tBeforeRegisterWrite);

Callback Function. Change registers before they are received from the client.

Group: [ModBus ASCII](#) ^[24] [ModBus RTU](#) ^[25]

4.524 mb_SetModBusDevID

Procedure mb_SetModBusDevID(id: byte);

Every MODBUS device must have a unique id.

Group: [ModBus ASCII](#) ^[24] [ModBus RTU](#) ^[25]

4.525 mb_SetModBusExceptionStatus

Procedure mb_SetModBusExceptionStatus (status: byte);

Sets the ExceptionStatus.

Group: [ModBus RTU](#) ^[25]

4.526 mb_SetModBusTimeout

Procedure *mb_SetModBusTimeout* (*time*: *word*);
Sets the Time-Out between two Rx-Bytes (msec).

Group: [ModBus ASCII](#)^[24] [ModBus RTU](#)^[25]

4.527 mDelay

Procedure *mDelay* (*d*: *word*);
Software Delay in msec.

Group: [System](#)^[7]

4.528 Min

Function *Min* (*x*: *type*): *type*;
Returns the lowest possible value of the type.

Group: [Maths](#)^[4]

4.529 Mirror16

Function *Mirror16* (*w*: *word*|*integer*): *word*|*integer*;
Mirrors the argument. Exchange bit15 <-> bit0, bit14 <-> bit1, ...

Group: [Maths](#)^[4]

4.530 Mirror32

Function *Mirror32* (*Lw*: *longword*|*longint*): *longword*|*longint*;
Mirrors the argument. Exchange bit31 <-> bit0, bit30 <-> bit1, ...

Group: [Maths](#)^[4]

4.531 Mirror8

Function *Mirror8* (*b*: *int8*|*byte*|*char*): *int8*|*byte*|*char*;
Mirrors the argument. Exchange bit7 <-> bit0, bit6 <-> bit1, ...

Group: [Maths](#)^[4]

4.532 MRFgetLostPkts

Function *mrfGetLostPkts*: *byte*;
returns the count of failed Tx-packets which were not successful despite of using all retries. Also the internal lost counter in the 24L01 is reset.

Group: [MIRF24 Port](#)^[24]

4.533 MRFgetRetryCnt

Function *mrfGetRetryCnt* : *byte*;

After a successful Tx-operation the function returns the used retries.
The higher the count is the worse the connection was.

Group: [MIRF24 Port](#) 

4.534 MRFgetRxPower

Function *mrfGetRxPower* : *byte*;

returns in the Bit0 the received field quality. 0 = bad reception, 1 = good reception.

Group: [MIRF24 Port](#) 

4.535 MRFgetRxType

Function *mrfGetRxType* : *tMRFpkt*;

returns the receive packet state.

tMRFpkt = (mrfPKTnone, mrfPKTdata, mrfPKTbcast) .

It serves for polling the reception.

Group: [MIRF24 Port](#) 

4.536 MRFgetState

Function *mrfGetState* : *tMRFstat*;

returns the Status register of the 24L01.

tMRFstat = BitSet of enMRFstat.

enMRFstat = (mrfTX_full, mrfRX_pn0, mrfRX_pn1, mrfRX_pn2, mrfMAX_RT,
mrfTX_DS,mrfRX_DR);

Normally it is not used. The meaning of the bits must be read in the 24L01 datasheet.

Group: [MIRF24 Port](#) 

4.537 MRFininit

Function *mrfInit* : *boolean*;

initialises the 24L01 with the values given above and starts the receiver.

If an error occurs a false is returned.

Group: [MIRF24 Port](#) 

4.538 MRFrxPacket

Function *mrfRxPacket* (*destPtr* : *pointer*; *TimeOut* : *byte*; *var recvd* : *byte*) : *tMRFpkt*

tries to get a packet out of the 24L01.

destPtr must point to a RAM data structure with 32bytes size.

TimeOut defines the time in msec which should be waited until success or abort.

In recvd the real received byte count is returned (max 32).

The function result is tMRFpkt = (mrfPKTnone, mrfPKTdata, mrfPKTbcast) .

With a timeout a mrfPKTnone is returned. If a standard data packet has been received a mrfPKTdata is returned.
And if a Broadcast has been received a mrfPKTbcast is returned.

Group: [MIRF24 Port](#) 

4.539 MRFsetChan

Procedure *mrfSetChan* (*chan* : *tMRFchan*; *wr* : *boolean*);
selects the desired RF channel in the ISM band. The channel *chan* must be passed with the enumeration *tMRFchan* (*mrfChan1*.. *mrfChan14*).
wr defines whether the channel must be immediately written into the 24L01 or not. Before the first Init the *wr* must be false. After an Init the channel can be changed at runtime with *wr* = true without doing a new Init.
As an alternative the procedure [MRFsetFreq](#) can be used.

Group: [MIRF24 Port](#) 

4.540 MRFsetFreq

Procedure *mrfSetFreq* (*freq* : *word*; *wr* : *boolean*);
selects the desired RF frequency in the ISM band.
The *freq* parameter counts in MHz steps. Valid values are 2400..2484.
wr defines whether the frequency must be immediately written into the 24L01 or not. Before the first Init the *wr* must be false. After an Init the frequency can be changed at runtime with *wr* = true without doing a new Init.
As an alternative the procedure [MRFsetChan](#) can be used.

Group: [MIRF24 Port](#) 

4.541 MRFsetLocalAdr

Procedure *mrfSetLocalAddr* (*adr* : *byte*);
defines the local address (0..255).
This is the logical address which other nodes must use to communicate with this node. This address must be unique: only one node in this network can have this address.

Group: [MIRF24 Port](#) 

4.542 MRFsetPower

Procedure *mrfSetPower* (*pwr* : *tMRFpwr*);
sets the desired RF power.
The power must be passed with the enumeration *tMRFpwr* (*mrfdBm0*.. *mrfdBm18*). *mrfdBm0* is max. Power and *mrfdBm18* is minimal power.

Group: [MIRF24 Port](#) 

4.543 MRFsetPWRdown

Procedure *mrfSetPWRdown*;

switches the controller 24L01 into the Power-down mode.
For a restart then the function [MRFinit](#) must be used.

Group: [MIRF24 Port](#) 

4.544 MRFsetRetryMax

Procedure *mrfSetRetryMax (rmax : byte)*;

defines the max. retry count (0..15). If there is an error in a data transfer, e.g. the acknowledge of the receiver is too late or fails then the transmitter repeats the telegram up to rmax.

If the ACK is still missing the transmit function *mrfTxPacket* returns with a false.

Group: [MIRF24 Port](#) 

4.545 MRFsetRetryTimeOut

Procedure *mrfSetRetryTimeOut (tmo : byte)*;

sets the delay between two retries (1..15). This time is calculated in this way:
 $250\text{usec} + (\text{tmo} * 250\text{usec})$. Typical values should be about 1msec ($\text{tmo} = 3..4$)

Group: [MIRF24 Port](#) 

4.546 MRFsetRFspeed

Procedure *mrfSetRFspeed (spd : tMRFrSpeed)*;

defines the "on air" data rate. The rate must be given with the enumeration *tMRFrSpeed* (*mrfRF250*, *mrfRF1000*, *mrfRF2000*).

Group: [MIRF24 Port](#) 

4.547 MRFtxPacket

Function *mrfTxPacket (adr : byte; srcPtr : pointer; cnt : byte; bc : boolean) : boolean*;

tries to transmit a data packet or a broadcast packet.

adr defines the logical address (0..255) of the desired receivers (Node).

srcPtr must point to a source in RAM.

cnt defines the count (size) of the packet to send (max 32).

bc defines whether this should be a broadcast or a data packet.

With a broadcast the parameter *adr* is ignored.

The result becomes false if any hardware problem occurred (Data or Broadcast).

A false is also returned (Data) if the receiver after x Tx-retries did not send any ACK or the ACK got lost.

The MIRF24port driver provides two types of data packet transfers:

1. Broadcast.

This packet has certain info so that all reachable MIRF24 nodes receive this packet and can interpret it. In this case the hardware ACK is suppressed and also the receiver application must not respond with any packet.

2. Data.

This packet is determined for an unique address (node) and will be received only by this node.

The receiver (Node) then automatically responds with hardware ACK to the sender and confirms the correct reception.

With a CRC error etc. the ACK is suppressed and the sender must do a retry.

An ACK from the receiver at this time doesn't mean that the receiving application has read out the packet from the 24L01.

As long as this didn't happen the receiver doesn't respond with any ACK for incoming packets.

So the send function returns with a timeout after the retry count has been elapsed.

Attention:

Before sending a broadcast and after this there should be a pause so all receiver are ready.

Group: [MIRF24 Port](#) ^[24]

4.548 MSPlinOut

***Procedure** MSPlinOut0 (source, dest : pointer; count : word);*

***Procedure** MSPlinOut1 (source, dest : pointer; count : word);*

...

Group: [ATMega MSPI HardwareDriver](#) ^[34]

XMega:

***Procedure** MSPlinOut_C0 (source, dest : pointer; count : word);*

***Procedure** MSPlinOut_C1 (source, dest : pointer; count : word);*

***Procedure** MSPlinOut_D0 (source, dest : pointer; count : word);*

...

Group: [XMega MSPI HardwareDriver](#) ^[35]

Reads and writes a data block from/into the SPI-slave.

4.549 MSPlinOutByte

***Function** MSPlinOutByte0 (b : byte) : byte;*

***Function** MSPlinOutByte1 (b : byte) : byte;*

...

Group: [ATMega MSPI HardwareDriver](#) ^[34]

XMega:

***Function** MSPlinOutByte_C0 (b : byte) : byte;*

***Function** MSPlinOutByte_C1 (b : byte) : byte;*

***Function** MSPlinOutByte_D0 (b : byte) : byte;*

...

Group: [XMega MSPI HardwareDriver](#) ^[35]

Reads and writes a byte from/into the SPI-slave.

4.550 MSPlinp

Procedure MSPlinp0 (dest : pointer; count : word);

Procedure MSPlinp1 (dest : pointer; count : word);

...

Group: [ATMega MSPI HardwareDriver](#)^[34]

XMega:

Procedure MSPlinp_C0 (dest : pointer; count : word);

Procedure MSPlinp_C1 (dest : pointer; count : word);

Procedure MSPlinp_D0 (dest : pointer; count : word);

...

Group: [XMega MSPI HardwareDriver](#)^[35]

Reads a block of data from the MSPI-slave.

4.551 MSPlinpByte

Function MSPlinpByte0 : byte;

Function MSPlinpByte1 : byte;

...

Group: [ATMega MSPI HardwareDriver](#)^[34]

XMega:

Function MSPlinpByte_C0 : byte;

Function MSPlinpByte_C1 : byte;

Function MSPlinpByte_D0 : byte;

...

Group: [XMega MSPI HardwareDriver](#)^[35]

Reads a byte from the MSPI-slave.

4.552 MSPlinpLong

Function MSPlinpLong0 : longword;

Function MSPlinpLong1 : longword;

...

Group: [ATMega MSPI HardwareDriver](#)^[34]

XMega:

Function MSPlinpLong_C0 : longword;

Function MSPlinpLong_C1 : longword;

Function MSPlinpLong_D0 : longword;

...

Group: [XMega MSPI HardwareDriver](#)^[35]

Reads a longword from the MSPI-slave.

4.553 MSPlinpWord

Function MSPlinpWord0 : word;

Function MSPlinpWord1 : word;

...

Group: [ATMega MSPI HardwareDriver](#)^[34]

XMega:**Function** *MSPlinpWord_C0* : word;**Function** *MSPlinpWord_C1* : word;**Function** *MSPlinpWord_D0* : word;

...

Group: [XMega MSPI HardwareDriver](#)^[35]

Reads a word from the MSPI-slave.

4.554 MSPlout

Procedure *MSPlout0* (source : pointer; count : word);**Procedure** *MSPlout1* (source : pointer; count : word);

...

Group: [ATMega MSPI HardwareDriver](#)^[34]**XMega:****Procedure** *MSPlout_C0* (source : pointer; count : word);**Procedure** *MSPlout_C1* (source : pointer; count : word);**Procedure** *MSPlout_D0* (source : pointer; count : word);

...

Group: [XMega MSPI HardwareDriver](#)^[35]

Writes a byte into the MSPI-slave.

4.555 MSPloutByte

Procedure *MSPloutByte0* (b : byte);**Procedure** *MSPloutByte1* (b : byte);

...

Group: [ATMega MSPI HardwareDriver](#)^[34]**XMega:****Procedure** *MSPloutByte_C0* (b : byte);**Procedure** *MSPloutByte_C1* (b : byte);**Procedure** *MSPloutByte_D0* (b : byte);

...

Group: [XMega MSPI HardwareDriver](#)^[35]

Writes a byte into the MSPI-slave.

4.556 MSPloutLong

Procedure *MSPloutLong0* (L : longword);**Procedure** *MSPloutLong1* (L : longword);

...

Group: [ATMega MSPI HardwareDriver](#)^[34]**XMega:****Procedure** *MSPloutLong_C0* (l : longword);**Procedure** *MSPloutLong_C1* (l : longword);**Procedure** *MSPloutLong_D0* (l : longword);

...

Group: [XMega MSPI HardwareDriver](#)^[35]

Writes a longword into the MSPI-slave.

4.557 MSPloutWord

Procedure MSPloutWord0 (w : word);

Procedure MSPloutWord1 (w : word);

...

Group: [ATMega MSPI HardwareDriver](#)^[34]

XMega:

Procedure MSPloutWord_C0 (w : word);

Procedure MSPloutWord_C1 (w : word);

Procedure MSPloutWord_D0 (w : word);

...

Group: [XMega MSPI HardwareDriver](#)^[35]

Writes a word into the MSPI-slave.

4.558 MulDivByte

Function MulDivByte (a1, a2, d : byte) : byte;

Calculates the 16bit result of the multiplication and divides this result by the 8bit divisor. This assures that an overflow error is impossible.

Group: [Maths](#)^[4]

4.559 MulDivInt

Function MulDivInt (a1, a2, d : integer|word) : integer|word;

Calculates the 32bit result of the multiplication and divides this result by the 16bit divisor. This assures that an overflow error is impossible.

Group: [Maths](#)^[4]

4.560 MulDivInt8

Function MulDivInt8 (a1, a2, d : int8) : int8;

Calculates the 16bit result of the multiplication and divides this result by the 8bit divisor. This assures that an overflow error is impossible.

Group: [Maths](#)^[4]

4.561 MulDivLong

Function MulDivLong (a1, a2, d : longint|longword) : longint|longword;

Calculates the 64bit result of the multiplication and divides this result by the 32bit divisor. This assures that an overflow error is impossible.

Group: [Maths](#)^[4]

4.562 Negate

Function *Negate* (*v : type*) : *type*;

Returns the negative value (Two's Complement) of a Byte, Int8..Longword, LongInt, Int64, Fix64 or Float value

a := Negate (a);

Group: [Maths](#) 

4.563 NoInts

Procedure *NoInts*;

Disables the interrupts without changing the corresponding system flag.

Group: [System](#) 

4.564 NOP

Procedure *NOP*;

Creates an assembler "NOP".

Group: [System](#) 

4.565 Odd

Function *Odd* (*x : type*) : *boolean*;

Tests the value for odd.

Group: [Maths](#) 

4.566 OnADCread

Procedure *OnADCread*;

Callback. Is called with each SysTick at the point where the ADC result is read.

Group: [ADC](#) 

4.567 OnFAT16_SS

Procedure *onFAT16_SS*;

Callback. Control of the SPI SS-pin by the application.

Group: [System](#) 

4.568 OnIdleProcess

Procedure *OnIdleProcess*;

Callback. Is called with on each restart of the Idle process.

Group: [System](#) 

4.569 OnSchedulerEntry

Procedure OnSchedulerEntry;

Callback. Is called every time at the entry into the Scheduler.

Group: [System](#) 

4.570 OnSchedulerExit

Procedure OnSchedulerExit;

Callback. Is called every time on exit of the Scheduler.

Group: [System](#) 

4.571 OnSerRxResumed

Procedure OnSerRxResumed1;

Procedure OnSerRxResumed2;

...

XMega:

Procedure OnSerRxResumedC0;

Procedure OnSerRxResumedC1;

Procedure OnSerRxResumedD0;

...

Callback. Is called if the SerPort driver re-enables the RxBuffer by sending a XON or de-activating the DSR handshake line.

Group: [SerPort](#) 

4.572 OnSerRxStopped

Procedure OnSerRxStopped1;

Procedure OnSerRxStopped2;

...

XMega:

Procedure OnSerRxStoppedC0;

Procedure OnSerRxStoppedC1;

Procedure OnSerRxStoppedD0;

...

Callback. Is called if the SerPort driver disables the RxBuffer by sending a XOFF or activating the DSR handshake line.

Group: [SerPort](#) 

4.573 OnSysTick

Procedure *OnSysTick [(SaveAllRegs)];*

Callback. Is called with each SysTick (Timer Interrupt).

Group: [System](#) 

4.574 OnTickTimer

Procedure *OnTickTimer [(SaveAllRegs)];*

Callback. Is called with each Timer Tick.

Group: [System](#) 

4.575 OnTINA_SS

Procedure *onTINA_SS;*

Callback. Control of the SPI SS-pin by the application.

Group: [System](#) 

4.576 Ord

Function *Ord (ch : char) : byte;*

Returns the ordinal number of a symbol/character.

Group: [Maths](#) 

4.577 PadLeft

Function *PadLeft (const st : string; len : byte [;pad : char]) : string;*

Inserts leading characters into the string.

Group: [Strings](#) 

4.578 PadRight

Function *PadRight (const st : string; len : byte [;pad : char]) : string;*

Appends characters to the string.

Group: [Strings](#) 

4.579 Parity

Function *Parity (const b : byte|char) : boolean;*

Returns the parity of a byte or char. True, if parity is odd.

Group: [Maths](#) 

4.580 PipeFlush

Procedure PipeFlush (p : pipe);

Empty whole pipe. Also RxBuffer and RxBuffer1, -2,- 3.

Group: [Pipes](#) 

4.581 PipeFull

Function PipeFull (p : pipe) : boolean;

Returns the full state of a pipe.

Group: [Pipes](#) 

4.582 PipeRecv

Function PipeRecv (p : pipe) : type;

Fetches (removes) an argument from a pipe.

Group: [Pipes](#) 

4.583 PipeRecv_ND

Function PipeRecv_ND (pipe1 : pipe) : type;

Reads out of a pipe without removing or changing the pipe contents.

Group: [Pipes](#) 

4.584 PipeSend

Function PipeSend (p : pipe; v : type) : boolean;

Add an argument to a pipe (append).

Group: [Pipes](#) 

4.585 PipeStat

Function PipeStat (p : pipe) : byte;

Returns the number of available parameters.

Group: [Pipes](#) 

4.586 PopAllRegs

Procedure PopAllRegs;

Special register restore for nested interrupts.

Group: [System](#) 

4.587 PopRegs

Procedure *PopRegs*; // working registers from stack

simplified version of PopAllRegs and can always be used paired with [PushRegs](#) within Interrupts only the 4 important registers (ACCA, ACCB etc) are saved.

Group: [System](#) 

4.588 Pos

Function *Pos* (*a* : *char*; *s* : *string*) : *byte*;

Returns the position of a character within a string.

Group: [Strings](#) 

4.589 PosN

Function *PosN* (*a* : *char*; *s* : *string*; *start* : *byte*) : *byte*;

Returns the position of a character within a string. Search starts at a given position.

Group: [Strings](#) 

4.590 Pow

Function *Pow* (*x*,*y* : *float*) : *float*;

Returns the result of x power of y .

Group: [Maths](#) 

4.591 Pow10

Function *Pow10* (*x* : *float*) : *float*;

Returns the result of 10 power of x.

Group: [Maths](#) 

4.592 PowerSave

Procedure *PowerSave* (**const** *mode* : *byte*; **const** *ticks* : *word*);

The CPU sleeps n SysTicks.

Group: [System](#) 

4.593 Pred

Function *Pred* (*x* : *type*) : *type*;

Returns the next-lower value of a variable.

Group: [Maths](#) 

4.594 PresetAVfilter

Procedure *PresetAVfilter* (**var** Filter : AVfilter; val : type);
Populates the entire filter with "val".

Group: [Diverse](#) 

4.595 Priority

Procedure *Priority* (p : process|task; prio : byte);
Sets the priority of a task or of a process.

Group: [MultiTasking](#) 

4.596 ProcWaitFlag

Function *ProcWaitFlag* (Flag : var [; timeout : word]) : boolean;
A Process/Task becomes inactive until the flag is <> 0.

Group: [MultiTasking](#) 

4.597 PulseCountClear

Function *PulseCountClear*;
Stops the counter and sets it to 0.

Group: [Pulse Counter](#) 

4.598 PulseCountClear2

Function *PulseCountClear2*;
Stops the counter and sets it to 0.

Group: [Pulse Counter](#) 

4.599 PulseCountStart

Function *PulseCountStart*;
Starts the counter after program start, [PulseCountClear](#) or [PulseCountStop](#).

Group: [Pulse Counter](#) 

4.600 PulseCountStart2

Function *PulseCountStart2*;
Starts the counter after program start, [PulseCountClear](#) or [PulseCountStop2](#).

Group: [Pulse Counter](#) 

4.601 PulseCountStop

Function *PulseCountStop*;

Stops the counter without changing the internal counter.

Group: [Pulse Counter](#) 

4.602 PulseCountStop2

Function *PulseCountStop2*;

Stops the counter without changing the internal counter.

Group: [Pulse Counter](#) 

4.603 PushAllRegs

Procedure *PushAllRegs*;

Special register saving for nested interrupts.

Group: [System](#) 

4.604 PushRegs

Procedure *PushRegs*; // working registers to stack

simplified version of PushAllRegs and can always be used paired with [PopRegs](#) within Interrupts if only the 4 important registers (ACCA, ACCB etc) are saved.

Group: [System](#) 

4.605 RadToDeg

Function *RadToDeg* (*w : float*) : float;

Turns an angular value in radians into degrees.

Group: [Maths](#) 

4.606 RaiseException

Procedure *RaiseException* (*num : byte*);

Raises the exception. See [try](#)

Group: [System](#) 

4.607 Random

Function *Random* : word;

Returns a random number of the type Word.

