
FastAVR
Basic Compiler for AVR
Created by: MICRODESIGN Bojan I.
www.FastAVR.com

User Manual

FastAVR

Basic like Compiler

Version 4.0.0

User Manual

March 2004 by MicroDesign

1. Introduction

*My sincere thanks to **Michael Henning** and **Steve Childress**, for theirs assistance in writing Help and Manual.*

Welcome to **FastAVR** – the best Basic-like language compiler for the Atmel AVR family of 8-bit microprocessors!

Basic is a High Level Language, much easier to learn and understand than assembler or C.

FastAVR Basic is a language consisting of most of the familiar BASIC keywords but has been significantly extended with many additional very useful functions, like LCD, I2C, 1WIRE, Keyboards and many others!
FastAVR Basic Compiler has been specially written to fully support the programmer's needs to control the new AVR Microcontroller family!
FastAVR Basic Compiler allows complex operations to be expressed as short but powerful Keywords, without detailed knowledge of the CPU instruction set and internal circuit architecture. However, the processor-s data sheets remains the main source anyway.
FastAVR Basic Compiler hides unnecessary system details from the beginning programmer, but also provides assembler output for advanced programmers!
FastAVR Basic Compiler enables a faster programming and testing cycle.
FastAVR Basic Compiler allows the structure of the program to be expressed more clearly.

1.1. Compiler Operating System Compatibility

Windows 98SE
Windows NT 4
Windows 2000
Windows XP

1.2. AVR chip supported

The most current list of AVR chips supported by the compiler is in the readme file that accompanies the installation software. Refer to Atmel's specifications or web site for descriptions. At this User's Guide release, the Atmel AVR chips supported are as follows.

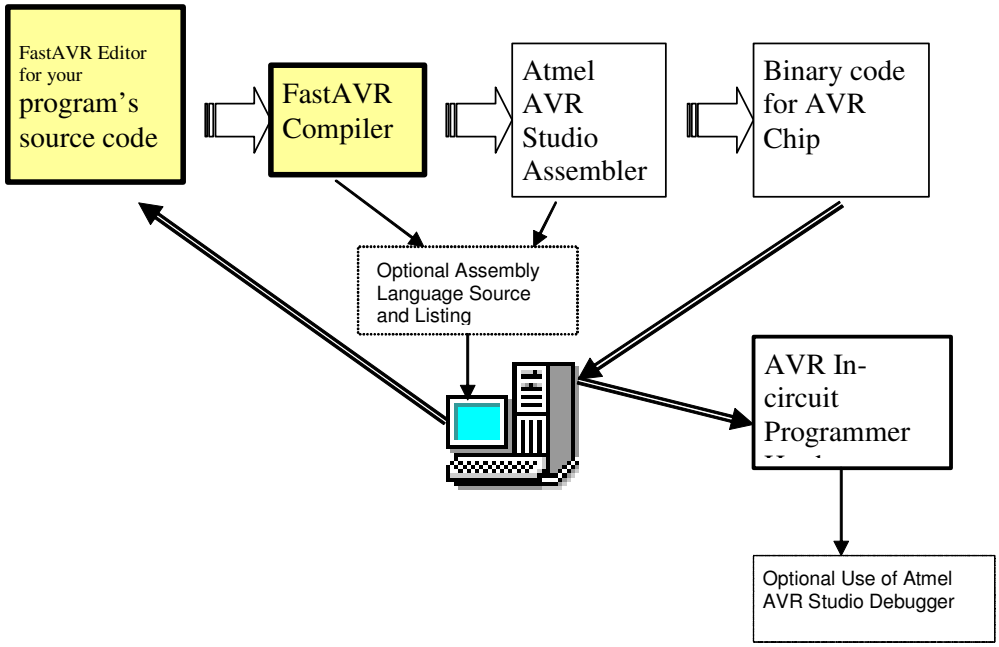
2313
2323
2343
2333
4433
4414
8515
4434
8535
8534
ATiny13
ATiny2313
ATiny26
ATmega161
ATmega163
ATmega103
Atmega8 (see note 1)
Atmega16 (see note 1)
Atmega32 (see note 1)
Atmega323 (see note 1)
ATmega64 (see note 1)
ATmega128 (see note 1)

 For processors with RAM sizes of 256 bytes or less, such as the 2313, the compiler will create much smaller code sizes – typically 30% or better. This is done by generating code with addresses that are 8 rather than 16 bits – eliminating much needless code.
Most other Basic and C compilers do not provide this important optimization for small-RAM microprocessors.

Note 1: Compiler support for these processors requires some user-written definitions of some new register names and special coding for certain new features.

1.3. Development Environment

The **FastAVR** Integrated Development Environment (IDE) runs on the Microsoft Windows operating systems. The IDE includes a user friendly, language-context based editor for easliy creating and altering program modules. The IDE, shown below, uses the Atmel AVR Studio software (freeware from Atmel and inexpensive and widely available in-circuit AVR chip programming hardware.