Random must be imported:

From System **Import** Random;

Group: [System](#) 

4.608 RandomRange

Function *RandomRange*(*min, max : word*) : *word*;

Returns a random number of the type Word. Value is limited by min and max.

Random must be imported:

From *System* **Import** *Random*;

Group: [System](#) 

4.609 Read

Function *Read* (*p : Function*]; **var** *st : string*]; *count : byte*|*limiter : char*]);

Reads a character or a string.

Group: [Various](#) 

4.610 ReadKey

Function *ReadKey* (**const** *key : Keys*) : *boolean*;

The current state of a key is returned.

Group: [KeyBoard 4x4](#) 

4.611 ReadKey8

Function *ReadKey8* (**const** *key : Keys*) : *boolean*;

The current state of a key is returned.

Group: [KeyBoard 8x8](#) 

4.612 ReadKeyBoard

Function *ReadKeyBoard* : *BitSet of Keys*;

Returns the current state of the keyboard.

Group: [KeyBoard 4x4](#) 

4.613 ReadKeyBoard8

Function *ReadKeyBoard8* : *BitSet of Keys*;

Returns the current state of the keyboard.

Group: [KeyBoard 8x8](#) 

4.614 ReadLn

Procedure *ReadLn* (*DeviceFunc : function*; **var** *str : string*);

Reads a string until CarriageReturn/LineFeed is found.

Group: [Various](#) 

4.615 RecvRC5

Function *RecvRC5* (**var** rxAdr : byte; **var** rxCmd : byte) : boolean;
Receives a RC5 telegram.

Group: [RC5 Driver](#) 

4.616 ResetProcess

Procedure *ResetProcess* (P : Name | i : ID);
Re-initializes a process completely and suspends it.

Group: [MultiTasking](#) 

4.617 ResetSysTimer

Procedure *ResetSysTimer* (tm : tSysTimer);
Resets a SysTimer to 0.

Group: [Diverse](#) 

4.618 RestoreInts

Procedure *RestoreInts*;
Conditional enable of interrupts depending on the corresponding system flag.

Group: [System](#) 

4.619 Resume

Procedure *Resume* (p : process|task);
A process/task, which is de-activated gets activated.

Group: [MultiTasking](#) 

4.620 ResumeAll

Procedure *ResumeAll* (Processes, Tasks);
Procedure *ResumeAll* (Processes);
Procedure *ResumeAll* (Tasks);
All processes/tasks, which are de-activated get activated.

Group: [MultiTasking](#) 

4.621 RotatePntI

Procedure *RotatePntI* (angle, XPo, YPo : integer; **var** XPd, YPd : integer);
The coordinate (XPo, YPo) will be rotated by „angle" (degrees).

Group: [Maths](#) 

4.622 Round

Function *Round* (*f* : float|fix64) : integer; {Byte,Word,LongInt,LongWord}
Rounds a Float, Fix64 to an Integer, Byte, Word, etc.

Group: [Maths](#) 

4.623 RTCalarm

Procedure *RTCalarm*;
CallBack. Is called at a Compare-match.

Group: [RTC Driver](#) 

4.624 RTCalarm_Date

Procedure *RTCalarm_Date* (*year, month, day* : byte);
Sets the date compare register of RTCalarm.

Group: [RTC Driver](#) 

4.625 RTCalarm_Start

Procedure *RTCalarm_Start* (*mode* : byte);
Starts the RTCalarm.

Group: [RTC Driver](#) 

4.626 RTCalarm_Stop

Procedure *RTCalarm_Stop*;
Stops the RTCalarm.

Group: [RTC Driver](#) 

4.627 RTCalarm_Time

Procedure *RTCalarm_Time* (*hour, min, sec* : byte);
Sets the time compare register of RTCalarm.

Group: [RTC Driver](#) 

4.628 RTCgetDay

Function *RTCgetDay* : byte;
Reads the days register.

Group: [RTC Driver](#) 

4.629 RTCgetHour

Function *RTCgetHour* : byte;
Reads the hours register.

Group: [RTC Driver](#) 

4.630 RTCgetMinute

Function *RTCgetMinute* : byte;
Reads the minutes register.

Group: [RTC Driver](#) 

4.631 RTCgetMonth

Function *RTCgetMonth* : byte;
Reads the months register.

Group: [RTC Driver](#) 

4.632 RTCgetSecond

Function *RTCgetSecond* : byte;
Reads the seconds register.

Group: [RTC Driver](#) 

4.633 RTCgetWeekDay

Function *RTCgetWeekDay* : byte; {0 = sunday}
Reads the weekdays register.

Group: [RTC Driver](#) 

4.634 RTCgetYear

Function *RTCgetYear* : byte;
Reads the years register.

Group: [RTC Driver](#) 

4.635 RTCsetDay

Procedure *RTCsetDay* (day : byte);
Sets the days register to "day".

Group: [RTC Driver](#) 

4.636 RTCsetHour

Procedure RTCsetHour (hour : byte);
Sets the hours register to "hour".

Group: [RTC Driver](#) 

4.637 RTCsetMinute

Procedure RTCsetMinute (min : byte);
Sets the minutes register to "min".

Group: [RTC Driver](#) 

4.638 RTCsetMonth

Procedure RTCsetMonth (month : byte);
Sets the months register to "month".

Group: [RTC Driver](#) 

4.639 RTCsetSecond

Procedure RTCsetSecond (sec : byte);
Sets the seconds register to "sec".

Group: [RTC Driver](#) 

4.640 RTCsetWeekDay

Procedure RTCsetWeekDay (wday : byte);
Sets the weekdays register to "wday". 0 = sunday.

Group: [RTC Driver](#) 

4.641 RTCsetYear

Procedure RTCsetYear (year : byte);
Sets the years register to "year".

Group: [RTC Driver](#) 

4.642 RTCTickHour

Procedure RTCTickHour;
CallBack. Is called at every hour overflow.

Group: [RTC Driver](#) 

4.643 RTCTickMinute

Procedure RTCTickMinute;

CallBack. Is called at every minute overflow.

Group: [RTC Driver](#) 

4.644 RTCTickSecond

Procedure RTCTickSecond;

CallBack. Is called at every second overflow.

Group: [RTC Driver](#) 

4.645 RTCTimer

Procedure RTCTimer;

Procedure RTCTimer (chan : byte);

CallBack. Is called when Downcounter = 0.

Group: [RTC Driver](#) 

4.646 RTCTimer_Load

Procedure RTCTimer_Load ([chan : byte;] seconds : word[longword]);

Sets the time compare register of RTCTimerX. Compare is stoped.

Group: [RTC Driver](#) 

4.647 RTCTimer_Start

Procedure RTCTimer_Start [(chan : byte)];

Starts the RTCTimer.

Group: [RTC Driver](#) 

4.648 RTCTimer_Stop

Procedure RTCTimer_Stop[(chan : byte)];

Stops the RTCTimer.

Group: [RTC Driver](#) 

4.649 RunErr

Function RunErr : boolean;

Returns the current runtime error.

Group: [System](#) 

4.650 RunTimeErr

procedure RunTimeErr;

CallBack. Is called when an error occurs.

Group: [System](#) 

4.651 Schedule

Procedure Schedule;

The process/task is suspended here.

Group: [MultiTasking](#) 

4.652 SchedulerOff

Procedure SchedulerOff;

The process scheduler is stopped.

Group: [MultiTasking](#) 

4.653 SchedulerOn

Procedure SchedulerOn;

The process scheduler is started.

Group: [MultiTasking](#) 

4.654 sDelay

Procedure sDelay (d : byte);

Software delay in CPU cycles.

Group: [System](#) 

4.655 SemaStat

Function SemaStat (s : semaphore) : byte;

Returns the content of a semaphore.

Group: [System](#) 

4.656 SendRC5

Procedure SendRC5 (const txAdr : byte; const txCmd : byte);

Sends the two bytes via the IR transmitter.

Group: [RC5 Driver](#) 

4.657 Ser_Enable

```
Procedure Ser_Enable1 (const ena : boolean);  
Procedure Ser_Enable2 (const ena : boolean);  
...
```

XMega:

```
Procedure Ser_EnableC0 (const ena : boolean);  
Procedure Ser_EnableC1 (const ena : boolean);  
Procedure Ser_EnableD0 (const ena : boolean);  
...
```

Is used to control the RS485 line driver.

Note: *Procedure* Ser_Enable (...) has been replaced by *Procedure* Ser_Enable1. For compatibility reasons it is still available.

Group: [SerPorts](#) 

4.658 SerBaud

```
Procedure SerBaud1 (var baud : word);  
Procedure SerBaud2 (var baud : word);  
...
```

XMega:

```
Procedure SerBaudC0 (var baud : word);  
Procedure SerBaudC1 (var baud : word);  
Procedure SerBaudD0 (var baud : word);  
...
```

Selects the baud rate of the SerPorts.

Note: *Procedure* SerBaud (...) has been replaced by *Procedure* SerBaud1. For compatibility reasons it is still available.

Group: [SerPorts](#) 

4.659 SetSerBaud

XMega:

```
Procedure SetSerBaud (UsartC0, var baud : word);  
Procedure SetSerBaud (UsartC1, var baud : word);  
Procedure SetSerBaud (UsartD0, var baud : word);  
...
```

Sets the baudrate for serial ports.

Group: [SerPorts](#) 

4.660 SerDataBits

Procedure SerDataBits1 (bits : tDataBits);
Procedure SerDataBits2 (bits : tDataBits);
 ...

XMega:

Procedure SerDataBitsC0 (bits : tDataBits);
Procedure SerDataBitsC1 (bits : tDataBits);
Procedure SerDataBitsD0 (bits : tDataBits);
 ...

Sets the number of databits.

Note: *Procedure SerDataBits (...)* has been replaced by *Procedure SerDataBits1*. For compatibility reasons it is still available.

Group: [SerPorts](#) 

4.661 SerInp

Function SerInp1 : byte|char;
Function SerInp2 : byte|char;
 ...

XMega:

Function SerInpC0 : byte|char;
Function SerInpC1 : byte|char;
Function SerInpD0 : byte|char;
 ...

Reads the serial interface or buffer.

Note: *Function SerInp* has been replaced by *Function SerInp1*. For compatibility reasons it is still available.

Group: [SerPorts](#) 

4.662 SerInp_TO

Function SerInp_TO1 (var rx : char|byte; const timeout : byte) : boolean;
Function SerInp_TO2 (var rx : char|byte; const timeout : byte) : boolean;
 ...

XMega:

Function SerInp_TOC0 (var rx : char|byte; const timeout : byte) : boolean;
Function SerInp_TOC1 (var rx : char|byte; const timeout : byte) : boolean;
Function SerInp_TOD0 (var rx : char|byte; const timeout : byte) : boolean;
 ...

Reads the serial interface or buffer with timeout.

Note: *Function SerInp_TO (...)* has been replaced by *Function SerInp_TO1 (...)*. For compatibility reasons it is still available.

Group: [SerPorts](#) 

4.663 SerInpBlock

Procedure SerInpBlock1 (**var** location: type);
Procedure SerInpBlock2 (**var** location: type);
 ...

XMega:

Procedure SerInpBlockC0 (**var** location: type);
Procedure SerInpBlockC1 (**var** location: type);
Procedure SerInpBlockD0 (**var** location: type);
 ...

Receive a block of data.

Note: *Procedure* SerInpBlock (...) has been replaced by *Procedure* SerInpBlock1. For compatibility reasons it is still available.

Group: [SerPorts](#) 

4.664 SerInpBlock_P

Procedure SerInpBlock1_P (p : pointer; len : word);
Procedure SerInpBlock2_P (p : pointer; len : word);
 ...

XMega:

Procedure SerInpBlockC0_P (p : pointer; len : word);
Procedure SerInpBlockC1_P (p : pointer; len : word);
Procedure SerInpBlockD0_P (p : pointer; len : word);
 ...

Receive a block of data with variable length.

Note: *Procedure* SerInpBlock_P (...) has been replaced by *Procedure* SerInpBlock1_P (...). For compatibility reasons it is still available.

Group: [SerPorts](#) 

4.665 SerInpBlock_TO

Function SerInpBlock_TO1 (**var** location : type; **const** TimeOut : byte) : boolean;
Function SerInpBlock_TO2 (**var** location : type; **const** TimeOut : byte) : boolean;
 ...

XMega:

Function SerInpBlock_TOC0 (**var** location : type; **const** TimeOut : byte) : boolean;
Function SerInpBlock_TOC1 (**var** location : type; **const** TimeOut : byte) : boolean;
Function SerInpBlock_TOD0 (**var** location : type; **const** TimeOut : byte) : boolean;
 ...

Receive a block of data with timeout.

Note: *Function SerInpBlock_TO (...)* has been replaced by *Function SerInpBlock_TO1 (...)*. For compatibility reasons it is still available.

Group: [SerPorts](#) ²⁹

4.666 SerInpBlockP_TO

Function SerInpBlockP_TO1 (p : pointer; len : word; to : byte) : boolean;
Function SerInpBlockP_TO2 (p : pointer; len : word; to : byte) : boolean;
 ...

XMega:

Function SerInpBlockP_TOC0 (p : pointer; len : word; to : byte) : boolean;
Function SerInpBlockP_TOC1 (p : pointer; len : word; to : byte) : boolean;
Function SerInpBlockP_TOD0 (p : pointer; len : word; to : byte) : boolean;
 ...

Reads the serial interface with variable length and timeout.

Note: *Function SerInpBlockP_TO (...)* has been replaced by *Function SerInpBlockP_TO1 (...)*. For compatibility reasons it is still available.

Group: [SerPorts](#) ²⁹

4.667 SerInpSLIP

Function SerInpSLIP1 (dst : pointer; tmo : byte; count : word) : word;
Function SerInpSLIP2 (dst : pointer; tmo : byte; count : word) : word;
 ...

XMega:

Function SerInpSLIPC0(dst : pointer; tmo : byte; count : word) : word; // UARTC0
Function SerInpSLIPC1(dst : pointer; tmo : byte; count : word) : word; // UARTC1
Function SerInpSLIPD0(dst : pointer; tmo : byte; count : word) : word; // UARTD0
 ...

Receives a SLIP package.

Note: *Function SerInpSLIP (...)* has been replaced by *Function SerInpSLIP1 (...)*. For compatibility reasons it is still available.

Group: [SerPorts](#) ²⁹

4.668 SerOut

Procedure SerOut1 (ch : byte|char);
Procedure SerOut2 (ch : byte|char);
 ...

XMega:

Procedure SerOutC0 (ch : byte|char);
Procedure SerOutC1 (ch : byte|char);
Procedure SerOutD0 (ch : byte|char);
 ...

Writes to the send buffer.

Note: *Procedure SerOut (...)* has been replaced by *Procedure SerOut1 (...)*. For compatibility reasons it is still available.

Group: [SerPorts](#) 

4.669 SerOutBlock

Procedure SerOutBlock1 (const location: type);
Procedure SerOutBlock2 (const location: type);
 ...

XMega:

Procedure SerOutBlockC0 (const location: type);
Procedure SerOutBlockC1 (const location: type);
Procedure SerOutBlockD0 (const location: type);
 ...

Sends a block of data.

Note: *Procedure SerOutBlock (...)* has been replaced by *Procedure SerOutBlock1 (...)*. For compatibility reasons it is still available.

Group: [SerPorts](#) 

4.670 SerOutBlock_P

Procedure SerOutBlock1_P (p : pointer; count : word);
Procedure SerOutBlock2_P (p : pointer; count : word);
 ...

XMega:

Procedure SerOutBlockC0_P (p : pointer; count : word);
Procedure SerOutBlockC1_P (p : pointer; count : word);
Procedure SerOutBlockD0_P (p : pointer; count : word);
 ...

Sends a block of data with variable length.

Note: *Procedure SerOutBlock_P (...)* has been replaced by *Procedure SerOutBlock1_P (...)*. For compatibility reasons it is still available.

Group: [SerPorts](#) 

4.671 SerOutSLIP

Procedure SerOutSLIP1 (src : pointer; count : word);
Procedure SerOutSLIP2 (src : pointer; count : word);
 ...

XMega:

Procedure SerOutSLIPC0(src : pointer; count : word); // UARTC0
Procedure SerOutSLIPC1(src : pointer; count : word); // UARTC1

Procedure *SerOutSLIPD0*(src : pointer; count : word); // UARTD0
...

Creates and sends a SLIP package.

Note: **Procedure** *SerOutSLIP* (...) has been replaced
by **Procedure** *SerOutSLIP1* (...). For compatibility reasons it is still available.

Group: [SerPorts](#) 

4.672 SerParity

Procedure *SerParity1* (par : tParity);
Procedure *SerParity2* (par : tParity);
...

XMega:

Procedure *SerParityC0* (par : tParity);
Procedure *SerParityC1* (par : tParity);
Procedure *SerParityD0* (par : tParity);
...

Selects the parity bit, even, odd, none.

Note: **Procedure** *SerParity* (...) has been replaced
by **Procedure** *SerParity1* (...). For compatibility reasons it is still available.

Group: [SerPorts](#) 

4.673 SerPort_Send

Procedure *SerPort_Send1*;
Procedure *SerPort_Send2*;
...

XMega:

Procedure *SerPort_SendC0*;
Procedure *SerPort_SendC1*;
Procedure *SerPort_SendD0*;
...

Frees a lock (DTR).

Note: **Procedure** *SerPort_Send* has been replaced
by **Procedure** *SerPort_Send1*. For compatibility reasons it is still available.

Group: [SerPorts](#) 

4.674 SerStat

Function *SerStat1* : boolean;
Function *SerStat2* : boolean;
...

XMega:**Function** SerStatC0 : boolean;**Function** SerStatC1 : boolean;**Function** SerStatD0 : boolean;

...

Returns TRUE, if a character exists.

Note: **Function** SerStat has been replaced by **Function** SerStat1. For compatibility reasons it is still available.

Group: [SerPorts](#) 

4.675 SerStopBits

Procedure SerStopBits1 (bits : tStopBits);**Procedure** SerStopBits2 (bits : tStopBits);

...

XMega:**Procedure** SerStopBitsC0 (bits : tStopBits);**Procedure** SerStopBitsC1 (bits : tStopBits);**Procedure** SerStopBitsD0 (bits : tStopBits);

...

Sets the number of stopbits.

Note: **Procedure** SerStopBits (...) has been replaced by **Procedure** SerStopBits1 (...). For compatibility reasons it is still available.

Group: [SerPorts](#) 

4.676 SetAdcFixed

Procedure SetAdcFixed (fix : boolean; chan : byte);

Converts continuously a single channel (fix = true).

Attention: AVRco counts the channels 1, 2, 3, ... (corresponding to controller count 0, 1, 2, ...)

Group: [ADC](#) 

4.677 SetAVfilter

Function SetAVfilter (var Filter : AVfilter; val : type) : type;

Replaces the oldest entry with "val" and returns with the new build average value.

Group: [Diverse](#) 

4.678 SetBit

Procedure SetBit (BitType, boolean);

Sets or resets the bit.

Group: [System](#)⁷

4.679 SetDacA

Function SetDacA_chan0 (val : word);

Function SetDacA_chan1 (val : word);

Writes a new output value (lower 12bit) to the DAC register.

Group: [DA Converter](#)⁹

4.680 SetDacB

Function SetDacB_chan0 (val : word);

Function SetDacB_chan1 (val : word);

Writes a new output value (lower 12bit) to the DAC register.

Group: [DA Converter](#)⁹

4.681 SetDeviceLock

Function SetDeviceLock (d : DeviceLock) : boolean;

Returns true if the device was free.

Group: [Diverse](#)⁵

4.682 SetFreqCountMode

Procedure SetFreqCountMode (mode : tFreqCountMode);

The parameter *mode* is the desired resolution.

Group: [Frequency Counter/Timer](#)¹²

4.683 SetFreqCountMode2

Procedure SetFreqCountMode2(mode : tFreqCountMode);

The parameter *mode* is the desired resolution.

Group: [Frequency Counter/Timer](#)¹²

4.684 SetIncrementVal

Procedure SetIncrementVal (val : integer [longint]);

Resets the absolute internal counter to "val".

Group: [Increment Counter](#)¹²

4.685 SetIncrVal4

Procedure *SetIncrVal4* (*chan* : byte; *val* : integer [longint]);
Resets the absolute internal counter at *chan* to "val".

Group: [Increment Counter 4chan](#) 

4.686 SetLength

Procedure *SetLength* (*st* : string; *len* : byte);
Changes the length of a string to the desired length.

Group: [Strings](#) 

4.687 SetMSPIClkPha

Procedure *SetMSPIClkPha0* (*phase* : byte);
Procedure *SetMSPIClkPha1* (*phase* : byte);
...

Group: [ATMega MSPI HardwareDriver](#) 

XMega:

Procedure *SetMSPIClkPha_C0* (*phase* : byte);
Procedure *SetMSPIClkPha_C1* (*phase* : byte);
Procedure *SetMSPIClkPha_D0* (*phase* : byte);
...

Group: [XMega MSPI HardwareDriver](#) 

Selects the clock phase. *phase* = 0/1.

4.688 SetMSPIClkPol

Procedure *SetMSPIClkPol0* (*pol* : byte);
Procedure *SetMSPIClkPol1* (*pol* : byte);
...

Group: [ATMega MSPI HardwareDriver](#) 

XMega:

Procedure *SetMSPIClkPol_C0* (*pol* : byte);
Procedure *SetMSPIClkPol_C1* (*pol* : byte);
Procedure *SetMSPIClkPol_D0* (*pol* : byte);
...

Group: [XMega MSPI HardwareDriver](#) 

Selects the clock polarity. *pol* = 0/1

4.689 SetMSPImode

Procedure *SetMSPI0mode* (*mode* : byte);
Procedure *SetMSPI1mode* (*mode* : byte);
...

Group: [ATMega MSPI HardwareDriver](#) 

XMega:*Procedure* SetMSPImode_C0 (mode : byte);*Procedure* SetMSPImode_C1 (mode : byte);*Procedure* SetMSPImode_D0 (mode : byte);

...

Group: [XMega MSPI HardwareDriver](#)^[35]Selects the MSPI 0 mode. *mode* = 0, 1, 2 or 3.**4.690 SetMSPIorder***Procedure* SetMSPIorder0 (msb : boolean);*Procedure* SetMSPIorder1 (msb : boolean);

...

Group: [ATMega MSPI HardwareDriver](#)^[34]**XMega:***Procedure* SetMSPIorder_C0 (msb : boolean);*Procedure* SetMSPIorder_C1 (msb : boolean);*Procedure* SetMSPIorder_D0 (msb : boolean);

...

Group: [XMega MSPI HardwareDriver](#)^[35]

Selects the bit sequence, msb or lsb.

4.691 SetMSPIpresc*Procedure* SetMSPIpresc0 (presc : byte);*Procedure* SetMSPIpresc1 (presc : byte);

...

Group: [ATMega MSPI HardwareDriver](#)^[34]**XMega:***Procedure* SetMSPIpresc_C0 (presc : boolean);*Procedure* SetMSPIpresc_C1 (presc : boolean);*Procedure* SetMSPIpresc_D0 (presc : boolean);

...

Group: [XMega MSPI HardwareDriver](#)^[35]Selects the clock prescaler. *presc* = 0..255.**4.692 SetPWM***procedure* SetPWM_C0A (pw : byte|word);*procedure* SetPWM_C0B (pw : byte|word);*procedure* SetPWM_C0C (pw : byte|word);

...

only XMega

Writes a new value to the corresponding PWM Port.

pw depends on the corresponding Define *PWMres_xxx* (>8 Bit Word, else Byte)Group: [PWM Port](#)^[26]

4.693 SetSema

Procedure *SetSema* (*sema* : semaphore; *v* : byte);
Sets a semaphore.

Group: [System](#) 

4.694 SetServoChan

Procedure *SetServoChan* (*chan* : byte; *pulse* : integer);
Sets the servo position.

Group: [Servo Driver](#) 

4.695 SetServoOffs

Procedure *SetServoOffs* (*chan* : byte; *offs* : integer);
Changes the servo neutral position.

Group: [Servo Driver](#) 

4.696 SetSPIClkPha

Procedure *SetSPIClkPha* (*phase* : byte);

XMega:

Procedure *SetSPIClkPhaC* (*phase* : byte);

Procedure *SetSPIClkPhaD* (*phase* : byte);

Procedure *SetSPIClkPhaE* (*phase* : byte);

Procedure *SetSPIClkPhaF* (*phase* : byte);

Selects the clock phase. *phase* = 0/1.

Group: [SPI HardwareDriver](#) 

4.697 SetSPIClkPol

Procedure *SetSPIClkPol* (*pol* : byte);

XMega:

Procedure *SetSPIClkPolC* (*pol* : byte);

Procedure *SetSPIClkPolD* (*pol* : byte);

Procedure *SetSPIClkPolE* (*pol* : byte);

Procedure *SetSPIClkPolF* (*pol* : byte);

Selects the clock polarity. 0/1

Group: [SPI HardwareDriver](#) 

4.698 SetSPIdoubleSpeed

Procedure SetSPIdoubleSpeed (ds : boolean);

XMega:

Procedure SetSPIdoubleSpeedC (ds : boolean);

Procedure SetSPIdoubleSpeedD (ds : boolean);

Procedure SetSPIdoubleSpeedE (ds : boolean);

Procedure SetSPIdoubleSpeedF (ds : boolean);

Switches the SPI bit rate between double and standard.

Group: [SPI HardwareDriver](#) 

4.699 SetSPImode

Procedure SetSPImode (mode : byte);

XMega:

Procedure SetSPImodeC (mode : byte);

Procedure SetSPImodeD (mode : byte);

Procedure SetSPImodeE (mode : byte);

Procedure SetSPImodeF (mode : byte);

Selects the SPI mode. *mode* = 0, 1, 2 or 3.

Group: [SPI HardwareDriver](#) 

4.700 SetSPlorder

Procedure SetSPlorder (msb : boolean);

XMega:

Procedure SetSPlorderC (msb : boolean);

Procedure SetSPlorderD (msb : boolean);

Procedure SetSPlorderE (msb : boolean);

Procedure SetSPlorderF (msb : boolean);

Selects the bit sequence, msb or lsb.

Group: [SPI HardwareDriver](#) 

4.701 SetSPIpresc

Procedure SetSPIpresc (presc : byte);

XMega:

Procedure SetSPIprescC (presc : byte);

Procedure SetSPIprescD (presc : byte);

Procedure SetSPIprescE (presc : byte);

Procedure SetSPIprescF (presc : byte);

Selects the clock prescaler. *presc* = 0..3 -> 4/16/64/128.

Group: [SPI HardwareDriver](#) 

4.702 SetSysBlinkTimer

Procedure *SetSysBlinkTimer* (*t* : *byte*);

Changes the blinkrate at runtime. *t* = 2..255 in SysTicks.

Group: [SysLeds](#) 

4.703 SetSysTimer

Procedure *SetSysTimer* (*tm* : *SysTimer*; *time* : *byte|word*);

Set timer, parameter in SysTicks.

Group: [Diverse](#) 

4.704 SetSysTimerM

Procedure *SetSysTimerM* (*tm* : *SysTimer*; *time* : *byte|word*);

Set timer, parameter in milliseconds.

Group: [Diverse](#) 

4.705 SetTable

Procedure *SetTable* (*t* : *Table*; *index* : *byte*; *new* : *type*);

Changes a member in a LookUp-Table.

Group: [System](#) 

4.706 SetTWImode

Procedure *SetTWImode* (**const** *twimode* : *tTWInetmode*);

Switches between master and slave.

Group: [TWI Network](#) 

4.707 SetTWInodeAddr

Procedure *SetTWInodeAddr* (*sAddr* : *byte*);

Sets the network adress of the master or slave.

Group: [TWI Network](#) 

4.708 SetVectTabBoot

Procedure *SetVectTabBoot* (*boot* : *boolean*);

Switch of the vector tables (IVSEL, MCUCR/GICR).

Group: [System](#) 

4.709 Sgn

Function *Sgn* (**const** num : integer, int8, longin, word64, int64, fix64t, float) : type;

If the argument is > 0 the function returns a '1'.

If the argument is zero the function returns a zero otherwise a '-1'.

The result has the same type as the argument.

Group: [Maths](#) 

4.710 SHT11convState

Function *SHT11convState* : boolean;

Returns the conversion state.

Group: [SHT11 Driver](#) 

4.711 SHT11getHum

Function *SHT11getHum* : word;

Reads out the measuring result.

Group: [SHT11 Driver](#) 

4.712 SHT11getStatus

Function *SHT11getStatus* : byte;

Reads the Status/Control register.

Group: [SHT11 Driver](#) 

4.713 SHT11getTemp

Function *SHT11getTemp* : word;

Reads out the measuring result.

Group: [SHT11 Driver](#) 

4.714 SHT11setStatus

Procedure *SHT11setStatus* (s : byte);

Some special functions.

Group: [SHT11 Driver](#) 

4.715 SHT11softReset

Procedure *SHT11softReset*;

Reset the Status/Control register to it's internal default values.

Group: [SHT11 Driver](#) 

4.716 SHT11startHum

Procedure SHT11startHum;

Starts the humidity conversion.

Group: [SHT11 Driver](#) 

4.717 SHT11startTemp

Procedure SHT11startTemp;

Starts the temperature conversion.

Group: [SHT11 Driver](#) 

4.718 SHT11synchronize

Procedure SHT11synchronize;

Resets the 2-wire interface.

Group: [SHT11 Driver](#) 

4.719 Sign

Function Sign (const num : integer[int8, longint, word64, int64, fix64, float]) : boolean;

If the argument is positive the function returns a true otherwise a false.

Group: [Maths](#) 

4.720 Sin

Function Sin (w : float) : float;

Returns the sine of the argument.

Group: [Maths](#) 

4.721 SinD

Function SinD (w : float) : float;

Returns the sine of the argument.

Group: [Maths](#) 

4.722 SinInt

Function SinInt (angle, v : integer) : integer;

Returns the sine of the angle multiplied by the Integer argument.

Group: [Maths](#) 

4.723 SinInt16

Function *SinInt16* (*angle* : integer) : integer; // angle in 0.1deg
Calculates the sine of an angle, multiplies the result by 10000.

Group: [Maths](#) 

4.724 SizeOf

Function *SizeOf* (*x* : type) : word;
Returns the required memory of an object in bytes.

Group: [Diverse](#) 

4.725 Sleep

Procedure *Sleep* (*p* : process; *t* : word);
Process Sleep in Ticks.

Group: [System](#) 

4.726 SLIPgetRxCount

Function *SLIPgetRxCount1* : word;
Function *SLIPgetRxCount2* : word;
...

XMega:

Function *SLIPgetRxCountC0*: word;
Function *SLIPgetRxCountC1*: word;
Function *SLIPgetRxCountD0*: word;
...

Requests the byte count of the received packet if Rx-state is *SLIPready*.

Group: [SLIPport](#) 

4.727 SLIPgetRxState

Function *SLIPgetRxState1*: tSLIPstate;
Function *SLIPgetRxState2*: tSLIPstate;
...

XMega:

Function *SLIPgetRxStateC0*: tSLIPstate;
Function *SLIPgetRxStateC1*: tSLIPstate;
Function *SLIPgetRxStateD0*: tSLIPstate;
...

Returns the actual Rx-state of the driver.
Possible results are: *SLIPready*, *SLIPbusy*, *SLIPovr*, *SLIPtout*, *SLIPfrm*.

Group: [SLIPport](#) 

4.728 SLIPgetTxState

Function SLIPgetTxState1 : tSLIPstate;
Function SLIPgetTxState2 : tSLIPstate;
...

XMega:

Function SLIPgetTxStateC0 : tSLIPstate;
Function SLIPgetTxStateC1 : tSLIPstate;
Function SLIPgetTxStateD0 : tSLIPstate;
...

Returns the actual Tx-state of the driver.
Possible results are: *SLIPready*, *SLIPbusy*, *SLIPtout*.

Group: [SLIPport](#)³²

4.729 SLIPPresumeRx

Function SLIPPresumeRx1 : boolean;
Function SLIPPresumeRx2 : boolean;
...

XMega:

Function SLIPPresumeRxC0 : boolean;
Function SLIPPresumeRxC1 : boolean;
Function SLIPPresumeRxD0 : boolean;
...

Re-enable the receive operation. After a program start and each received packet, also faulty ones. If the Rx-state of the driver is *SLIPbusy* the function returns a false.

Group: [SLIPport](#)³²

4.730 SLIPrxReady

Function SLIPrxReady1 : boolean;
Function SLIPrxReady2 : boolean;
...

XMega:

Function SLIPrxReadyC0 : boolean;
Function SLIPrxReadyC1 : boolean;
Function SLIPrxReadyD0 : boolean;
...

returns a true if a packet is ready. Also faulty packets or time-outs return a true.
The status of the packet must be interrogated with [SLIPgetRxState1](#)

Group: [SLIPport](#)³²

4.731 SLIPsetMode

Function *SLIPsetMode1* : (sMode : tSLIPmode) : boolean;

Function *SLIPsetMode2* : (sMode : tSLIPmode) : boolean;

...

XMega:

Function *SLIPsetModeC0* : (sMode : tSLIPmode) : boolean;

Function *SLIPsetModeC1* : (sMode : tSLIPmode) : boolean;

Function *SLIPsetModeD0* : (sMode : tSLIPmode) : boolean;

...

Sets the operation mode of the driver via bitset parameter.

Bitnames: *slpHsk*, *slpChkS*, *slpAddr*

It is possible to configure it with any combination of handshake, checksum and address check.

The function returns with a false if either Rx or Tx is still busy.

Attention: in handshake mode the driver works in the half-duplex mode. This means that as long as a packet including the handshake is not complete processed no further packet can be send, from none of the participants!

Group: [SLIPport](#) 

4.732 SLIPsetRxAddr

Procedure *SLIPsetRxAddr1* (rxAddr : byte);

Procedure *SLIPsetRxAddr2* (rxAddr : byte);

...

XMega:

Procedure *SLIPsetRxAddrC0* (rxAddr : byte);

Procedure *SLIPsetRxAddrC1* (rxAddr : byte);

Procedure *SLIPsetRxAddrD0* (rxAddr : byte);

...

Defines the internal Rx-address. It is absolutely necessary id the *slpAddr* mode is selected.

This address can be changed at every time. It is ignored in the non-address modes.

Group: [SLIPport](#) 

4.733 SLIPsetRxBuffer

Function *SLIPsetRxBuffer1* (rxBuff : pointer; size : word) : boolean;

Function *SLIPsetRxBuffer2* (rxBuff : pointer; size : word) : boolean;

...

XMega:

Function *SLIPsetRxBufferC0* (rxBuff : pointer; size : word) : boolean;

Function *SLIPsetRxBufferC1* (rxBuff : pointer; size : word) : boolean;

Function *SLIPsetRxBufferD0* (rxBuff : pointer; size : word) : boolean;

...

Defines the address of the Rx-buffer and its size.

If an incoming packet is larger then it will be discarded and an overrun error is the result.

If the mode *slpChkS* is active then also the received checksum is placed into the Rx-buffer as the last byte. If the Rx-state of the driver is *SLIPbusy* the function returns a false.

Group: [SLIPport](#) 

4.734 SLIPsetTimeOut

Procedure *SLIPsetTimeOut1* (TimeOut : byte);

Procedure *SLIPsetTimeOut2* (TimeOut : byte);

...

XMega:

Procedure *SLIPsetTimeOutC0* (TimeOut : byte);

Procedure *SLIPsetTimeOutC1* (TimeOut : byte);

Procedure *SLIPsetTimeOutD0* (TimeOut : byte);

...

Defines the timeout for receive operations and the handshake (in SysTicks).
If the receive of a packet aborts in an unexpected way the driver waits until the timeout is expired and then the Rx-state is set to *SLIPtout*.
With the handshake mode the operation is also aborted with a timeout if the acknowledge of the receivers fails.

Group: [SLIPport](#) 

4.735 SLIPsetTxAddr

Procedure *SLIPsetTxAddr1* (txAddr : byte);

Procedure *SLIPsetTxAddr2* (txAddr : byte);

...

XMega:

Procedure *SLIPsetTxAddrC0* (txAddr : byte);

Procedure *SLIPsetTxAddrC1* (txAddr : byte);

Procedure *SLIPsetTxAddrD0* (txAddr : byte);

...

Defines the target Tx-address. It is absolutely necessary if the *slpAddr* mode is selected. This address can be changed at every time. It is ignored in the non-address modes.

Group: [SLIPport](#) 

4.736 SLIPsetTxBuffer

Function *SLIPsetTxBuffer1* (txBuff : pointer; size : word) : boolean;

Function *SLIPsetTxBuffer2* (txBuff : pointer; size : word) : boolean;

...

XMega:

Function *SLIPsetTxBufferC0* (txBuff : pointer; size : word) : boolean;

Function *SLIPsetTxBufferC1* (txBuff : pointer; size : word) : boolean;

Function *SLIPsetTxBufferD0* (txBuff : pointer; size : word) : boolean;

...

Defines the address of the Tx-buffer and its size. If the Tx-state of the driver is *SLIPbusy* then the function doesn't execute and returns a false.

Group: [SLIPport](#)³²

4.737 SLIPstartTx

Function SLIPstartTx1 : boolean;

Function SLIPstartTx2 : boolean;

...

XMega:

Function SLIPstartTxC0 : boolean;

Function SLIPstartTxC1 : boolean;

Function SLIPstartTxD0 : boolean;

...

Starts a transmit. The data residing in *txBuff* are sent with the length *size* (from SLIPsetTxBuffer) or *count* (from SLIPstartTxC).

If the Tx-state of the driver is *SLIPbusy* the function returns a false.

Group: [SLIPport](#)³²

4.738 SLIPstartTxC

Function SLIPstartTxC1 (count : word) : boolean;

Function SLIPstartTxC2 (count : word) : boolean;

...

XMega:

Function SLIPstartTxC_C0 (count : word) : boolean;

Function SLIPstartTxC_C1 (count : word) : boolean;

Function SLIPstartTxC_D0 (count : word) : boolean;

...

Starts a transmit. The data residing in *txBuff* are sent with the length *count*.

The value *count* then controls all further packet sizes until SLIPsetTxBuffer or SLIPstartTxC changes the packet size. If the Tx-state of the driver is *SLIPbusy* function returns a false.

Group: [SLIPport](#)³²

4.739 SLIPstopRx

Procedure SLIPstopRx1;

Procedure SLIPstopRx2;

...

XMega:

Procedure SLIPstopRxC0;

Procedure SLIPstopRxC1;

Procedure SLIPstopRxD0;

...

If the receiver was enabled by SLIPPresumeRx.. this procedure can disable

receiving at every time.

Group: [SLIPport](#)³²

4.740 SLIPwasBC

Function SLIPwasBC1 : boolean;

Function SLIPwasBC2 : boolean;

...

XMega:

Function SLIPwasBC_C0 : boolean;

Function SLIPwasBC_C1 : boolean;

Function SLIPwasBC_D0 : boolean;

...

Returns a true if the last received packet was a broadcast packet

Group: [SLIPport](#)³²

4.741 SoftPWMstart

Procedure SoftPWMstart;

The controlling Timer will be restarted and all PWM values are reset to 0

Group: [SoftwarePWM](#)³⁴

4.742 SoftPWMstop

Procedure SoftPWMstop;

The controlling Timer will be stopped and all PWM pins become idle/inactive.

Group: [SoftwarePWM](#)³⁴

4.743 SpeechOutFlash

Procedure SpeechOutFlash (start : pointer; count : word);

Replays a data block out of the Flash.

Group: [Speech Port](#)³³

4.744 SpeechOutRAM

Procedure SpeechOutRAM (start : pointer; count : word);

Replays a data block out of the Ram.

Group: [Speech Port](#)³³

4.745 SpeechReady

Function *SpeechReady* : boolean;

Returns a true if the current output/replay operation is completed. Useful with double buffering.

Group: [Speech Port](#) ³³

4.746 SpeechStop

Procedure *SpeechStop*;

Aborts a running replay operation.

Group: [Speech Port](#) ³³

4.747 SPlinOut

Procedure *SPlinOut* (source, dest : pointer; count : word);

XMega:

Procedure *SPlinOutC* (source, dest : pointer; count : word);

Procedure *SPlinOutD* (source, dest : pointer; count : word);

Procedure *SPlinOutE* (source, dest : pointer; count : word);

Procedure *SPlinOutF* (source, dest : pointer; count : word);

Reads and writes a block of data.

Group: [SPI HardwareDriver](#) ³⁶

4.748 SPlinOut1

Procedure *SPlinOut1* (source, dest : pointer; count : word);

Read and writes a block of data from/into the SPI-slave.

Group: [SPI SoftDriver](#) ³⁷

4.749 SPlinOut2

Procedure *SPlinOut2* (source, dest : pointer; count : word);

Read and writes a block of data from/into the SPI-slave.

Group: [SPI SoftDriver](#) ³⁷

4.750 SPlinOutByte

Function *SPlinOutByte* (b : byte) : byte;

XMega:

Function *SPlinOutByteC* (b : byte) : byte;

Function *SPlinOutByteD* (b : byte) : byte;

Function *SPlinOutByteE* (b : byte) : byte;

Function *SPlinOutByteF* (b : byte) : byte;

Reads and writes a byte into the SPI-slave.

Group: [SPI HardwareDriver](#)^[36]

4.751 SPlinOutByte1

Function *SPlinOutByte1* (*b* : byte) : byte;

Read and writes a byte from/into the SPI-slave.

Group: [SPI SoftDriver](#)^[37]

4.752 SPlinOutByte2

Function *SPlinOutByte2* (*b* : byte) : byte;

Read and writes a byte from/into the SPI-slave.

Group: [SPI SoftDriver](#)^[37]

4.753 SPlinp

Function *SPlinp* : byte;

XMega:

Function *SPlinpC* : byte;

Function *SPlinpD* : byte;

Function *SPlinpE* : byte;

Function *SPlinpF* : byte;

Reads a byte from the RxBuffer.

Group: [SPI Network](#)^[37]

Procedure *SPlinp* (*dest* : pointer; *count* : word);

XMega:

Procedure *SPlinpC* (*dest* : pointer; *count* : word);

Procedure *SPlinpD* (*dest* : pointer; *count* : word);

Procedure *SPlinpE* (*dest* : pointer; *count* : word);

Procedure *SPlinpF* (*dest* : pointer; *count* : word);

Reads a block of data from the SPI-slave.

Group: [SPI HardwareDriver](#)^[36]

4.754 SPlinp1

Procedure *SPlinp1* (*dest* : pointer; *count* : word);

Reads a block of data from the SPI-slave into the memory location *dest* with the length *count*.

Group: [SPI SoftDriver](#)^[37]

4.755 SPlinp2

Procedure *SPlinp2* (*dest* : pointer; *count* : word);

Reads a block of data from the SPI-slave into the memory location *dest* with the length *count*.

Group: [SPI SoftDriver](#)³⁷

4.756 SPlinpByte

Function *SPlinpByte* : byte;

XMega:

Function *SPlinpByteC* : byte;

Function *SPlinpByteD* : byte;

Function *SPlinpByteE* : byte;

Function *SPlinpByteF* : byte;

Reads a byte from the SPI-slave.

Group: [SPI HardwareDriver](#)³⁶

4.757 SPlinpByte1

Function *SPlinpByte1* : byte;

Reads a byte from the SPI-slave.

Group: [SPI SoftDriver](#)³⁷

4.758 SPlinpByte2

Function *SPlinpByte2* : byte;

Reads a byte from the SPI-slave.

Group: [SPI SoftDriver](#)³⁷

4.759 SPlinpLong

Function *SPlinpLong* : longword;

XMega:

Function *SPlinpLongC* : longword;

Function *SPlinpLongD* : longword;

Function *SPlinpLongE* : longword;

Function *SPlinpLongF* : longword;

Reads a longword from the SPI-slave.

Group: [SPI HardwareDriver](#)³⁶

4.760 SPlinpLong1

Function *SPlinpLong1* : longword;

Reads a longword from the SPI-slave.

Group: [SPI SoftDriver](#)³⁷

4.761 SPlinpLong2

Function *SPlinpLong2* : longword;

Reads a longword from the SPI-slave.

Group: [SPI SoftDriver](#)³⁷

4.762 SPlinpLong64

Function *SPlinpLong64C* : longword;

Function *SPlinpLong64D* : longword;

Function *SPlinpLong64E* : longword;

Function *SPlinpLong64F* : longword;

only **XMega**

Reads a 64bit longword from the SPI-slave.

Group: [SPI HardwareDriver](#)³⁶

4.763 SPlinpWord

Function *SPlinpWord* : word;

XMega:

Function *SPlinpWordC* : word;

Function *SPlinpWordD* : word;

Function *SPlinpWordE* : word;

Function *SPlinpWordF* : word;

Reads a word from the SPI-slave.

Group: [SPI HardwareDriver](#)³⁶

4.764 SPlinpWord1

Function *SPlinpWord1* : word;

Reads a word from the SPI-slave.

Group: [SPI SoftDriver](#)³⁷

4.765 SPlinpWord2

Function *SPlinpWord2* : word;
Reads a word from the SPI-slave.

Group: [SPI SoftDriver](#)^[37]

4.766 SPlout

Function *SPlout* (const b : byte) : boolean;

XMega:

Procedure *SPloutC* (const b : byte);
Procedure *SPloutD* (const b : byte);
Procedure *SPloutE* (const b : byte);
Procedure *SPloutF* (const b : byte);

Writes a byte into the TxBuffer.

Group: [SPI Network](#)^[37]

Procedure *SPlout* (source : pointer; count : word);

XMega:

Function *SPloutC* (source : pointer; count : word) : boolean;
Function *SPloutD* (source : pointer; count : word) : boolean;
Function *SPloutE* (source : pointer; count : word) : boolean;
Function *SPloutF* (source : pointer; count : word) : boolean;

Writes a byte into the SPI-slave.

Group: [SPI HardwareDriver](#)^[36]

4.767 SPlout1

Procedure *SPlout1* (source : pointer; count : word);
Writes a block of data into the SPI-slave.

Group: [SPI SoftDriver](#)^[37]

4.768 SPlout2

Procedure *SPlout2* (source : pointer; count : word);
Writes a block of data into the SPI-slave.

Group: [SPI SoftDriver](#)^[37]

4.769 SPloutByte

Procedure SPloutByte (b : byte);

XMega:

Procedure SPloutByteC (b : byte);

Procedure SPloutByteD (b : byte);

Procedure SPloutByteE (b : byte);

Procedure SPloutByteF (b : byte);

Writes a byte into the SPI-slave.

Group: [SPI HardwareDriver](#)³⁶

4.770 SPloutByte1

Procedure SPloutByte1 (b : byte);

Writes a byte into the SPI-slave.

Group: [SPI SoftDriver](#)³⁷

4.771 SPloutByte2

Procedure SPloutByte2 (b : byte);

Writes a byte into the SPI-slave.

Group: [SPI SoftDriver](#)³⁷

4.772 SPloutLong

Procedure SPloutLong (L : longword);

XMega:

Procedure SPloutLongC (L : longword);

Procedure SPloutLongD (L : longword);

Procedure SPloutLongE (L : longword);

Procedure SPloutLongF (L : longword);

Writes a longword into the SPI-slave.

Group: [SPI HardwareDriver](#)³⁶

4.773 SPloutLong1

Procedure SPloutLong1 (L : longword);

Writes a longword into the SPI-slave.

Group: [SPI SoftDriver](#)³⁷

4.774 SPIoutLong2

Procedure SPIoutLong2 (L : longword);
Writes a longword into the SPI-slave.

Group: [SPI SoftDriver](#)^[37]

4.775 SPIoutLong64

Function SPIoutLong64C : longword;
Function SPIoutLong64D : longword;
Function SPIoutLong64E : longword;
Function SPIoutLong64F : longword;

only **XMega**
Writes a 64bit longword into the SPI-slave.

Group: [SPI HardwareDriver](#)^[36]

4.776 SPIoutWord

Procedure SPIoutWord (w : word);

XMega

Procedure SPIoutWordC (w : word);
Procedure SPIoutWordD (w : word);
Procedure SPIoutWordE (w : word);
Procedure SPIoutWordF (w : word);

Writes a word into the SPI-slave.

Group: [SPI HardwareDriver](#)^[36]

4.777 SPIoutWord1

Procedure SPIoutWord1 (w : word);
Writes a word into the SPI-slave.

Group: [SPI SoftDriver](#)^[37]

4.778 SPIoutWord2

Procedure SPIoutWord2 (w : word);
Writes a word into the SPI-slave.

Group: [SPI SoftDriver](#)^[37]

4.779 SPIrxClear

Function *SPIrxClear* : *boolean*;

The read and write pointers of the RxBuffer are cleared.

Group: [SPI Network](#) 

4.780 SPIrxFrame

Function *SPIrxFrame* : *boolean*;

Receives a Frame/Packet.

Group: [SPI Network](#) 

4.781 SPIrxStat

Function *SPIrxStat* : *boolean*;

Returns a true if a frame was received.

Group: [SPI Network](#) 

4.782 SPItxClear

Procedure *SPItxClear*;

The read and write pointers of the TxBuffer are cleared.

Group: [SPI Network](#) 

4.783 SPItxFrame

Function *SPItxFrame* (*const len* : *byte*) : *boolean*;

Transmits a Frame/Packet.

Group: [SPI Network](#) 

4.784 SPItxStat

Function *SPItxStat* : *boolean*;

Checks whether a frame to send is sent.

Group: [SPI Network](#) 

4.785 Sqr

Function *Sqr* (*f* : *float*) : *float*;

Function *Sqr* (*f* : *fix64*) : *fix64*;

Returns the square of the argument.

Group: [Maths](#) 

4.786 Sqrt

Function *Sqrt* (f : float) : float;
Function *Sqrt* (i : LongInt) : LongInt;
Function *Sqrt* (i : Int64) : Int64;
Function *Sqrt* (f : float) : float;
Function *Sqrt* (f : fix64) : fix64;
Returns the square root of the argument.

Note:

Sqrt (fix64) needs 600usec @16MHz, precision: 5 frac digits

[Fix64Sqrt](#) needs 2.2msec @16MHz, precision: 9 frac digits
(only in AVRco Profi version available)

Group: [Maths](#) 

4.787 SquareDivByte

Function *SquareDivByte* (val, divfact : byte) : byte;
Calculates the square of a value and divide the result by another value.

Group: [Maths](#) 

4.788 SquareDivInt

Function *SquareDivInt* (val, divfact : word|integer) : word|integer;
Calculates the square of a value and divide the result by another value.

Group: [Maths](#) 

4.789 SquareDivInt8

Function *SquareDivInt* (val, divfact : int8) : int8;
Calculates the square of a value and divide the result by another value.

Group: [Maths](#) 

4.790 Start_Processes

Procedure *Start_Processes*;
Starts Interrupts, Processes and Tasks.

Xmega

extended with a parameter which defines the enabled/disabled Interrupt Levels.

Procedure *Start_Processes* (level : byte);

Group: [System](#) 

4.791 StepDestCCW

Procedure StepDestCCW;

Step up ramp counter clock wise.

Group: [Stepper Driver](#) 

4.792 StepDestCW

Procedure StepDestCW;

Step up ramp clock wise.

Group: [Stepper Driver](#) 

4.793 StepOneCCW

Procedure StepOneCCW;

Step one step counter clock wise.

Group: [Stepper Driver](#) 

4.794 StepOneCW

Procedure StepOneCW;

Step one step clock wise.

Group: [Stepper Driver](#) 

4.795 StepPanicStop

Procedure StepPanicStop

Immediately stops a ramp or destination move.

Group: [Stepper Driver](#) 

4.796 StepperOff

Procedure StepperOff;

Switches the powerstage inactive (removes the current from motor).

Group: [Stepper Driver](#) 

4.797 StepperOn

Procedure StepperOn;

Switches the power stage active.

Group: [Stepper Driver](#) 

4.798 StepRampCCW

Procedure StepRampCCW;

Step up ramp counter clock wise.

Group: [Stepper Driver](#) 

4.799 StepRampCW

Procedure StepRampCW;

Step up ramp clock wise.

Group: [Stepper Driver](#) 

4.800 StepRampStop

Procedure StepRampStop;

Breaks a step ramp command.

Group: [Stepper Driver](#) 

4.801 StepVelocity

Function StepVelocity (v : word) : boolean;

Sets the current StepEndFreq.

Group: [Stepper Driver](#) 

4.802 StrClean

Function StrClean (const st : string; gt127 : boolean; subst : char) : string;

Removes/replaces control characters from a string.

Group: [Strings](#) 

4.803 StrReplace

Procedure StrReplace (src : string; var dest : string; pos : byte);

The string *src* overwrites the destination string *dest* at position *pos*.

Group: [Strings](#) 

4.804 StrToArr

Function StrToArr (var st : string) : array of char

Generation of Null-terminated strings (C Strings).

Group: [Strings](#) 

4.805 StrToFix64

Function *StrToFix64* (*st* : *string*) : *Fix64*;

convert a strings to a fix64.

Possible variables in RAM or EEPROM. Strings in Flash are not allowed.

Groups: [Strings](#)^[6], [Fix64](#)^[4]

4.806 StrToFloat

Function *StrToFloat* (*st* : *string*) : *float*;

Converts of a string into a float value.

Group: [Strings](#)^[6]

4.807 StrToInt

Function *StrToInt* (*st* : *string*) : *byte*|...|*int64*|*word64*;

Converts a string to a Int8, Byte, Word, Integer, etc.

Group: [Strings](#)^[6]

4.808 StrToIP

Procedure *StrToIP* (*IPstr* : *String*[15]; **var** *Result* : *TIPAddr*);

Converts an IP-address string "aaa.bbb.ccc.ddd" to a byte array.

Group: [Strings](#)^[6] [TINA TCP/IP](#)^[40] [WizNet TCP/IP](#)^[44]

4.809 StrToMac

Procedure *StrToMAC* (*MACstr* : *String*[17]; **var** *Result* : *TMACAddr*);

Converts a MAC-address string "aa:bb:cc:dd:ee:ff" to a Byte array.

Group: [Strings](#)^[6] [TINA TCP/IP](#)^[40] [WizNet TCP/IP](#)^[44]

4.810 Succ

Function *Succ* (*x* : *type*) : *type*;

Returns the next-higher value of a variable.

Group: [Maths](#)^[4]

4.811 Suspend

Procedure *Suspend* (*p* : *process*|*task*);

The Prozess/Task is suspended.

Group: [MultiTasking](#)^[5]

4.812 SuspendAll

Procedure *SuspendAll (Processes, Tasks);*

Procedure *SuspendAll (Processes);*

Procedure *SuspendAll (Tasks);*

All processes and/or tasks are stoped.

Group: [MultiTasking](#) 

4.813 Swap

Function *Swap (x : type) : type;*

Swaps LoByte with HiByte.

Group: [Maths](#) 

4.814 SwapIPAddr

Procedure *SwapIPAddr (var ip : TIPAddr);*

Mirrors an IP-address. Converts A3-A2-A1-A0 to A0-A1-A2-A3.

Group: [Diverse](#)  [TINA TCP/IP](#)  [WizNet TCP/IP](#) 

4.815 SwapLong

Function *SwapLong (const x : LongWord|LongInt) : LongWord|LongInt;*

The first and the last byte exchange their position. The same is true for the two middle bytes.

Group: [Maths](#) 

4.816 SwapMACAddr

Procedure *SwapMACAddr (var mac : TMACAddr);*

Mirrors a MAC-address. Converts A5-A4-A3-A2-A1-A0 to A0-A1-A2-A3-A4-A5.

Group: [Various](#)  [TINA TCP/IP](#)  [WizNet TCP/IP](#) 

4.817 SwitchKeyRepeat

Procedure *SwitchKeyRepeat1 (rept : boolean);*

Procedure *SwitchKeyRepeat2 (rept : boolean);*

Procedure *SwitchKeyRepeat_G (rept : boolean);*

Enable or disable the repeat function at runtime.

Group: [SwitchPorts](#) 

4.818 SwitchPort_Clear

procedure SwitchPort1_Clear;
procedure SwitchPort2_Clear;
procedure SwitchPort_G_Clear;
Resets the port.

Group: [SwitchPorts](#) 

4.819 SysLEDallOff

Procedure SysLEDallOff;
Switches all LEDs permant off.

Group: [SysLeds](#) 

4.820 SysLEDallOn

Procedure SysLEDallOn;
Switches all LEDs permanent on.

Group: [SysLeds](#) 

4.821 SysLEDenable

Procedure SysLEDenable (ena : boolean);
Enables/disables the driver.

Group: [SysLeds](#) 

4.822 SysLEDflashAllOff

Procedure SysLEDflashAllOff;
Stops blinking of all LEDs.

Group: [SysLeds](#) 

4.823 SysLEDflashAllOn

Procedure SysLEDflashAllOn;
Blinks all LEDs.

Group: [SysLeds](#) 

4.824 SysLEDflashMsg

Procedure SysLEDflashMsg (led, msg, rept : byte);
Blinks an error message.

Group: [SysLeds](#) 

4.825 SysLEDflashOff

Procedure *SysLEDflashOff (b : byte);*
Stops blinking of a LED.

Group: [SysLeds](#) 

4.826 SysLEDflashOn

Procedure *SysLEDflashOn (b : byte);*
Blinks a LED.

Group: [SysLeds](#) 

4.827 SysLEDflashOnce

Procedure *SysLEDflashOnce (b : byte);*
Blinks a LED once.

Group: [SysLeds](#) 

4.828 SysLEDflashOnOff

Procedure *SysLEDflashOnOff (b : byte; on : boolean); // b = 0..7 LED number*
Blinks a LED or stops blinking.

Group: [SysLeds](#) 

4.829 SysLEDOff

Procedure *SysLEDOff (b : byte);*
Switches a LED permant off.

Group: [SysLeds](#) 

4.830 SysLEDOn

Procedure *SysLEDOn (b : byte);*
Switches a LED permanent on.

Group: [SysLeds](#) 

4.831 SysLEDOnOff

Procedure *SysLEDOnOff (b : byte; on : boolean); // b= 0..7 LED number*
Switches a LED permanent on or off.

Group: [SysLeds](#) 

4.832 System_Init

Procedure System_Init;

CallBack. Is executed after the stack-initialisation.

Group: [System](#) 

4.833 System_MCUCR_Init

Procedure System_MCUCR_Init;

CallBack. Is executed after the power-on.

Group: [System](#) 

4.834 System_Reset

Procedure System_Reset;

Resets and restarts the whole system.

Group: [System](#) 

4.835 System_ShutDown

Procedure System_ShutDown;

Shuts the system down.

Group: [System](#) 

4.836 SysTickDisable

Procedure SysTickDisable;

Disables only timer interrupts. Timer keeps running.

Group: [System](#) 

4.837 SysTickEnable

Procedure SysTickEnable;

Enables only the SysTick timer.

Group: [System](#) 

4.838 SysTickRestart

Procedure SysTickRestart;

Re-initializes the hardware timer which triggers the SysTick.

Group: [System](#) 

4.839 SysTickStop

Procedure *SysTickStop*;

Disable Timer Interrupt, STOP Timer.

Group: [System](#) 

4.840 Tan

Function *Tan* (*w* : float) : float;

Returns the tangent of an angle w (in radians).

Group: [Maths](#) 

4.841 TanD

Function *TanD* (*w* : float) : float;

Returns the tangent of the angle w (in degrees).

Group: [Maths](#) 

4.842 TestDeviceLock

Function *TestDeviceLock* (*d* : DeviceLock) : boolean;

Checks a device lock state without changing it.

Group: [Various](#) 

4.843 TickTimer2OutEnable

Procedure *TickTimer2OutEnable* (*enable* : boolean; *Level* : byte);

Connects or disconnects the output pin to/from the timer.

Only for xMega available!

Group: [TickTimer](#) 

4.844 TickTimer2RawVal

Procedure *TickTimer2RawVal* (*presc* : byte; *cmp* : word|byte);

The prescaler and the compare register are written directly.

Only for xMega available!

Group: [TickTimer](#) 

4.845 TickTimer2Reload

Function *TickTimer2Reload* (*time* : longword) : boolean;

Changes the timing without stopping or disturbing the Timer.

Only for xMega available!

Group: [TickTimer](#) 

4.846 TickTimer2Start

Procedure *TickTimer2Start*;

Starts the timer and its interrupt, if present.

Only for xMega available!

Group: [TickTimer](#)⁴¹

4.847 TickTimer2Stop

Procedure *TickTimer2Stop*;

Stops the timer and its interrupt, if present.

Only for xMega available!

Group: [TickTimer](#)⁴¹

4.848 TickTimer2Time

Function *TickTimer2Time* (*time* : longword) : boolean;

Stops a running timer and calculates the new setting.

Only for xMega available!

Group: [TickTimer](#)⁴¹

4.849 TickTimerOutpEnable

Procedure *TickTimerOutpEnable* (*enable* : boolean; *Level* : byte);

Connects or disconnects the output pin to/from the timer.

Group: [TickTimer](#)⁴¹

4.850 TickTimerRawVal

Procedure *TickTimerRawVal* (*presc* : byte; *cmp* : word|byte);

The prescaler and the compare register are written directly.

Group: [TickTimer](#)⁴¹

4.851 TickTimerReload

Function *TickTimerReload* (*time* : longword) : boolean;

Changes the timing without stopping or disturbing the Timer.

Group: [TickTimer](#)⁴¹

4.852 TickTimerStart

Procedure *TickTimerStart*;

Starts the timer and its interrupt, if present.

Group: [TickTimer](#)⁴¹

4.853 TickTimerStop

Procedure *TickTimerStop*;

Stops the timer and its interrupt, if present.

Group: [TickTimer](#) 

4.854 TickTimerTime

Function *TickTimerTime* (*time* : longword) : boolean;

Stops a running timer and calculates the new setting.

Group: [TickTimer](#) 

4.855 TINA_Init

Function *TINA_Init* : Boolean;

Initializes the driver.

Group: [TINA TCP/IP](#) 

4.856 TINA_Ping

Function *TINA_Ping* (*PingAdr* : TIPAddr; *TimeOut* : Word) : Word;

Sends a ping.

Group: [TINA TCP/IP](#) 

4.857 TINA_Start

Procedure *TINA_Start*;

Starts the communication.

Group: [TINA TCP/IP](#) 

4.858 TINA_Stop

Procedure *TINA_Stop*;

Stops the communication.

Group: [TINA TCP/IP](#) 

4.859 TINACreateSocket

Function *TINACreateSocket* : tSocketHandle;

Creates a socket handle.

Group: [TINA TCP/IP](#) 

4.860 TINAfreeSocket

Procedure *TINAfreeSocket* (*SocketPtr* : *tSocketHandle*);
Frees a socket handle.

Group: [TINA TCP/IP](#) 

4.861 TINAinitSocket

Function *TINAinitSocket* (*SocketPtr* : *tSocketHandle*) : *Boolean*;
Initializes a socket.

Group: [TINA TCP/IP](#) 

4.862 TINALinkStat

Function *TINALinkStat* : *Boolean*;
Returns the link state of the ENC chip.

Group: [TINA TCP/IP](#) 

4.863 TINApacketReceived

Function *TINApacketReceived* (*SocketPtr* : *tSocketHandle*) : *Boolean*;
Asks for a new packet.

Group: [TINA TCP/IP](#) 

4.864 TINAresumeReceive

Function *TINAresumeReceive* (*SocketPtr* : *tSocketHandle*) : *Boolean*;
Resumes the receiver.

Group: [TINA TCP/IP](#) 

4.865 TINArxStat

Function *TINArxStat* (*SocketPtr* : *tSocketHandle*) : *Boolean*;
Returns the receive state of a socket. The function doesn't change any socket states.

Group: [TINA TCP/IP](#) 

4.866 TINAsendPacket

Function *TINAsendPacket* (*SocketPtr* : *tSocketHandle*; *Buffer* : *Pointer*; *Len* : *Word*) : *Boolean*;
Sends a packet.

Group: [TINA TCP/IP](#) 

4.867 TINASetPriority

Procedure *TINASetPriority* (*prio* : *TTinaPriority*);
Sets the priority of the TINA task.

Group: [TINA TCP/IP](#) 

4.868 Toggle

Procedure *Toggle* (*b* : *byte*; *v* : *byte*);

Procedure *Toggle* (*b* : *Bit*);

Toggles a bit.

Procedure *Toggle* (*SrcDest* : *BitSet*; *op* : *BitSet*);

Toggles bits in a bitset.

Group: [System](#) 

4.869 Trap

Procedure *Trap* (*t* : *byte*);

Calls a trap procedure.

Group: [System](#) 

4.870 Trim

Function *Trim* (**const** *st* : *string*) : *string*;

Removes leading and trailing spaces from the string.

Group: [Strings](#) 

4.871 TrimLeft

Function *TrimLeft* (**const** *st* : *string*) : *string*;

Removes leading spaces from the string.

Group: [Strings](#) 

4.872 TrimRight

Function *TrimRight* (**const** *st* : *string*) : *string*;

Removes trailing spaces from the string.

Group: [Strings](#) 

4.873 Trunc

Function *Trunc* (*f* : float|fix64) : integer; {LongInt}

Truncates a value of the type Float, Fix64 to an Integer value.

Group: [Maths](#) 

4.874 TWIgetBusy

Function *TWIgetBusy* : boolean;

Returns the state of the TWI-interface.

Group: [TWI \(I2C\) Port](#) 

4.875 TWIgetCmd

Function *TWIgetCmd* : byte;

Reads the last command issued by the Master.

Group: [TWI \(I2C\) Port](#) 

4.876 TWIgetRdy

Function *TWIgetRdy* : boolean;

Returns the state of the TWI-interface.

Group: [TWI \(I2C\) Port](#) 

4.877 TWIgetRxStat

Function *TWIgetRxStat* : boolean;

Checks whether there are new data received.

Group: [TWI \(I2C\) Port](#) 

4.878 TWIgetTxStat

Function *TWIgetTxStat* : boolean;

Checks whether the Master had data fetched from the Slave.

Group: [TWI \(I2C\) Port](#) 

4.879 TWIinp

Function *TWIinp* (*SlaveAdr* : byte; **var** *Data*) : boolean;

Reads at least one byte of the selected slave.

XMega

Function *TWIinpC* (*SlaveAdr* : byte; **var** *Data*) : boolean;

Function *TWIinpD* (*SlaveAdr* : byte; **var** *Data*) : boolean;

Function *TWIinpE* (*SlaveAdr* : byte; **var** *Data*) : boolean;

Function *TWlinpF* (*SlaveAdr* : byte; **var** *Data*) : boolean;

Group: [TWI \(I2C\) Port](#) 

4.880 TWlinpP

Function *TWlinpP* (*slv* : byte; *dst* : pointer; *len* : word) : boolean;

Reads at least one byte of the selected slave.

XMega

Function *TWlinpPC* (*SlaveAdr* : byte; *dst* : pointer; *len* : word) : boolean;

Function *TWlinpPD* (*SlaveAdr* : byte; *dst* : pointer; *len* : word) : boolean;

Function *TWlinpPE* (*SlaveAdr* : byte; *dst* : pointer; *len* : word) : boolean;

Function *TWlinpPF* (*SlaveAdr* : byte; *dst* : pointer; *len* : word) : boolean;

Group: [TWI \(I2C\) Port](#) 

4.881 TWlout

Function *TWlout* (*const SlaveAdr* : byte, *Command* : byte|word [, *Data*]) : boolean;

XMega

Function *TWloutC* (*SlaveAdr* : byte; *Command* : byte|word [, *Data*]) : boolean;

Function *TWloutD* (*SlaveAdr* : byte; *Command* : byte|word [, *Data*]) : boolean;

Function *TWloutE* (*SlaveAdr* : byte; *Command* : byte|word [, *Data*]) : boolean;

Function *TWloutF* (*SlaveAdr* : byte; *Command* : byte|word [, *Data*]) : boolean;

Writes at least one byte into the selected slave.

Group: [TWI \(I2C\) Port](#) 

4.882 TWloutP

Function *TWloutP* (*SlaveAdr* : byte; *Command* : byte; *src* : pointer; *len* : word) : boolean;

XMega

Function *TWloutPC* (*SlaveAdr* : byte; *Command* : byte; *src* : pointer; *len* : word) : boolean;

Function *TWloutPD* (*SlaveAdr* : byte; *Command* : byte; *src* : pointer; *len* : word) : boolean;

Function *TWloutPE* (*SlaveAdr* : byte; *Command* : byte; *src* : pointer; *len* : word) : boolean;

Function *TWloutPF* (*SlaveAdr* : byte; *Command* : byte; *src* : pointer; *len* : word) : boolean;

Writes at least one byte into the selected slave.

Group: [TWI \(I2C\) Port](#) 

4.883 TWloutWP

Function *TWloutWP* (*SlaveAdr* : byte; *Command* : word; *src* : pointer; *len* : word) : boolean;

XMega

Function *TWloutWPC* (*SlaveAdr* : byte; *Command* : byte; *src* : pointer; *len* : word) : boolean;

Function *TWloutWPD* (*SlaveAdr* : byte; *Command* : byte; *src* : pointer; *len* : word) : boolean;

Function *TWloutWPE* (*SlaveAdr* : byte; *Command* : byte; *src* : pointer; *len* : word) : boolean;

Function *TWloutWPF* (*SlaveAdr* : byte; *Command* : byte; *src* : pointer; *len* : word) : boolean;

Writes at least one byte into the selected slave.

Group: [TWI \(I2C\) Port](#) 

4.884 TWlrxClear

Procedure *TWlrxClear*;

The statusbyte of the RxBuffer is reset.

Group: [TWI Network](#) 

4.885 TWlrxFrame

Function *TWlrxFrame* (*node* : byte) : boolean;

Reads a frame from a Slave.

Group: [TWI Network](#) 

4.886 TWlrxStat

Function *TWlrxStat* : boolean;

Returns a true if a frame has been received completely.

Group: [TWI Network](#) 

4.887 TWlsetBusy

Function *TWlsetBusy* (*busy* : boolean) : boolean;

Disables or enables the buffers against Master accesses.

Group: [TWI \(I2C\) Port](#) 

4.888 TWlsetGC

Procedure *TWlsetGC* (*BroadcastEnable* : boolean);

Determines whether the Slave can receive Broadcasts.

Group: [TWI \(I2C\) Port](#) 

4.889 TWlsetRdy

Function *TWlsetRdy* (*ready* : boolean) : boolean;

Resets the state of the RxBuffer to empty and the TxBuffer to full.

Group: [TWI \(I2C\) Port](#) 

4.890 TWIsetSlaveAddr

Function *TWIsetSlaveAddr (SlaveAddr : byte);*
Changes the Slave address.

Group: [TWI \(I2C\) Port](#) 

4.891 TWIstat

Function *TWIstat (SlaveAdr : byte) : boolean;*

XMega

Function *TWIstatC (SlaveAdr : byte) : boolean;*

Function *TWIstatD (SlaveAdr : byte) : boolean;*

Function *TWIstatE (SlaveAdr : byte) : boolean;*

Function *TWIstatF (SlaveAdr : byte) : boolean;*

Returns a true if the chosen slave exists.

Group: [TWI \(I2C\) Port](#) 

4.892 TWItxBroadcast

Function *TWItxBroadcast (cmd : byte; subnode : byte) : boolean;*
Generates an I2C Broadcast to all Slaves.

Group: [TWI Network](#) 

4.893 TWItxClear

Procedure *TWItxClear;*
The statusbyte of the TxBuffer is reset.

Group: [TWI Network](#) 

4.894 TWItxFrame

Function *TWItxFrame (node : byte; len : byte[word]) : boolean;*
Sends a package.

Group: [TWI Network](#) 

4.895 TWItxStat

Function *TWItxStat : boolean;*
Checks whether a frame is already send or not.

Group: [TWI Network](#) 

4.896 uDelay

Procedure *uDelay* (*d* : byte);
Software Delay in usec x 10.

Group: [System](#) 

4.897 uDelay_1

Procedure *uDelay_1* (*d* : byte);
Software Delay in 1 usec.

Group: [System](#) 

4.898 UnLock

Procedure *UnLock* (*p* : process);
A locked process is released.

Group: [MultiTasking](#) 

4.899 UpCase

Function *UpCase* (*ch* : char) : char;
Converts a char into a capital.

Group: [Strings](#) 

4.900 UpperCase

Function *UpperCase* (*st* : string) : string;
Converts a string into capitals.

Group: [Strings](#) 

4.901 UsrDevPtr

Function *UsrDevPtr* (*p*:pointer): pointer;
Redirects the access to the User Device.

Group: [System](#) 

4.902 ValueInRange

Function *ValueInRange* (*value*, *vmin*, *vmax* : type) : boolean;
Compares the variable "value" with the two limits "vmin" and "vmax".
Type = enum, byte, int8, char, word, integer, longword, longint, word64, int64, fix64, float

Group: [Maths](#) 

4.903 ValueInTolerance

Function *ValueInTolerance* (*value, ref, tol : type*) : *boolean*;
Compares the content of "value" to the limits.

Group: [Maths](#) 

4.904 ValueInToleranceP

Function *ValueInToleranceP* (*value, ref : type; tol : byte*) : *boolean*;
Compares the content of "value" to the limits.

Group: [Maths](#) 

4.905 ValueTrimLimit

Function *ValueTrimLimit* (*value, vmin, vmax : type*) : *type*;
Returns the result in the limits *vmin/vmax*.
Type = enum, byte, int8, char, word, integer, longword, longint, word64, int64, fix64, float

Group: [Maths](#) 

4.906 WaitDeviceFree

Function *WaitDeviceFree* (*s : DeviceLock [; timeout: word]*) : *boolean*;
Process/Task will be suspended until the Device is released.

Group: [MultiTasking](#) 

4.907 WaitPipe

Function *WaitPipe* (*p : pipe [; timeout: word]*) : *boolean*;
A Process/Task becomes inactive until the pipe contains data.

Group: [MultiTasking](#)  [Pipes](#) 

4.908 WaitSema

Function *WaitSema* (*s : semaphore [; timeout: word]*) : *boolean*;
A Process/Task becomes inactive until the Semaphore is > 0.

Group: [MultiTasking](#) 

4.909 WatchDogStart

Procedure *WatchDogStart*;
Initializing and start of the WatchDog.

Group: [Diverse](#) 

4.910 WatchDogStop

Procedure *WatchDogStop*;
Stops the WatchDog.

Group: [Diverse](#) 

4.911 WatchDogTrig

Procedure *WatchDogTrig*;
Triggers the hardware WatchDog of the CPU.

Group: [Diverse](#) 

4.912 WithIn

Function *WithIn* (*lo, x, hi : type*) : *type*;
Checks against bounds. All types must be the same.
type: byte, int8, word, integer, longint, longword, int64, word64, fix64, float.

Group: [Maths](#) 

4.913 WordToBCD

Function *WordToBCD* (*w : word*) : *word*;
Converts a word value into the BCD format.

Group: [Maths](#) 

4.914 Write

Procedure *Write* (*DeviceFunc : function*; *arg : char|string*);
String output by a procedure.

Group: [Diverse](#) 

4.915 WriteLn

Procedure *WriteLn* (*DeviceFunc : function*; *arg : char|string*);
String output by a procedure. Appends CR/LF.

Group: [Diverse](#) 

4.916 wzAcceptConnection

Function *wzAcceptConnection* (*SocketPtr : tSocketHandle*; *YesNo : boolean*) : *boolean*;
If a Client is connecting, the connection can be accepted or denied.

Group: [WizNet TCP/IP](#) 

4.917 wzClientConnected

Function *wzClientConnected (SocketPtr : tSocketHandle) : boolean;*
A **Server** must poll this function until a true is returned.

Group: [WizNet TCP/IP](#) 

4.918 wzConnect

Function *wzConnect (SocketPtr : tSocketHandle) : boolean;*
Connects a client to a server.

Group: [WizNet TCP/IP](#) 

4.919 wzCreateSocket

Function *wzCreateSocket : tSocketHandle;*
Requests a socket handle.

Group: [WizNet TCP/IP](#) 

4.920 wzDisconnect

Function *wzDisconnect (SocketPtr : tSocketHandle) : boolean;*
Disconnects a client from a server.

Group: [WizNet TCP/IP](#) 

4.921 wzDNSQueryHost

Function *wzDNSQueryHost (Buffer : pointer; BuffLen : word; Hostname : pointer to string;
var Result_IP : tIPAddr) : boolean;*
Creates the DNS client and requests an IP address.

Group: [WizNet TCP/IP](#) 

4.922 wzFreeSocket

Procedure *wzFreeSocket (SocketPtr : tSocketHandle);*
Frees a socket handle.

Group: [WizNet TCP/IP](#) 

4.923 wzGetLastError

Function *wzGetLastError (SocketPtr : tSocketHandle) : TwzStatus;*
Returns the error state of the last operation.

Group: [WizNet TCP/IP](#) 

4.924 wzGetSocketState

Function *wzGetSocketState (SocketPtr : tSocketHandle) : TwzStatus;*
Returns the current state of a socket.

Group: [WizNet TCP/IP](#) 

4.925 wzInit

Function *wzInit : boolean;*
Initiates a software reset in the W3100A chip and then initializes the chip.

Group: [WizNet TCP/IP](#) 

4.926 wzInitSocket

Function *wzInitSocket (SocketPtr : tSocketHandle) : boolean;*
Only for **UDP**. Basic initialization of a socket for UDP transfers.

Group: [WizNet TCP/IP](#) 

4.927 wzListen

Function *wzListen (SocketPtr : tSocketHandle) : boolean;*
Activates a server.

Group: [WizNet TCP/IP](#) 

4.928 wzPacketReceived

Function *wzPacketReceived (SocketPtr : tSocketHandle) : boolean;*
Returns a true if a packet has been received.

Group: [WizNet TCP/IP](#) 

4.929 wzReceiveBuffer

Function *wzReceiveBuffer (SocketPtr : tSocketHandle): word;*
Receives data.

Group: [WizNet TCP/IP](#) 

4.930 wzReInitSocket

Function *wzReInitSocket (SocketPtr : tSocketHandle) : boolean;*
Only for **UDP**. Similar to wzInitSocket.

Group: [WizNet TCP/IP](#) 

4.931 wzReset

Procedure *wzReset*;

Initiates a software reset in the W3100A chip.

Group: [WizNet TCP/IP](#) 

4.932 wzResumeReceive

Function *wzResumeReceive* (*SocketPtr* : *tSocketHandle*) : *boolean*;

Resumes the receive.

Group: [WizNet TCP/IP](#) 

4.933 wzSendBuffer

Function *wzSendBuffer* (*SocketPtr* : *tSocketHandle*; *Buffer* : *Pointer*; *Len* : *word*): *boolean*;

Send command for the Kernel.

Group: [WizNet TCP/IP](#) 

4.934 wzSetDNSserver

Procedure *wzSetDNSserver* (*IPAddr* : *tIPAddr*);

The IP-address of a remote DNS server is set.

Group: [WizNet TCP/IP](#) 

4.935 wzSetGatewayAddr

Procedure *wzSetGatewayAddr* (*IPAddr* : *tIPAddr*);

Defines the Gateway IP-address of this stack.

Group: [WizNet TCP/IP](#) 

4.936 wzSetHWAddr

Procedure *wzSetHWAddr* (*MacAddr* : *TMacAddr*);

Defines the MAC address of this stack.

Group: [WizNet TCP/IP](#) 

4.937 wzSetIPAddr

Procedure *wzSetIPAddr* (*IPAddr*, *Mask* : *tIPAddr*);

Defines the IP-address and the subnet mask of this stack.

Group: [WizNet TCP/IP](#) 

4.938 wzSetPriority

Procedure *wzSetPriority (prio : TwzPriority);*
Defines the priority of the internal wzTask.

Group: [WizNet TCP/IP](#) 

4.939 wzSetRetryCount

Procedure *wzSetRetryCount (Retry : byte);*
Defines the count of retries if a packet send failed.

Group: [WizNet TCP/IP](#) 

4.940 wzSetSNTPserver

Procedure *wzSetSNTPserver (IPAddr : tIPAddr);*
The IP-address of a remote SNTP server is set.

Group: [WizNet TCP/IP](#) 

4.941 wzSetTimeOut

Procedure *wzSetTimeOut (RetryTimeout : word);*
Defines the timeout between the retries.

Group: [WizNet TCP/IP](#) 

4.942 wzSNTPqueryDateTime

Function *wzSNTPqueryDateTime (Buffer : pointer; BuffLen : word;
var DateTime : tDateTime) : boolean;*
Creates the SNTP client and receives the actual time (GMT) and date.

Group: [WizNet TCP/IP](#) 

4.943 wzTelnetClose

Procedure *wzTelnetClose;*
Shuts the server down. Re-initialized with wzTelnetListen.

Group: [WizNet TCP/IP](#) 

4.944 wzTelnetConnected

Function *wzTelnetConnected : boolean;*
Returns the connect state of the server.

Group: [WizNet TCP/IP](#) 

4.945 wzTelnetCreate

Function *wzTelnetCreate* : *boolean*;
Creates the Telnet server.

Group: [WizNet TCP/IP](#) 

4.946 wzTelnetEcho

Procedure *wzTelnetEcho* (*EchoOn* : *boolean*);
Enables/disabled the echo of the Telnet server.

Group: [WizNet TCP/IP](#) 

4.947 wzTelnetFree

Procedure *wzTelnetFree*;
Frees a Telnet socket.

Group: [WizNet TCP/IP](#) 

4.948 wzTelnetGetClient

Function *wzTelnetGetClient* (*var ClientIP* : *tlPaddr*; *var ClientPort* : *word*) : *boolean*;
Reads the IP-address and Port of a client.

Group: [WizNet TCP/IP](#) 

4.949 wzTelnetGetState

Function *wzTelnetGetState* : *TwzStatus*;
Requests the state without any influence to the state of the server.

Group: [WizNet TCP/IP](#) 

4.950 wzTelnetIdleTimeout

Procedure *wzTelnetIdleTimeout* (*tnTimeout* : *byte*);
Sets the timeout for an inactive client.

Group: [WizNet TCP/IP](#) 

4.951 wzTelnetListen

Function *wzTelnetListen* : *boolean*;
Switches the server after a create into the Listen mode.

Group: [WizNet TCP/IP](#) 

4.952 wzTelnetRead

Function *wzTelnetRead* (*Prompt : tTelnetStr*) : *tTelnetStr*;
The Telnet server gets a command from the client.

Group: [WizNet TCP/IP](#) 

4.953 wzTelnetWrite

Function *wzTelnetWrite* (*Str : tTelnetStr*) : *boolean*;
Transmits instruction for the server.

Group: [WizNet TCP/IP](#) 

4.954 wzTelnetWriteLn

Function *wzTelnetWriteLn* (*Str : tTelnetStr*) : *boolean*;
Same as wzTelnetWrite, appends a CRLF to the string.

Group: [WizNet TCP/IP](#) 

5 Notes

Index

- \$ -

\$ANALYSIS_ON 46
 \$BDATA 46
 \$BootApplication 47
 \$BOOTRST 47
 \$CodeStart 47
 \$D 47
 \$DATA 48
 \$DEBDELAY 48
 \$DEFINE 48
 \$DEPHASE 48
 \$DEVICE 48
 \$EEPROM 49
 \$EEPROM1 49
 \$ELSE 49
 \$ELSIF 49
 \$ELSIFDEF 50
 \$ENDIF 50
 \$ENUMTOASM 50
 \$HEXNAME 51
 \$HEXPATH 50
 \$I 51
 \$IDATA 51
 \$IDATA1 51
 \$IF 52
 \$IFDEF 52
 \$IFNDEF 52
 \$J 52
 \$LCDNOINIT 53
 \$LCDNOWAIT 53
 \$MODBUS 53
 \$NOADDRCHECK 53
 \$NOFRAME 53
 \$NOINIT 54
 \$NOOVRCHECK 54
 \$NORAMCHECK 54
 \$NOREGSAVE 54
 \$NORETURNCHECK 54
 \$NOSAVE 55
 \$NOSHADOW 55

\$NOWATCHDOGAUTO 55
 \$OPTI ALLOW_INLINE 55
 \$OPTI CHECK_RETURN_REGS 56
 \$OPTI NO_ALLOW_INLINE 56
 \$OPTI NO_CHECK_RETURN_REGS 56
 \$OPTI NO_CSE_OPT 57
 \$OPTI SMARTLINK_ONLY 57
 \$OPTI_BETA_OFF 56
 \$OPTI_QUICK 57
 \$OPTIMISE 55
 \$OVERLAY 57
 \$PCU 58
 \$PDATA 58
 \$PHASE 58
 \$Q 58
 \$REUTILIZE 59
 \$SHOWERROR 59
 \$SHOWWARNING 59
 \$SL 59
 \$TYPEDCONST 60
 \$UDATA 60
 \$UNDEF 60
 \$VALIDATE 60
 \$VALIDATE_ALL 61
 \$VALIDATE_OFF 61
 \$VALIDATE_ON 61
 \$VectTab 62
 \$W 62
 \$WG 62
 \$X 62
 \$XDATA 63
 \$XDATA1 63
 \$XDATA2 63
 \$XDATA3 63
 \$XDATA4 63
 \$XIO 63
 \$ZEROLOCVARS 63

- 0 -

0070 18
 0073 18

- 1 -

14seg 18

- 4 -

44780 18

- 6 -

66712 18

- 7 -

7seg 17

- 8 -

8bit Filesystem 11

- A -

Abs 92
 absolute 64
 ActiveEditor 20
 ADC 7
 ADCchans 7
 ADCPort 7
 ADCpresc 7
 AddApp 68
 AddAVFilter 92
 Addr 92
 and 64
 Append 92
 ArcTan 92
 array 64
 ArrToStr 92
 asm 64
 assign 65
 at 65
 AVR_CAN_BaudRate 93
 AVR_CAN_Disable 93
 AVR_CAN_Enable 93
 AVR_Can_GetError 93
 AVR_Can_GetStatus 93
 AVR_CAN_Init 93
 AVR_CAN_RxErrCount 93
 AVR_CAN_SetRxEMask 94
 AVR_CAN_SetRxMask 94

AVR_CAN_StartMessage 94

AVR_CAN_TxErrCount 94

- B -

BankDevIni 8
 BankDevInp 8
 BankDevOut 8
 BankDevPtr 94
 BankingPort 8
 BankPort 8
 BCDtoByte 94
 Beep 9
 BeepChirpH 94
 BeepChirpL 95
 BEEPCLICK 95
 BeepOut 95
 BeepOutErr 95
 BeepOutHL 95
 BeepOutLH 95
 BeepPort 9
 BeepSiren 95
 BeepStepHL 96
 BeepStepLH 96
 begin 65
 Bit 96
 BitCountOf 96
 bitset 65
 boolean 65
 BoolToStr 96
 Boot_Init 96
 BootRestart 97
 break 66
 Broadcast 40, 44
 BufferLen 40
 BufferPtr 40
 by 66
 byte 66
 ByteToBCD 97
 ByteToBin 97
 ByteToHex 97
 ByteToStr 97

- C -

CalByte 68

CalcChecksum 97
 CalcFlashCheck 97
 CalcFlashCheck_A 98
 CalcFlashCheck_B 98
 CalcFlashCheck_S 98
 CAN 7
 CAN_ACKError 7
 CAN_AVR 7
 CAN_AVRbaud 7
 CAN_Baud100 7
 CAN_Baud1000 7
 CAN_Baud125 7
 CAN_Baud200 7
 CAN_Baud25 7
 CAN_Baud250 7
 CAN_Baud50 7
 CAN_Baud500 7
 CAN_BitError 7
 CAN_CRCError 7
 CAN_DLCwarn 7
 CAN_FrameError 7
 CAN_RxOK 7
 CAN_RxPipe 7
 CAN_StuffError 7
 CAN_TxOK 7
 CAN_TxPipe 7
 case 66
 ChangeDir 98
 char 67
 CharSet 22
 CheckFrameValid 99
 CheckStackValid 99
 chr 67
 class 67
 ClearDeviceLock 99
 ClearIncrAll4 99
 ClearIncrementVal 99
 ClearIncrVal4 99
 ClearKeyBoard 99
 ClearKeyBoard8 100
 CLEARRUNERR 100
 close 67
 CLOSED 44
 column 22
 CompareBlock 100
 CompareIP 100

CompareMAC 100
 CompareNet 100
 CompilerSwitch 3
 COMport 68
 const 67
 continue 68
 Copy 100
 CopyBlock 101
 Cos 101
 CosD 101
 CosInt 101
 CosInt16 101
 CPUSleep 101
 CRCcheck 101
 CRCstreamAdd 102
 CRCstreamAddP 102
 CRCstreamInit 102

- D -

DataBit5 29
 DataBit6 29
 DataBit7 29
 DataBit8 29
 DCF77 9
 DCFclock 9
 DCFDayLightSave 102
 DCFfield 102
 DCFfieldMode 9
 DCFport 9
 DCFready 102
 DCFupdate 102
 DDS10 10
 DDS10buildTab 103
 DDS10port 10
 DDS10setFrequ 103
 DDS10setTab 103
 DDS10start 103
 DDS10stop 103
 DDS10Tables 10
 DDS10Timer 10
 Debounce 16, 17, 38
 Dec 103
 DeclAVfilter 103
 DecSema 104
 DecToLim 104

DecToLimWrap 104
 DefCharSet 22
 define 68
 define_fuses 68
 define_usr 69
 definefrom 69
 DegToRad 104
 DelayedAck 40, 44
 Delete 104
 device 69
 devicelock 69
 Disable_JTAGport 104
 DisableInts 104
 Disk_A 11
 Disk_B 11
 DiskFormat 105
 DiskFree 105
 DiskReset 105
 DiskSelect 105
 Disp14Blink 105
 Disp14Buff 18
 Disp14Clear 105
 Disp14ClrEOL 105
 Disp14DigBlink 106
 Disp14Digits 18
 Disp14Mode 18
 Disp14Out 106
 Disp14Pos 106
 Disp14sPort 18
 Disp14Test 106
 Disp7sPort 17
 Disp7Test 106
 DispBlink 106
 DispClear 106
 DispCleol 107
 DispDigBlink 107
 DispDigits 17
 DispMode 17
 DispOut 107
 DispPos 107
 div 69
 Diverse 5
 do 69
 downto 70
 dsSine 10
 dsSquare 10

dsTriLeft 10
 dsTriRight 10
 dsTriSym 10
 DynamicTimeOut 40, 44

- E -

EdBoolean 107
 edBooleanEd 20
 EdBooleanFalse 20
 EdBooleanTrue 20
 EdByte 107
 edByteEd 20
 EdDate 107
 EdDateDelim 20
 edDateEd 20
 edDownLim 20
 EdEditLength 20
 edEEProm 20
 edFlash 20
 EdInteger 108
 edIntegerEd 20
 EdIPAddress 108
 edIPAddressEd 20
 EdIPDelim 20
 EdKeyDown 20
 EdKeyExit 20
 EdKeyLeft 20
 EdKeyNone 20
 EdKeyRight 20
 EdKeyUP 20
 EdLabelLength 20
 EdLCDMulti 20
 edLCDnone 20
 EdLCDStandard 20
 edLeftLim 20
 EdList 108
 edListEd 20
 EdLongInt 108
 edLongIntEd 20
 EdLongWord 109
 edLongWordEd 20
 edNoKeyHandler 20
 edNoLCDDefined 20
 edNoneEd 20
 EdPreClearLine 20

edRam 20
 edRightLim 20
 EdString 109
 edStringEd 20
 EdTime 109
 EdTimeDelim 20
 edTimeEd 20
 edUPLim 20
 EdWord 109
 edWordEd 20
 EEpromPtr 109
 els_if 70
 else 70
 elsif 70
 Enable_JTAGport 110
 ENABLEINTS 110
 EnableIntsX 110
 EnablePWM 110
 EnablePWMC0A 110
 EnablePWMC0B 110
 EnablePWMC0C 110
 EnablePWMC0D 110
 EnablePWMC1A 110
 EnablePWMC1B 110
 EnablePWMD0A 110
 EnablePWMD0B 110
 EnablePWMD0C 110
 EnablePWMD0D 110
 EnablePWMD1A 110
 EnablePWMD1B 110
 EnablePWME0A 110
 EnablePWME0B 110
 EnablePWME0C 110
 EnablePWME0D 110
 EnablePWME1A 110
 EnablePWME1B 110
 EnablePWMF0A 110
 EnablePWMF0B 110
 EnablePWMF0C 110
 EnablePWMF0D 110
 EnablePWMF1A 110
 EnablePWMF1B 110
 end 70
 end_asm 70, 72
 end_case 71, 72
 end_for 71, 72
 end_if 71, 72
 end_loop 71, 73
 end_try 71, 73
 end_while 71, 73
 end_with 72, 73
 endasm 70, 72
 endcase 71, 72
 endfor 71, 72
 endif 71, 72
 endloop 71, 73
 EndOfFile 110
 endtry 71, 73
 endwhile 71, 73
 endwith 72, 73
 enMRFstat 24
 eof 73
 eoln 73
 Even 110
 except 74
 Excl 111
 exec 74
 exit_loop 74
 exitloop 74
 Exp 111
 export 74
 EXTRACTFILEEXT 111
 EXTRACTFILENAME 111
 EXTRACTFILEPATH 111

- F -

F16_BlockRandomWrite 111
 F16_BlockRead 111
 F16_BlockWrite 112
 F16_ChangeDir 112
 F16_CheckDisk 112
 F16_CheckHandle 112
 F16_CreateDir 112
 F16_DateToStr 112
 F16_DirGetDate 112
 F16_DirLevels 10
 F16_DiskInit 113
 F16_DiskReset 113
 F16_EndOfFile 113
 F16_FileAppend 113
 F16_FileAssign 113

- F16_FileClose 113
- F16_FileCopy 113
- F16_FileCreate 114
- F16_FileDelete 114
- F16_FileExist 114
- F16_FileGetAttr 114
- F16_FileGetDate 114
- F16_FileHandles 10
- F16_FilePos 114
- F16_FileRename 114
- F16_FileReset 115
- F16_FileRewrite 115
- F16_FileSeek 115
- F16_FileSetAttr 115
- F16_FileSetDate 115
- F16_FileSize 115
- F16_FileSizeH 115
- F16_FindFirst 116
- F16_FindNext 116
- F16_GetCurDir 116
- F16_GetDiskError 116
- F16_GetDiskFree 116
- F16_GetDiskSize 116
- F16_GetUsedHandles 116
- F16_MMCSpeed 10
- F16_PathExist 117
- F16_PathExpand 117
- F16_RandomWrite 117
- F16_ReadSector 117
- F16_RemoveDir 117
- F16_StrToDate 117
- F16_StrToTime 117
- F16_TimeToStr 118
- F16_WriteSector 118
- false 74
- Fat16 10
- FEdit 20
- file 75
- FileAppend 118
- FileBuffer 11
- FileChangeDir 118
- FileClose 118
- FileCreate 118
- FileDelete 118
- FileExists 119
- FileFirst 119
- FileGetAttr 119
- FileHandleCheck 119
- FileHandles 11
- FileNext 119
- FileOpen 119
- FilePos 119
- FileRead 120
- FileRename 120
- FileReset 120
- FileRewrite 120
- FileSeek 120
- FileSetAttr 120
- FileSize 120
- FileSysReset 121
- Filesystem 10, 11
- FileWrite 121
- FillBlock 121
- FillRandom 121
- finalization 75, 88
- Fix64 4
- Fix64ArcCos 121
- Fix64ArcCosD 121
- Fix64ArcCosh 122
- Fix64ArcCot 122
- Fix64ArcCotD 122
- Fix64ArcCsc 122
- Fix64ArcCscD 122
- Fix64ArcSec 122
- Fix64ArcSecD 123
- Fix64ArcSin 123
- Fix64ArcSinD 123
- Fix64ArcSinh 123
- Fix64ArcTan 123
- Fix64ArcTan2 123
- Fix64ArcTan2D 124
- Fix64ArcTanD 124
- Fix64ArcTanh 124
- Fix64Cos 124
- Fix64CosD 124
- Fix64Cosh 124
- Fix64Cot 125
- Fix64CotD 125
- Fix64Csc 125
- Fix64CscD 125
- Fix64DegToRad 125
- Fix64DegToRadD 125

Fix64DivInt 126
 Fix64DivLong 126
 Fix64Even 126
 Fix64Exp 126
 Fix64Integrate 126
 Fix64IsPowOfTwo 126
 Fix64Ln 127
 Fix64Log 127
 Fix64Log10 127
 Fix64LogN 127
 Fix64Mod 127
 Fix64ModInt 127
 Fix64MulInt 128
 Fix64MulLong 128
 Fix64Odd 128
 Fix64Power 128
 Fix64PowerInt 128
 Fix64Quadrant 128
 Fix64RadToDeg 129
 Fix64Sec 129
 Fix64SecD 129
 Fix64Sin 129
 Fix64SinD 129
 Fix64Sinh 129
 Fix64Sqrt 129
 Fix64Tan 130
 Fix64TanD 130
 Fix64Tanh 130
 Fix64ToFloat 130
 Fix64ToHex 130
 Fix64ToInt 130
 Fix64ToLongInt 131
 Fix64ToStr 131
 Fix64ValueInTolerance 131
 Fix64ValueInToleranceP 131
 FlashClearPage 131
 FlashCopyF2R 131
 FlashCopyR2F 132
 FlashDownloader 132
 FlashErasePage 132
 FlashInitPage 132
 FlashLoaderExit 132
 FlashLoaderInit 132
 FlashLoaderRecv 132
 FlashLoaderTransm 133
 FlashProgPage 133

FlashPtr 133
 FLASHREADFUSES 133
 FlashReadPage 133
 Flashwrite 11
 FlashWriteFuses 133
 FlashWritePage 133
 FlashWriter 11
 float 75
 FloatAsLong 134
 FloatToFix64 134
 FloatToStr 134
 FlushBuffer 134
 for 75
 forward 76
 Frac 135
 FrameSize 8
 FreeMem 135
 FreqBaseNone 12
 FreqCount 12
 FreqCount2 12
 FreqCountRestart 135
 FreqCountRestart2 135
 FreqTimer 12
 FreqTimer2 12
 Frequencycounter 12
 from 76
 function 76
 FuseBits0 68
 FuseBits1 68
 FuseBits2 68

- G -

gClearPixel 135
 gClearView 135
 gClrScr 135
 gDispRefresh 136
 gDrawBitmap 136
 gDrawBitmapN 136
 gDrawCircle 136
 gDrawLine 136
 gDrawLineTo 136
 gDrawLineToRel 136
 gDrawRect 137
 gDrawString 137
 gDrawStringRel 137

GetAdc 137
 GetAVfilter 137
 GetBankNum 137
 GetCurDir 138
 GetCurDisk 138
 GetCurProcess 138
 GetExceptResult 138
 GetFrameFree 138
 GetFreqCounter 138
 GetFreqCounter2 138
 GetFreqCounter2L 139
 GetFreqCounterL 139
 GetFreqCountMode 139
 GetFreqCountMode2 139
 GetFreqCountOvrFlow 139
 GetFreqCountOvrFlow2 139
 GetIncrementRel 139
 GetIncrementVal 140
 GetIncrRel4 140
 GetIncrVal4 140
 GetKey 140
 GetKey8 140
 GetKeyRaised 140
 GetKeyRaised8 140
 GetLargestBlock 141
 GetMem 141
 GetMemAvail 141
 GETPRIORITY 141
 GetProcessID 141
 GetProcessState 141
 GetPulseCount 141
 GetPulseCount2 142
 GetStackFree 142
 GetSysTimer 142
 GetTable 142
 GetTaskFrameFree 142
 GetTaskStackFree 142
 GetTimeCounter 142
 GetTimeCounter2 143
 GetTimeCounterP 143
 GetTimeCounterP2 143
 GetTWIsIslaveSTAT 143
 GetWatchDogFlag 143
 gFillCircle 143
 gFillRect 143
 gFrameView 144

gGetCurView 144
 gGetLineColor 144
 gGetLineMode 144
 gGetTextBkGnd 144
 gGetTextColor 144
 gGetTextJustify 145
 gGetTextMode 145
 gMoveTo 145
 gMoveToRel 145
 gOpenView 145
 goto 76
 gPntToScale 145
 Graphic 22
 gRestoreView 145
 gSaveView 146
 gScaleToPnt 146
 gScaleView 146
 gSetBitMapRAM 146
 gSetCharSet 146
 gSetCharSetRAM 146
 gSetLineColor 146
 gSetLineMode 147
 gSetPixel 147
 gSetTextBkGnd 147
 gSetTextColor 147
 gSetTextJustify 147
 gSetTextMode 147
 gSwitchView 147
 GViewports 22
 gXorPixel 148

- H -

HexToInt 148
 Hi 148
 Higher 148
 HiNibble 148
 HiWord 148

- I -

I2C Hardware Driver 42
 I2C_7Buff1 13
 I2C_7Buff2 13
 I2C_7Buff3 13
 I2C_7Buff4 13

- I2C_7Mode 13
- I2C_7sDig1 13
- I2C_7sDig2 13
- I2C_Disp7 13
- I2C_Disp7Clear 148
- I2C_Disp7CLEOL 149
- I2C_Disp7Ctrl 149
- I2C_Disp7DigitBlink 149
- I2C_Disp7Dim 149
- I2C_Disp7Get 149
- I2C_Disp7Init 149
- I2C_Disp7Out 149
- I2C_Disp7Pos 150
- I2C_Disp7Refresh 150
- I2C_Disp7Set 150
- I2Cclk 13, 14, 15, 19, 23
- I2Cdat 13, 14, 15, 23
- I2Cdat 19
- I2Cdisp7 13
- I2Cexpand 14
- I2Cexpand_5 15
- I2CexpPorts 14
- I2CexpPorts_5 15
- I2CexpStat 150
- I2CexpStat_5 150
- I2Cinp 150
- I2Cout 150
- I2Cport 13, 14, 19, 23
- I2Cstat 151
- idle 76
- if 77
- implementation 77, 88
- import 77
- in 77
- inaPrioMedium 40
- Inc 151
- Incl 151
- IncrCount4start 151
- IncrCount4stop 151
- IncrCounter 12
- IncrementCounter 12
- IncrementCounter4 12
- IncrPort 12
- IncrPort4 12
- IncrScan4 12
- IncSema 151
- IncToLim 151
- IncToLimWrap 152
- inherit 77
- inherited 78
- initialization 78, 88
- Inp_Raise1 152
- Inp_Raise2 152
- Inp_RaiseG 152
- Inp_Stable1 152
- Inp_Stable2 152
- Inp_StableG 152
- Insert 152
- Int 152
- int64 78
- int8 78
- integer 78
- IntegrateB 152
- Integratel 153
- Integratel8 153
- IntegrateW 153
- interface 79, 88
- InterPolX 153
- InterPolY 153
- interrupt 79
- IntToBin 153
- IntToFix64 153
- IntToHex 154
- IntToStr 154
- IOexpand 16
- IOexplnp 16
- IOexplnp0 16
- IOexplnp1 16
- IOexplnp2 16
- IOexplnpArr 16
- IOexpOutp 16
- IOexpOutp0 16
- IOexpOutp1 16
- IOexpOutp2 16
- IOexpOutpArr 16
- IOexpUpdate 154
- IPOct1 20
- IPOct2 20
- IPOct3 20
- IPOct4 20
- IPtoSTR 154
- IsCurProcess 154

IsPowOfTwo 154
isSysTimerZero 154

- K -

KBRepeatDelay 20
KBRepeatTrigger 20
KeyB8Col 17
KeyB8Pipe 17
KeyB8Row 17
KeyB8Type 17
KeyBoard 16
KeyBoard8 16, 17
KeyBoardEnable 155
KeyBoardEnable8 155
KeyboardPipe 17
KeyBoardRepeat 155
KeyBoardRepeat8 155
KeyBoardTimer 17
KeyPort8 17
KeyRaised 155
KeyRaised8 155
Keys 16, 17
KeySet 16, 17
KeyStat 155
KeyStat8 156
KeyStatRaised 156
KeyStatRaised8 156

- L -

label 79
LANadr 29
LANbaud 29
LANcheck 29
LANctrl 29
LANframe 29
LANmode 29
LANport 29
LANrxAutoAck 156
LANrxBuff 29
LANrxClear 156
LANrxStat 156
LANTxBuff 29
LANTxClear 156
LANTxFrame 157

LANTxStat 157
lastKeyBoardKey 17
lastMatrixKey 16
LCD_m1 19
LCD_m2 19
LCD_m3 19
LCD_m4 19
LCD_m5 19
LCD_m6 19
LCD_m7 19
LCD_m8 19
LCDbarGraph 18
LCDBargraph1 18
LCDBargraph2 18
LCDBargraph3 18
LCDBargraph4 18
LCDbarInit_M 157
LCDbarInit_P 157
LCDbarOut1 157
LCDbarOut2 157
LCDbarOut3 157
LCDbarOut4 157
LCDbarSet1 157
LCDbarSet2 157
LCDbarSet3 157
LCDbarSet4 157
LCDcharset 158
LCDCHARSET_M 158
LCDCHARSET_MP 158
LCDcharsetP 158
LCDclr 158
LCDCLR_M 158
LCDclrEol 158
LCDCLREOL_M 159
LCDclrLine 159
LCDCLRLINE_M 159
LCDColumns 18
LCDcolumns_M 19
LCDctrl 159
LCDCTRL_M 159
LCDcursor 159
LCDCURSOR_M 159
LCDdisplay 18
LCDdisplyM 19
LCDEdit 20
LCDgetPort_M 160

LCDgetXY 160
LCDgetXY_M 160
LCDgraphic 22
LCDgraphMode 22
LCDhome 160
LCDHOME_M 160
LCDinp 160
LCDINP_M 160
LCDlower 161
LCDmultiPort 19
LCDoff 161
LCDOFF_M 161
LCDon 161
LCDON_M 161
LcdOut 161
LCDOUT_M 161
LCDport 18
LCDportInp_M 162
LCDRows 18
LCDrows_M 19
LCDsetPort_M 162
LCDsetup 162
LCDSETUP_M 162
LCDstat 162
LCDSTAT_M 162
LCDtype 18
LCDTYPE_M 19
LCDupper 162
LCDxy 163
LCDXY_M 163
LED14seg 18
LED7seg 17
LEDdot 22
LEDdotBlink 163
LEDdotBlinkDigit 163
LEDdotCharset 163
LEDdotCharsetP 163
LEDdotClr 163
LEDdotClrEOL 164
LEDdotClrLine 164
LEDdotCursor 164
LEDdotDim 164
LEDdotDisplay 22
LEDdotGetXY 164
LEDdotHome 164
LEDdotInit 164
LEDdotOff 165
LEDdotOn 165
LEDdotOut 165
LEDdotXY 165
Length 165
linear 22
Lo 165
Lock 165
LockBits0 68
LockBits1 68
locked 79
Log10 166
LogN 166
Long64ToHex 166
Long64ToStr 166
LongAsFloat 166
longint 79
LongToHex 166
LongToStr 166
longword 80
LoNibble 167
loop 80
LowerCase 167
Lower 167
LowerCase 167
LoWord 167
LowPassFW 167
LPT 23
LPTctrl 167
LPTdir 168
LPTinit 168
LPTinp 168
LPTout 168
LPTport 23
LPTreset 168
LPTstat 168

- M -

MACtoSTR 168
Math 4
MatrixCol 16
MatrixKeyPipe 16
MatrixPipe 16
MatrixRow 16
MatrixTimer 16

-
- Max 169
 - mb_GetModBusDevID 169
 - mb_GetModBusExceptionStatus 169
 - mb_GetModBusTimeout 169
 - mb_InpB 24, 25
 - mb_InpF 24, 25
 - mb_InpI 24, 25
 - mb_InpI32 24, 25
 - mb_InpW 24, 25
 - mb_InpW32 24, 25
 - mb_RdWrB 24, 25
 - mb_RdWrF 24, 25
 - mb_RdWrI 24, 25
 - mb_RdWrI32 24, 25
 - mb_RdWrW 24, 25
 - mb_RdWrW32 24, 25
 - mb_setAfterCoilRead 169
 - mb_setAfterCoilWrite 169
 - mb_setAfterRegisterRead 169
 - mb_setAfterRegisterWrite 170
 - mb_setBeforeCoilRead 170
 - mb_setBeforeCoilWrite 170
 - mb_setBeforeRegisterRead 170
 - mb_setBeforeRegisterWrite 170
 - mb_SetModBusDevID 170
 - mb_SetModBusExceptionStatus 170
 - mb_SetModBusTimeout 171
 - mDelay 171
 - Min 171
 - MiniFilesystem 11
 - MIRF24 24
 - MIRF24port 24
 - Mirror16 171
 - Mirror32 171
 - Mirror8 171
 - mod 80
 - ModBus 24, 25
 - ModBusMode 24, 25
 - ModBusServASCII 24
 - ModBusServRTU 25
 - mrfChan1 24
 - mrfChan10 24
 - mrfChan11 24
 - mrfChan12 24
 - mrfChan13 24
 - mrfChan14 24
 - mrfChan2 24
 - mrfChan3 24
 - mrfChan4 24
 - mrfChan5 24
 - mrfChan6 24
 - mrfChan7 24
 - mrfChan8 24
 - mrfChan9 24
 - mrfdBm0 24
 - mrfdBm12 24
 - mrfdBm18 24
 - mrfdBm6 24
 - MRFgetLostPkts 171
 - MRFgetRetryCnt 172
 - MRFgetRxPower 172
 - MRFgetRxType 172
 - MRFgetState 172
 - MRFinit 172
 - mrfMAX_RT 24
 - mrfPKTbcast 24
 - mrfPKTdata 24
 - mrfPKTnone 24
 - mrfRF1000 24
 - mrfRF2000 24
 - mrfRF250 24
 - mrfRX_DR 24
 - mrfRX_pn0 24
 - mrfRX_pn1 24
 - mrfRX_pn2 24
 - MRFRxPacket 172
 - MRFsetChan 173
 - MRFsetFreq 173
 - MRFsetLocalAdr 173
 - MRFsetPower 173
 - MRFsetPWRdown 174
 - MRFsetRetryMax 174
 - MRFsetRetryTimeOut 174
 - MRFsetRFspeed 174
 - mrfTX_DS 24
 - mrfTX_full 24
 - MRFTxPacket 174
 - MSPI 34, 35
 - MSPI Hardware Driver 34, 35
 - MSPIinOut 175
 - MSPIinOut_C0 175
 - MSPIinOut_C1 175

MSPlinOut_D0	175	MSPlinpByte1	176
MSPlinOut_D1	175	MSPlinpByte2	176
MSPlinOut_E0	175	MSPlinpByte3	176
MSPlinOut_E1	175	MSPlinpLong	176
MSPlinOut_F0	175	MSPlinpLong_C0	176
MSPlinOut_F1	175	MSPlinpLong_C1	176
MSPlinOut0	175	MSPlinpLong_D0	176
MSPlinOut1	175	MSPlinpLong_D1	176
MSPlinOut2	175	MSPlinpLong_E0	176
MSPlinOut3	175	MSPlinpLong_E1	176
MSPlinOutByte	175	MSPlinpLong_F0	176
MSPlinOutByte_C0	175	MSPlinpLong_F1	176
MSPlinOutByte_C1	175	MSPlinpLong0	176
MSPlinOutByte_D0	175	MSPlinpLong1	176
MSPlinOutByte_D1	175	MSPlinpLong2	176
MSPlinOutByte_E0	175	MSPlinpLong3	176
MSPlinOutByte_E1	175	MSPlinpWord	176
MSPlinOutByte_F0	175	MSPlinpWord_C0	176
MSPlinOutByte_F1	175	MSPlinpWord_C1	176
MSPlinOutByte0	175	MSPlinpWord_D0	176
MSPlinOutByte1	175	MSPlinpWord_D1	176
MSPlinOutByte2	175	MSPlinpWord_E0	176
MSPlinOutByte3	175	MSPlinpWord_E1	176
MSPlinp	176	MSPlinpWord_F0	176
MSPlinp_C0	176	MSPlinpWord_F1	176
MSPlinp_C1	176	MSPlinpWord0	176
MSPlinp_D0	176	MSPlinpWord1	176
MSPlinp_D1	176	MSPlinpWord2	176
MSPlinp_E0	176	MSPlinpWord3	176
MSPlinp_E1	176	MSPlout	177
MSPlinp_F0	176	MSPlout_C0	177
MSPlinp_F1	176	MSPlout_C1	177
MSPlinp0	176	MSPlout_D0	177
MSPlinp1	176	MSPlout_D1	177
MSPlinp2	176	MSPlout_E0	177
MSPlinp3	176	MSPlout_E1	177
MSPlinpByte	176	MSPlout_F0	177
MSPlinpByte_C0	176	MSPlout_F1	177
MSPlinpByte_C1	176	MSPlout0	177
MSPlinpByte_D0	176	MSPlout1	177
MSPlinpByte_D1	176	MSPlout2	177
MSPlinpByte_E0	176	MSPlout3	177
MSPlinpByte_E1	176	MSPloutByte	177
MSPlinpByte_F0	176	MSPloutByte_C0	177
MSPlinpByte_F1	176	MSPloutByte_C1	177
MSPlinpByte0	176	MSPloutByte_D0	177

MSPloutByte_D1 177
 MSPloutByte_E0 177
 MSPloutByte_E1 177
 MSPloutByte_F0 177
 MSPloutByte_F1 177
 MSPloutByte0 177
 MSPloutByte1 177
 MSPloutByte2 177
 MSPloutByte3 177
 MSPloutLong 177
 MSPloutLong_C0 177
 MSPloutLong_C1 177
 MSPloutLong_D0 177
 MSPloutLong_D1 177
 MSPloutLong_E0 177
 MSPloutLong_E1 177
 MSPloutLong_F0 177
 MSPloutLong_F1 177
 MSPloutLong0 177
 MSPloutLong1 177
 MSPloutLong2 177
 MSPloutLong3 177
 MSPloutWord 178
 MSPloutWord_C0 178
 MSPloutWord_C1 178
 MSPloutWord_D0 178
 MSPloutWord_D1 178
 MSPloutWord_E0 178
 MSPloutWord_E1 178
 MSPloutWord_F0 178
 MSPloutWord_F1 178
 MSPloutWord0 178
 MSPloutWord1 178
 MSPloutWord2 178
 MSPloutWord3 178
 MtrixType 16
 MulDivByte 178
 MulDivInt 178
 MulDivInt8 178
 MulDivLong 178
 MultiLCD 19
 MultiTasking 5

- N -

Negate 179

nil 80
 NoBroadcast 40, 44
 NoDelayedAck 40, 44
 NoDynamicTimeOut 40, 44
 NoInts 179
 NOP 179
 NoSillyWindow 40, 44
 not 80
 Notebook 68

- O -

object 81
 Odd 179
 of 81
 OnADCread 179
 OnFAT16_SS 179
 OnIdleProcess 179
 OnSchedulerEntry 180
 OnSchedulerExit 180
 OnSerRxResumed 180
 OnSerRxResumed1 180
 OnSerRxResumed2 180
 OnSerRxResumed3 180
 OnSerRxResumed4 180
 OnSerRxResumedC0 180
 OnSerRxResumedC1 180
 OnSerRxResumedD0 180
 OnSerRxResumedD1 180
 OnSerRxResumedE0 180
 OnSerRxResumedE1 180
 OnSerRxResumedF0 180
 OnSerRxResumedF1 180
 OnSerRxStopped 180
 OnSerRxStopped1 180
 OnSerRxStopped2 180
 OnSerRxStopped3 180
 OnSerRxStopped4 180
 OnSerRxStoppedC0 180
 OnSerRxStoppedC1 180
 OnSerRxStoppedD0 180
 OnSerRxStoppedD1 180
 OnSerRxStoppedE0 180
 OnSerRxStoppedE1 180
 OnSerRxStoppedF0 180
 OnSerRxStoppedF1 180

OnSysTick 181
OnTickTimer 181
OnTINA_SS 181
or 81
Ord 181
override 81
Override_Fuses 68

- P -

PadLeft 181
PadRight 181
parEven 29
Parity 181
parNone 29
parOdd 29
PeerIP 40
PeerPort 40
pidcontrol 81
pipe 26, 81
PipeFlush 182
PipeFull 182
PipeRecv 182
PipeRecv_ND 182
Pipes 26
PipeSend 182
PipeStat 182
pointer 82
PolarityP_G 38
PolarityP1 38
PolarityP2 38
PopAllRegs 182
PopRegs 183
PORT_STABLE_G 38
PORT_STABLE1 38
PORT_STABLE2 38
Pos 183
PosN 183
Pow 183
Pow10 183
PowerSave 183
Pred 183
PresetAVfilter 184
Priority 184
ProcClock 7
procedure 82

process 5, 82
Processes 24
ProcWaitFlag 184
ProgEEProm 68
ProgFlash 68
ProgFuses 68
ProgLock 68
ProgMode 68
program 82
protIPRAW 44
protMACRaw 44
protTCP 44
protUDP 44
PulseCount 26
PulseCount2 26
PulseCountClear 184
PulseCountClear2 184
PulseCounter 26
PulseCountStart 184
PulseCountStart2 184
PulseCountStop 185
PulseCountStop2 185
PushAllRegs 185
PushRegs 185
PWMport 26
PWMport1A 26
PWMport1B 26
PWMport1C 26
PWMport2A 26
PWMport2B 26
PWMport3A 26
PWMport3B 26
PWMport3C 26
PWMport4A 26
PWMport4B 26
PWMport4C 26
PWMport5A 26
PWMport5B 26
PWMport5C 26

- R -

RadToDeg 185
RaiseException 185
RAM 22
Random 185

- RandomRange 186
- RC5Driver 28
- RC5mode 28
- RC5Rxport 28
- RC5Txport 28
- Read 186
- ReadKey 186
- ReadKey8 186
- ReadKeyBoard 186
- ReadKeyBoard8 186
- ReadLn 186
- readonly 22
- RealTimeClock 28
- RecLen 40
- record 83
- RecvRC5 187
- repeat 83
- ResetProcess 187
- ResetSysTimer 187
- Reseverd Words 6
- RestoreInts 187
- Resume 187
- ResumeAll 187
- return 83
- rol 83
- ror 84
- RotatePntl 187
- Round 188
- RTC 28
- RTCalarm 188
- RTCalarm_Date 188
- RTCalarm_Start 188
- RTCalarm_Stop 188
- RTCalarm_Time 188
- RTCgetDay 188
- RTCgetHour 189
- RTCgetMinute 189
- RTCgetMonth 189
- RTCgetSecond 189
- RTCgetWeekDay 189
- RTCgetYear 189
- RTclock 28
- RTCsetDay 189
- RTCsetHour 190
- RTCsetMinute 190
- RTCsetMonth 190
- RTCsetSecond 190
- RTCsetWeekDay 190
- RTCsetYear 190
- RTCsource 28
- RTCTickHour 190
- RTCTickMinute 191
- RTCTickSecond 191
- RTCtimer 28, 191
- RTCtimer_Load 191
- RTCtimer_Start 191
- RTCtimer_Stop 191
- RunErr 191
- RunTimeErr 192
- RxBuffer 29, 134
- RxBuffer1 134
- RxBuffer2 134
- RxBuffer3 134
- RxBuffer4 134
- RxBufferC0 134
- RxBufferC1 134
- RxBufferD0 134
- RxBufferD1 134
- RxBufferE0 134
- RxBufferE1 134
- RxBufferF0 134
- RxBufferF1 134

- S -

- Schedule 192
- Scheduler 24
- SchedulerOff 192
- SchedulerOn 192
- sDelay 192
- SecTrk_A 11
- SecTrk_B 11
- Selfprog 11
- semaphore 84
- SemaStat 192
- SendRC5 192
- sendsema 84
- Ser_Enable 193
- Ser_Enable1 193
- Ser_Enable2 193
- Ser_Enable3 193
- Ser_Enable4 193

Ser_EnableC0	193	SerInp1	194
Ser_EnableC1	193	SerInp2	194
Ser_EnableD0	193	SerInp3	194
Ser_EnableD1	193	SerInp4	194
Ser_EnableE0	193	SerInpBlock	195
Ser_EnableE1	193	SerInpBlock_P	195
Ser_EnableF0	193	SerInpBlock_TO	195
Ser_EnableF1	193	SerInpBlock_TO1	195
SerBaud	193	SerInpBlock_TO2	195
SerBaud1	193	SerInpBlock_TO3	195
SerBaud2	193	SerInpBlock_TO4	195
SerBaud3	193	SerInpBlock_TOC0	195
SerBaud4	193	SerInpBlock_TOC1	195
SerCtrl	29	SerInpBlock_TOD0	195
SerCtrl1	29	SerInpBlock_TOD1	195
SerCtrl2	29	SerInpBlock_TOE0	195
SerCtrl3	29	SerInpBlock_TOE1	195
SerCtrl4	29	SerInpBlock_TOF0	195
SerDataBits	194	SerInpBlock_TOF1	195
SerDataBits1	194	SerInpBlock1	195
SerDataBits2	194	SerInpBlock1_P	195
SerDataBits3	194	SerInpBlock2	195
SerDataBits4	194	SerInpBlock2_P	195
SerDataBitsC0	194	SerInpBlock3	195
SerDataBitsC1	194	SerInpBlock3_P	195
SerDataBitsD0	194	SerInpBlock4	195
SerDataBitsD1	194	SerInpBlock4_P	195
SerDataBitsE0	194	SerInpBlockC0	195
SerDataBitsE1	194	SerInpBlockC0_P	195
SerDataBitsF0	194	SerInpBlockC1	195
SerDataBitsF1	194	SerInpBlockC1_P	195
SerialLan	29	SerInpBlockD0	195
SerInp	194	SerInpBlockD0_P	195
SerInp_TO	194	SerInpBlockD1	195
SerInp_TO1	194	SerInpBlockD1_P	195
SerInp_TO2	194	SerInpBlockE0	195
SerInp_TO3	194	SerInpBlockE0_P	195
SerInp_TO4	194	SerInpBlockE1	195
SerInp_TOC0	194	SerInpBlockE1_P	195
SerInp_TOC1	194	SerInpBlockF0	195
SerInp_TOD0	194	SerInpBlockF0_P	195
SerInp_TOD1	194	SerInpBlockF1	195
SerInp_TOE0	194	SerInpBlockF1_P	195
SerInp_TOE1	194	SerInpBlockP_TO	196
SerInp_TOF0	194	SerInpBlockP_TO1	196
SerInp_TOF1	194	SerInpBlockP_TO2	196

SerInpBlockP_TO3	196	SerOutBlockC0	197
SerInpBlockP_TO4	196	SerOutBlockC0_P	197
SerInpBlockP_TOC0	196	SerOutBlockC1	197
SerInpBlockP_TOC1	196	SerOutBlockC1_P	197
SerInpBlockP_TOD0	196	SerOutBlockD0	197
SerInpBlockP_TOD1	196	SerOutBlockD0_P	197
SerInpBlockP_TOE0	196	SerOutBlockD1	197
SerInpBlockP_TOE1	196	SerOutBlockD1_P	197
SerInpBlockP_TOF0	196	SerOutBlockE0	197
SerInpBlockP_TOF1	196	SerOutBlockE0_P	197
SerInpC0	194	SerOutBlockE1	197
SerInpC1	194	SerOutBlockE1_P	197
SerInpD0	194	SerOutBlockF0	197
SerInpD1	194	SerOutBlockF0_P	197
SerInpE0	194	SerOutBlockF1	197
SerInpE1	194	SerOutBlockF1_P	197
SerInpF0	194	SerOutC0	196
SerInpF1	194	SerOutC1	196
SerInpSLIP	196	SerOutD0	196
SerInpSLIP1	196	SerOutD1	196
SerInpSLIP2	196	SerOutE0	196
SerInpSLIP3	196	SerOutE1	196
SerInpSLIP4	196	SerOutF0	196
SerInpSLIPC0	196	SerOutF1	196
SerInpSLIPC1	196	SerOutSLIP	197
SerInpSLIPD0	196	SerOutSLIP1	197
SerInpSLIPD1	196	SerOutSLIP2	197
SerInpSLIPE0	196	SerOutSLIP3	197
SerInpSLIPE1	196	SerOutSLIP4	197
SerInpSLIPF0	196	SerOutSLIPC0	197
SerInpSLIPF1	196	SerOutSLIPC1	197
SerOut	196	SerOutSLIPD0	197
SerOut1	196	SerOutSLIPD1	197
SerOut2	196	SerOutSLIPE0	197
SerOut3	196	SerOutSLIPE1	197
SerOut4	196	SerOutSLIPF0	197
SerOutBlock	197	SerOutSLIPF1	197
SerOutBlock_P	197	SerParity	198
SerOutBlock1	197	SerParity1	198
SerOutBlock1_P	197	SerParity2	198
SerOutBlock2	197	SerParity3	198
SerOutBlock2_P	197	SerParity4	198
SerOutBlock3	197	SerParityC0	198
SerOutBlock3_P	197	SerParityC1	198
SerOutBlock4	197	SerParityD0	198
SerOutBlock4_P	197	SerParityD1	198

SerParityE0	198	SerStat4	198
SerParityE1	198	SerStatC0	198
SerParityF0	198	SerStatC1	198
SerParityF1	198	SerStatD0	198
SerPort	24, 25, 29	SerStatD1	198
SerPort_Send	198	SerStatE0	198
SerPort_Send1	198	SerStatE1	198
SerPort_Send2	198	SerStatF0	198
SerPort_Send3	198	SerStatF1	198
SerPort_Send4	198	SerStopBits	199
SerPort_SendC0	198	SerStopBits1	199
SerPort_SendC1	198	SerStopBits2	199
SerPort_SendD0	198	SerStopBits3	199
SerPort_SendD1	198	SerStopBits4	199
SerPort_SendE0	198	SerStopBitsC0	199
SerPort_SendE1	198	SerStopBitsC1	199
SerPort_SendF0	198	SerStopBitsD0	199
SerPort_SendF1	198	SerStopBitsD1	199
SerPort1	29	SerStopBitsE0	199
SerPort2	24, 25, 29	SerStopBitsE1	199
SerPort3	24, 25, 29	SerStopBitsF0	199
SerPort4	24, 25, 29	SerStopBitsF1	199
SerPortC0	29	ServoChans	31
SerPortC1	29	ServoDriver	31
SerPortD0	29	ServoNeutral	31
SerPortD1	29	ServoPort	31
SerPortDSR	29	ServoSwing	31
SerPortDSR1	29	set	84
SerPortDSR2	29	SetAdcFixed	199
SerPortDSR3	29	SetAVfilter	199
SerPortDSR4	29	SetBit	200
SerPortDTR	29	SetDacA	200
SerPortDTR1	29	SetDacB	200
SerPortDTR2	29	SetDeviceLock	200
SerPortDTR3	29	SetFreqCountMode	200
SerPortDTR4	29	SetFreqCountMode2	200
SerPortE0	29	SetIncrementVal	200
SerPortE1	29	SetIncrVal4	201
SerPortF0	29	SetLength	201
SerPortF1	29	SetMSPI0mode	201
SerPorts	29	SetMSPI1mode	201
SerPortSelect	29	SetMSPI2mode	201
SerStat	198	SetMSPI3mode	201
SerStat1	198	SetMSPIclkPha	201
SerStat2	198	SetMSPIclkPha_C0	201
SerStat3	198	SetMSPIclkPha_C1	201

SetMSPIClkPha_D0	201	SetMSPIpresc_C0	202
SetMSPIClkPha_D1	201	SetMSPIpresc_C1	202
SetMSPIClkPha_E0	201	SetMSPIpresc_D0	202
SetMSPIClkPha_E1	201	SetMSPIpresc_D1	202
SetMSPIClkPha_F0	201	SetMSPIpresc_E0	202
SetMSPIClkPha_F1	201	SetMSPIpresc_E1	202
SetMSPIClkPha0	201	SetMSPIpresc_F0	202
SetMSPIClkPha1	201	SetMSPIpresc_F1	202
SetMSPIClkPha2	201	SetMSPIpresc0	202
SetMSPIClkPha3	201	SetMSPIpresc1	202
SetMSPIClkPol	201	SetMSPIpresc2	202
SetMSPIClkPol_C0	201	SetMSPIpresc3	202
SetMSPIClkPol_C1	201	SetPWM	202
SetMSPIClkPol_D0	201	SetPWMC0A	202
SetMSPIClkPol_D1	201	SetPWMC0B	202
SetMSPIClkPol_E0	201	SetPWMC0C	202
SetMSPIClkPol_E1	201	SetPWMC0D	202
SetMSPIClkPol_F0	201	SetPWMC1A	202
SetMSPIClkPol_F1	201	SetPWMC1B	202
SetMSPIClkPol0	201	SetPWMD0A	202
SetMSPIClkPol1	201	SetPWMD0B	202
SetMSPIClkPol2	201	SetPWMD0C	202
SetMSPIClkPol3	201	SetPWMD0D	202
SetMSPImode	201	SetPWMD1A	202
SetMSPImode_C0	201	SetPWMD1B	202
SetMSPImode_C1	201	SetPWME0A	202
SetMSPImode_D0	201	SetPWME0B	202
SetMSPImode_D1	201	SetPWME0C	202
SetMSPImode_E0	201	SetPWME0D	202
SetMSPImode_E1	201	SetPWME1A	202
SetMSPImode_F0	201	SetPWME1B	202
SetMSPImode_F1	201	SetPWMF0A	202
SetMSPIorder	202	SetPWMF0B	202
SetMSPIorder_C0	202	SetPWMF0C	202
SetMSPIorder_C1	202	SetPWMF0D	202
SetMSPIorder_D0	202	SetPWMF1A	202
SetMSPIorder_D1	202	SetPWMF1B	202
SetMSPIorder_E0	202	SetSema	203
SetMSPIorder_E1	202	SetSerBaud (UsartC0)	193
SetMSPIorder_F0	202	SetSerBaud (UsartC1)	193
SetMSPIorder_F1	202	SetSerBaud (UsartD0)	193
SetMSPIorder0	202	SetSerBaud (UsartD1)	193
SetMSPIorder1	202	SetSerBaud (UsartE0)	193
SetMSPIorder2	202	SetSerBaud (UsartE1)	193
SetMSPIorder3	202	SetSerBaud (UsartF0)	193
SetMSPIpresc	202	SetSerBaud (UsartF1)	193

SetServoChan	203
SetServoOffs	203
SetSPIClkPha	203
SetSPIClkPhaC	203
SetSPIClkPhaD	203
SetSPIClkPhaE	203
SetSPIClkPhaF	203
SetSPIClkPol	203
SetSPIClkPolC	203
SetSPIClkPolD	203
SetSPIClkPolE	203
SetSPIClkPolF	203
SetSPIdoubleSpeed	204
SetSPIdoubleSpeedC	204
SetSPIdoubleSpeedD	204
SetSPIdoubleSpeedE	204
SetSPIdoubleSpeedF	204
SetSPImode	204
SetSPImodeC	204
SetSPImodeD	204
SetSPImodeE	204
SetSPImodeF	204
SetSPIorder	204
SetSPIorderC	204
SetSPIorderD	204
SetSPIorderE	204
SetSPIorderF	204
SetSPIpresc	204
SetSPIprescC	204
SetSPIprescD	204
SetSPIprescE	204
SetSPIprescF	204
SetSysBlinkTimer	205
SetSysTimer	205
SetSysTimerM	205
SetTable	205
SetTWImode	205
SetTWInodeAddr	205
SetVectTabBoot	205
Sgn	206
shl	84
shla	84
shortint	85
shr	85
shra	85
SHT11	32
SHT11clk	32
SHT11ConvState	206
SHT11crc	32
SHT11dat	32
SHT11drv	32
SHT11getHum	206
SHT11getStatus	206
SHT11getTemp	206
SHT11sema	32
SHT11setStatus	206
SHT11softReset	206
SHT11startHum	207
SHT11startTemp	207
SHT11synchronize	207
Sign	207
SillyWindow	40, 44
Sin	207
SinD	207
SinInt	207
SinInt16	208
SizeOf	208
Sleep	208
SLIPgetRxCount	208
SLIPgetRxCount1	208
SLIPgetRxCount2	208
SLIPgetRxCount3	208
SLIPgetRxCount4	208
SLIPgetRxCountC0	208
SLIPgetRxCountC1	208
SLIPgetRxCountD0	208
SLIPgetRxCountD1	208
SLIPgetRxCountE0	208
SLIPgetRxCountE1	208
SLIPgetRxCountF0	208
SLIPgetRxCountF1	208
SLIPgetRxState	208
SLIPgetRxState1	208
SLIPgetRxState2	208
SLIPgetRxState3	208
SLIPgetRxState4	208
SLIPgetRxStateC0	208
SLIPgetRxStateC1	208
SLIPgetRxStateD0	208
SLIPgetRxStateD1	208
SLIPgetRxStateE0	208
SLIPgetRxStateE1	208

SLIPgetRxStateF0	208	SlipRxSemaC0	32
SLIPgetRxStateF1	208	SlipRxSemaC1	32
SLIPgetTxState	209	SlipRxSemaD0	32
SLIPgetTxState1	209	SlipRxSemaD1	32
SLIPgetTxState2	209	SlipRxSemaE0	32
SLIPgetTxState3	209	SlipRxSemaE1	32
SLIPgetTxState4	209	SlipRxSemaF0	32
SLIPgetTxStateC0	209	SlipRxSemaF1	32
SLIPgetTxStateC1	209	SLIPsetMode	210
SLIPgetTxStateD0	209	SLIPsetMode1	210
SLIPgetTxStateD1	209	SLIPsetMode2	210
SLIPgetTxStateE0	209	SLIPsetMode3	210
SLIPgetTxStateE1	209	SLIPsetMode4	210
SLIPgetTxStateF0	209	SLIPsetModeC0	210
SLIPgetTxStateF1	209	SLIPsetModeC1	210
SLIPport	32	SLIPsetModeD0	210
SLIPPresumeRx	209	SLIPsetModeD1	210
SLIPPresumeRx1	209	SLIPsetModeE0	210
SLIPPresumeRx2	209	SLIPsetModeE1	210
SLIPPresumeRx3	209	SLIPsetModeF0	210
SLIPPresumeRx4	209	SLIPsetModeF1	210
SLIPPresumeRxC0	209	SLIPsetRxAddr	210
SLIPPresumeRxC1	209	SLIPsetRxAddr1	210
SLIPPresumeRxD0	209	SLIPsetRxAddr2	210
SLIPPresumeRxD1	209	SLIPsetRxAddr3	210
SLIPPresumeRxE0	209	SLIPsetRxAddr4	210
SLIPPresumeRxE1	209	SLIPsetRxAddrC0	210
SLIPPresumeRxF0	209	SLIPsetRxAddrC1	210
SLIPPresumeRxF1	209	SLIPsetRxAddrD0	210
SLIPPrxReady	209	SLIPsetRxAddrD1	210
SLIPPrxReady1	209	SLIPsetRxAddrE0	210
SLIPPrxReady2	209	SLIPsetRxAddrE1	210
SLIPPrxReady3	209	SLIPsetRxAddrF0	210
SLIPPrxReady4	209	SLIPsetRxAddrF1	210
SLIPPrxReadyC0	209	SLIPsetRxBuffer	210
SLIPPrxReadyC1	209	SLIPsetRxBuffer1	210
SLIPPrxReadyD0	209	SLIPsetRxBuffer2	210
SLIPPrxReadyD1	209	SLIPsetRxBuffer3	210
SLIPPrxReadyE0	209	SLIPsetRxBuffer4	210
SLIPPrxReadyE1	209	SLIPsetRxBufferC0	210
SLIPPrxReadyF0	209	SLIPsetRxBufferC1	210
SLIPPrxReadyF1	209	SLIPsetRxBufferD0	210
SlipRxSema1	32	SLIPsetRxBufferD1	210
SlipRxSema2	32	SLIPsetRxBufferE0	210
SlipRxSema3	32	SLIPsetRxBufferE1	210
SlipRxSema4	32	SLIPsetRxBufferF0	210

SLIPsetRxBufferF1	210	SLIPstartTxC_C0	212
SLIPsetTimeOut	211	SLIPstartTxC_C1	212
SLIPsetTimeOut1	211	SLIPstartTxC_D0	212
SLIPsetTimeOut2	211	SLIPstartTxC_D1	212
SLIPsetTimeOut3	211	SLIPstartTxC_E0	212
SLIPsetTimeOut4	211	SLIPstartTxC_E1	212
SLIPsetTimeOutC0	211	SLIPstartTxC_F0	212
SLIPsetTimeOutC1	211	SLIPstartTxC_F1	212
SLIPsetTimeOutD0	211	SLIPstartTxC0	212
SLIPsetTimeOutD1	211	SLIPstartTxC1	212
SLIPsetTimeOutE0	211	SLIPstartTxC2	212
SLIPsetTimeOutE1	211	SLIPstartTxC3	212
SLIPsetTimeOutF0	211	SLIPstartTxC4	212
SLIPsetTimeOutF1	211	SLIPstartTxD0	212
SLIPsetTxAddr	211	SLIPstartTxD1	212
SLIPsetTxAddr1	211	SLIPstartTxEO	212
SLIPsetTxAddr2	211	SLIPstartTxE1	212
SLIPsetTxAddr3	211	SLIPstartTxF0	212
SLIPsetTxAddr4	211	SLIPstartTxF1	212
SLIPsetTxAddrC0	211	SLIPstopRx	212
SLIPsetTxAddrC1	211	SLIPstopRx1	212
SLIPsetTxAddrD0	211	SLIPstopRx2	212
SLIPsetTxAddrD1	211	SLIPstopRx3	212
SLIPsetTxAddrE0	211	SLIPstopRx4	212
SLIPsetTxAddrE1	211	SLIPstopRxCO	212
SLIPsetTxAddrF0	211	SLIPstopRxC1	212
SLIPsetTxAddrF1	211	SLIPstopRxD0	212
SLIPsetTxBuffer	211	SLIPstopRxD1	212
SLIPsetTxBuffer1	211	SLIPstopRxE0	212
SLIPsetTxBuffer2	211	SLIPstopRxE1	212
SLIPsetTxBuffer3	211	SLIPstopRxF0	212
SLIPsetTxBuffer4	211	SLIPstopRxF1	212
SLIPsetTxBufferC0	211	SLIPwasBC	213
SLIPsetTxBufferC1	211	SLIPwasBC_C0	213
SLIPsetTxBufferD0	211	SLIPwasBC_C1	213
SLIPsetTxBufferD1	211	SLIPwasBC_D0	213
SLIPsetTxBufferE0	211	SLIPwasBC_D1	213
SLIPsetTxBufferE1	211	SLIPwasBC_E0	213
SLIPsetTxBufferF0	211	SLIPwasBC_E1	213
SLIPsetTxBufferF1	211	SLIPwasBC_F0	213
SLIPstartTx	212	SLIPwasBC_F1	213
SLIPstartTx1	212	SLIPwasBC1	213
SLIPstartTx2	212	SLIPwasBC2	213
SLIPstartTx3	212	SLIPwasBC3	213
SLIPstartTx4	212	SLIPwasBC4	213
SLIPstartTxC	212	SoftPWM	34

SoftPWMchans	34
SoftPWMport	34
SoftPWMres	34
SoftPWMstart	213
SoftPWMstop	213
SoftPWMtimer	34
Software I2C	14
SoftwarePWM	34
SpeechIOS	33
SpeechOutFlash	213
SpeechOutRAM	213
SpeechPort	33
SpeechReady	214
SpeechStop	214
SpeechTimer	33
SPI	18
SPI Hardware Driver	36
SPI Software Driver	37
SPI_Soft	18
SPI_SS	33, 36
SPIclk	68
SPIcpha	33, 36, 37
SPIcpha1	37
SPIcpha2	37
SPIcpol	33, 36, 37
SPIcpol1	37
SPIcpol2	37
SPIdriver	33, 36
SPIdriver1	37
SPIdriver2	37
SPlinOut	214
SPlinOut1	214
SPlinOut2	214
SPlinOutByte	214
SPlinOutByte1	215
SPlinOutByte2	215
SPlinOutByteC	214
SPlinOutByteD	214
SPlinOutByteE	214
SPlinOutByteF	214
SPlinOutC	214
SPlinOutD	214
SPlinOutE	214
SPlinOutF	214
SPlinp	215
SPlinp1	215
SPlinp2	216
SPlinpByte	216
SPlinpByte1	216
SPlinpByte2	216
SPlinpByteC	216
SPlinpByteD	216
SPlinpByteE	216
SPlinpByteF	216
SPlinpC	215
SPlinpD	215
SPlinpE	215
SPlinpF	215
SPlinpLong	216
SPlinpLong1	217
SPlinpLong2	217
SPlinpLong64	217
SPlinpLong64C	217
SPlinpLong64D	217
SPlinpLong64E	217
SPlinpLong64F	217
SPlinpLongC	216
SPlinpLongD	216
SPlinpLongE	216
SPlinpLongF	216
SPlinpWord	217
SPlinpWord1	217
SPlinpWord2	218
SPlinpWordC	217
SPlinpWordD	217
SPlinpWordE	217
SPlinpWordF	217
SPlinet	37
SPlorder	33, 36, 37
SPlorder1	37
SPlorder2	37
SPlout	218
SPlout1	218
SPlout2	218
SPloutByte	219
SPloutByte1	219
SPloutByte2	219
SPloutByteC	219
SPloutByteD	219
SPloutByteE	219
SPloutByteF	219
SPloutC	218

SPloutD 218
SPloutE 218
SPloutF 218
SPloutLong 219
SPloutLong1 219
SPloutLong2 220
SPloutLong64 220
SPloutLong64C 220
SPloutLong64D 220
SPloutLong64E 220
SPloutLong64F 220
SPloutLongC 219
SPloutLongD 219
SPloutLongE 219
SPloutLongF 219
SPloutWord 220
SPloutWord1 220
SPloutWord2 220
SPloutWordC 220
SPloutWordD 220
SPloutWordE 220
SPloutWordF 220
SPlport 37
SPlpresc 33, 36, 37
SPlretry 37
SpiRxBuff 37
SPlrxClear 221
SPlrxFrame 221
SpiRxBuff 37
SPlrxStat 221
SpiTxBuff 37
SPltxClear 221
SPltxFrame 221
SpiTxLen 37
SPltxStat 221
SQR 221
SQRT 222
SquareDivByte 222
SquareDivInt 222
SquareDivInt8 222
StackSize 8
Start_Processes 222
StepAccValue 38
StepCount 38
StepDestCCW 223
StepDestCW 223
StepDown 38
StepEndFreq 38
StepFull 2 38
StepFull 4 38
StepHalf 6 38
StepMaxFreq 38
StepMicro 2 38
StepMicro 8 38
StepMinFreq 38
StepMini 4 38
StepMini 6 38
StepMode 38
StepOneCCW 223
StepOneCW 223
Stepper Driver 38
StepperOff 223
StepperOn 223
StepperSema 38
SteppHalf 4 38
StepPort 38
StepRampCCW 224
StepRampCW 224
StepRampStop 224
StepRun 38
StepStartFreq 38
StepStop 38
StepType 38
StepUp 38
StepVelocity 224
StopBit1 29
StopBit2 29
str 85
StrClean 224
string 6, 85
StrReplace 224
StrToArr 224
StrToFix64 225
StrToFloat 225
StrToInt 225
STRtolP 225
StrToMac 225
structconst 86
Succ 225
Supply 68
Suspend 225
SuspendAll 226

Swap 226
 SwapIPAddr 226
 SwapLong 226
 SwapMACAddr 226
 SwitchKeyRepeat1 226
 SwitchKeyRepeat2 226
 SwitchKeyRepeatG 226
 SwitchPort 38
 SwitchPort_G 38
 SwitchPort1 38
 SwitchPort1_Clear 227
 SwitchPort2 38
 SwitchPort2_Clear 227
 SwitchPortG_Clear 227
 SysLed 39
 SysLEDallOff 227
 SysLEDallOn 227
 SysLEDBlink 39
 SysLEDBlink0 39
 SysLEDBlink1 39
 SysLEDBlink2 39
 SysLEDBlink3 39
 SysLEDBlink4 39
 SysLEDBlink5 39
 SysLEDBlink6 39
 SysLEDBlink7 39
 SysLEDenable 227
 SysLEDflashAllOff 227
 SysLEDflashAllOn 227
 SysLEDflashMsg 227
 SysLEDflashOff 228
 SysLEDflashOn 228
 SysLEDflashOnce 228
 SysLEDflashOnOff 228
 SysLEDOff 228
 SysLEDOn 228
 SysLEDOnOff 228
 System 7
 System LED 39
 System_Init 229
 SYSTEM_MCUCR_INIT 229
 System_Reset 229
 System_ShutDown 229
 SysTick 7
 SysTickDisable 229
 SysTickEnable 229

SysTickRestart 229
 SysTickStop 230
 systimer 86
 systimer32 86
 systimer8 86

- T -

table 86
 Tan 230
 TanD 230
 task 5, 87
 Tasks 44
 tAVR_CAN_Stat 7
 tAVR_CAN_States 7
 tCAN_baud 7
 tCANMessage11 7
 TCP/IP 40, 44
 tDataBits 29
 tdsMode 10
 TEdActEditor 20
 tEdArrayLocation 20
 tEdErrorEvent 20
 tEdErrorEventHandler 20
 tEdIPAddress 20
 tEdKeyboardHandler 20
 tEdKeys 20
 tEdLCDType 20
 TestDeviceLock 230
 TFreqBase100Hz 12
 TFreqBase100kHz 12
 TFreqBase10kHz 12
 TFreqBase1kHz 12
 TFreqBase1MHz 12
 tFreqCountMode 12
 TGraphStr 22
 TGraphString 22
 then 87
 TI2C_Ctrl7 13
 TI2C_DISP7 13
 TickTimer 41
 TickTimer2 41
 TickTimer2OutpEnable 230
 TickTimer2RawVal 230
 TickTimer2Reload 230
 TickTimer2Start 231

TickTimer2Stop 231
 TickTimer2Time 231
 TickTimerOutpEnable 231
 TickTimerPin 41
 TickTimerRawVal 231
 TickTimerReload 231
 TickTimerStart 231
 TickTimerStop 232
 TickTimerTime 232
 Timecounter 12
 TINA 40
 TINA_Init 232
 TINA_Ping 232
 TINA_Start 232
 TINA_Stop 232
 TinaCore 40
 TinaCreateSocket 232
 TINADriver 40
 TinaFreeSocket 233
 TinaInitSocket 233
 TINALinkStat 233
 TinaPacketReceived 233
 TINApport 40
 TinaPrioAuto 40
 TinaPrioHigh 40
 TinaPrioLow 40
 TinaPrioResume 40
 TinaPrioSuspend 40
 TinaPrioVeryHigh 40
 TinaPrioVeryLow 40
 TinaResumeReceive 233
 TINArxStat 233
 TinasBufferParam 40
 TinaSendPacket 233
 TINASetPriority 234
 TinasInitFailed 40
 TinasInvalidHandle 40
 TinasListenFailed 40
 TinasNoErrors 40
 TinasNotInitialized 40
 TINAsockets 40
 TinasSendFailed 40
 TinasSockClosed 40
 TinasSockCloseWait 40
 TinasSockClosing 40
 TinasSockConnected 40
 TinasSockListen 40
 TinasSockRaw 40
 TinasSockUDP 40
 TINAsack 40
 TinasTimeOutErr 40
 TINAtimer 40
 TIPAddr 40, 44
 TLCD_num 19
 tLPTlines 23
 tLPTlineSet 23
 TMACAddr 40, 44
 tMRFchan 24
 tMRFpkt 24
 tMRFpwr 24
 tMRFrfSpeed 24
 tMRFstat 24
 to 87
 Toggle 234
 tParity 29
 TPulseBase100ms 12
 TPulseBase100s 12
 TPulseBase10s 12
 TPulseBase1s 12
 transparent 42
 TRAP 234
 Trim 234
 TrimLeft 234
 TrimRight 234
 TRKOFFS_A 11
 true 87
 Trunc 235
 try 87
 tSocketHandle 40, 44
 TStepMode 38
 tStopBits 29
 TTextBkGnd 22
 TTimeBase100ms 12
 TTimeBase100s 12
 TTimeBase10s 12
 TTimeBase1s 12
 TTinaBroadcast 40
 TTinaCore 40
 TTinaNDTimeOut 40
 TtinaPacketReceive 40
 TTinaPriority 40
 TTinaSocket 40

TTinaSocketNDAck	40	TWInet	14, 15, 19, 43
TTinaSocketSWS	40	TWInetMode	14, 15, 19, 43
TTINASStatus	40	TWInode	14, 15, 19, 43
TTinaxUDPAKNPort	40	TWlout	236
tTWInetState	43	TWloutC	236
tTWIStates	43	TWloutD	236
TtxtAlHor	22	TWloutE	236
TTxtAlVert	22	TWloutF	236
TTxtRotate	22	TWloutP	236
TWI	42, 43	TWloutPC	236
TWI Networkk	43	TWloutPD	236
TWI_BR100	42, 43	TWloutPE	236
TWI_BR400	42, 43	TWloutPF	236
TWI_BR500	42, 43	TWloutWP	236
TWI_BR600	42, 43	TWloutWPC	236
TWI_BR800	42, 43	TWloutWPD	236
TWI_C	42	TWloutWPE	236
TWI_D	42	TWloutWPF	236
TWI_DevLock	14, 15, 19, 42, 43, 44	TWlpresc	14, 15, 19, 23, 42, 43, 44
TWI_DevLockC	14, 15, 19, 42	TWlprescC	23, 42
TWI_DevLockD	14, 15, 19, 42	TWlprescC	19
TWI_DevLockE	14, 15, 19, 42	TWlprescD	19, 23, 42
TWI_DevLockF	14, 15, 19, 42	TWlprescE	23, 42
TWI_E	42	TWlprescE	19
TWI_F	42	TWlprescF	23, 42
TWladdr	42	TWlprescF	19
TWlbuffer	42	TWlrxAdr	43
TWlframe	14, 15, 19, 43	TWlrxBuff	43
TWlframeBC	14, 15, 19, 43	TWlrxClear	237
TWlgetBusy	235	TWlrxFrame	237
TWlgetCMD	235	TWlrxLen	43
TWlgetRdy	235	TWlrxStat	237
TWlgetRxStat	235	TWlrxStatReg	43
TWlgetTxStat	235	TWlsetBusy	237
TWlinp	235	TWlsetGC	237
TWlinpC	235	TWlsetRdy	237
TWlinpD	235	TWlsetSlaveAddr	238
TWlinpE	235	TWlslave	42
TWlinpF	235	TWlstat	238
TWlinpP	236	TWlstatC	238
TWlinpPC	236	TWlstatD	238
TWlinpPD	236	TWlstatE	238
TWlinpPE	236	TWlstatF	238
TWlinpPF	236	TWltxAdr	43
TWlmaster	14, 15, 19, 23, 42, 44	TWltxBroadCast	238
TWlmode	42	TWltxBuff	43

TWItxClear 238
 TWItxFrame 238
 TWItxLen 43
 TWItxStat 238
 TWItxStatReg 43
 TWriteMode 22
 TwzBroadcast 44
 TwzNDTimeOut 44
 TwzPacketReceive 44
 TwzPriority 44
 TwzSocket 44
 TwzSocketNDAck 44
 TwzSocketProtocol 44
 TwzSocketSWS 44
 TwzStatus 44
 TxBuffer 29, 134
 TxBuffer1 134
 TxBuffer2 134
 TxBuffer3 134
 TxBuffer4 134
 TxBufferC0 134
 TxBufferC1 134
 TxBufferD0 134
 TxBufferD1 134
 TxBufferE0 134
 TxBufferE1 134
 TxBufferF0 134
 TxBufferF1 134
 type 88

- U -

uDelay 239
 uDelay_1 239
 uFix64 4
 uMIRF24 24
 unit 88
 UnLock 239
 until 88
 UpCase 239
 UpperCase 239
 userdevice 89
 UserPort 38
 uses 88, 89
 using 89
 UsrDevPtr 239

- V -

val 89
 ValueInRange 239
 ValueInTolerance 240
 ValueInToleranceP 240
 ValueTrimLimit 240
 var 89
 variant 89

- W -

WaitDeviceFree 240
 WaitPipe 240
 WaitSema 240
 WatchDogStart 240
 WatchDogStop 241
 WatchDogTrig 241
 while 90
 with 90
 WithIn 241
 WizNet 40, 44
 word 90
 word64 90
 WordToBCD 241
 Write 241
 WriteLn 241
 wzAcceptConnection 241
 wzClientConnected 242
 wzConnect 242
 wzCreateSocket 242
 wzDisConnect 242
 wzDNSQueryHost 242
 wzFreeSocket 242
 wzGetLastError 242
 wzGetSocketState 243
 wzInit 243
 wzInitSocket 243
 wzListen 243
 wzNet4 44
 wzPacketReceived 243
 wzPrioAuto 44
 wzPrioHigh 44
 wzPrioLow 44
 wzPrioMedium 44

wzPrioResume 44
wzPrioSuspend 44
WzPrioVeryHigh 44
wzReceiveBuffer 243
wzReInitSocket 243
wzReset 244
wzResumeReceive 244
wzsBufferParam 44
wzSendBuffer 244
wzSetDNSserver 244
wzSetGatewayAddr 244
wzSetHwAddr 244
wzSetIPAddr 244
wzSetPriority 245
wzSetRetryCount 245
wzSetSNTPserver 245
wzSetTimeOut 245
wzsInitFailed 44
wzsInvalidHandle 44
wzsListenFailed 44
wzsNoErrors 44
wzsNotInitialized 44
wzSNTPQueryDateTime 245
wzSocks 44
wzsSendFailed 44
wzsSockClosed 44
wzsSockCloseWait 44
wzsSockClosing 44
wzsSockConnected 44
wzsSockListen 44
wzsSockRaw 44
wzsSockUDP 44
wzsTimeOutErr 44
wzTelnetClose 245
wzTelnetConnected 245
wzTelnetCreate 246
wzTelnetEcho 246
wzTelnetFree 246
wzTelnetGetClient 246
wzTelnetGetState 246
wzTelnetIdleTimeout 246
wzTelnetListen 246
wzTelnetRead 247
wzTelnetWrite 247
wzTelnetWriteLn 247

- X -

xAKNLocalPort 40
xAKNRemotePort 40
xor 90

If you want to write your applications with greek, kyrillic or sanscrit characters or language, then you must use C

If you need readable sources then use AVRco and Delphi

(c)1996-2011 ***E-LAB Computers***
Grombacherstr. 27
D74906 Bad Rappenau
Germany

Tel. 07268/9124-0
Fax. 07268/9124-24

Internet: www.e-lab.de
e-mail: info@e-lab.de