FastAVR Development Environment

The **FastAVR** user interface makes the use of the editor, compiler, and Atmel Studio assembler and chip programmer very easy – there is a single window to control the process. The Atmel Studio's user interface is not needed in most cases, with **FastAVR**.
The Atmel Studio software is freely available from the Atmel Web site.
There are several in-circuit programmers (ISP) for the AVR chip family. Choose one that is compatible with the ISP software you intend to use. Many of the inexpensive hardware ISP products are compatible with the ISP in the AVR Studio package. A PC parallel port interface to the ISP hardware is commonly used.

2. FastAVR Language Reference

2.1. Source code / File Data

FastAVR source code is plain text.

The editor within **FastAVR** should be used to manage source code and assure the code is properly tab-indented. Source code is displayed by the editor with color highlighting. These denote different keywords and variable types in the program. The editor supports indenting and un-indenting blocks of code.

2.2. Source code - Structure

Source code must be written in the following order:

1. Meta-statements (compiler directives) - where the keywords begin with "$"
2. Declaration of Subroutines (also known as procedures), Functions or Interrupt service routines, using the DECLARE statement

    Note that Subs, Functions and Interrupt service routines must be declared before they are referenced or coded.

3. Declaration of variables – scalars and arrays, using the DIM statement
4. Constants – their definitions
5. Programm body

2.3. Statements - multiple per line

One line of text in the source code may contain multiple statements. Each statement is separated from the next with a colon ":".

Example:

```
A = B/2: Print A
```

A statement may not be continued to a second line.

2.4. Comments

All text after a single quote is commentary. A code statement must be completed prior to beginning a comment.
A single quote may be the first character in a line, making the entire line a comment.

```
' this is comment
```

Multi-line comments are also supported:

```
'(...
...
...
)'
```

2.5. Names - Symbols

User-defined constants, variable names, line labels (for GOTO) and I/O aliases (see $DEF) must not begin with a digit and may contain the letters A-Z, a-z, the digits 0-9 and the underscore "_". The compiler is case-insensitive, so "Abc" and "abC" and "ABC" are the same name.

Names must be no longer than 31 characters.

Examples of valid names:

a, A, abc, abc, abc_123

 Avoid use of a leading underscore "_" to reduce the chance of conflicts with compiler-generated run-time symbols . User-defined names must differ from the reserved words in FastAVR. These include the compiler directives, AVR register names, AVR instruction names, etc.

To reduce mistakes in the use of types, some programmers use naming conventions such as:
Constants are in UPPERCASE
Variables use the first few characters of every name depict the type, such as:

```
Const strTITLE = "King Of The World"
Const STRLENMAX1 = 20
Dim strSomeName As String * STRLENMAX1
Dim SomeName As Byte  ' no prefix implies Byte
Dim bSomeName As Byte  ' or be explicit
Dim bitSomeName As Bit
Dim wdSomeName As Word
DimM intSomeName As Integer
Dim lngSomeName As Long
```

The capital letters help readability.

2.6. Types

2.6.1. FastAVR language General

FastAVR is a compiled language. Thus, all variables must be defined prior to their use.
The AVR chips use two independent memory address spaces: One for program code in ROM and one for random access, read/write memory (RAM). EEPROM storage is handled differently than program ROM or RAM. Code cannot be executed from RAM. Constants can be retrieved from program ROM using special instructions or meta-directives when the constant is declared.


The AVR chip family uses separate memory for code and for data. You must make known to the compiler which data is to go into which memory space. The compiler of course places executable code automatically. Where to store constants is the planning challenge. Programmers not accustomed to this architecture often write code assuming that constants and variables are in the same address space. In some cases, the compiler cannot detect the coding mistake. In AVR chips with EEPROM storage, decisions must be made as to which constants and infrequently changing variables are to be stored in that address space.

There are compiler directives (meta-statements) to tell **FastAVR** how constants should be stored:

- Store a constant in program memory (typically, flash ROM).
- Store a constant in program memory but copy them to RAM during program startup. With this method, the programmer accesses constants using the same coding methods as for variables, as in traditional microprocessor architectures. This costs extra storage space and program boot-up initialization code and execution time.
- Store constants or variables in EEPROM. These are read or written using run-time procedures rather than instructions to directly address EEPROM space as is done for RAM and ROM accesses.

2.6.2. Type Conversions

In some languages, such as Microsoft's Visual Basic, the compiler and run-time libraries do automatic type conversions, e.g., to/from string/arithmetic. This permits the programmer to be generally unconcerned about mixed types in an expression or as arguments to functions and subroutine procedures. In embedded microprocessors, this is not done for reasons of code size and speed optimization. The embedded microprocessor programmer must beware of mixing types in the same expression, and the type of the variable receiving the result of an expression. The compiler will not produce warnings in all cases. For example, adding a **FastAVR** "word" to an "Integer" with the result going to an integer or word type can produce different results, based on the programmers assumptions about signed or unsigned arithmetic.

Type Conversions – In Assignments

There are no compile-time type conversions built into the compiler for expressions. Any such conversions must be done by coding such in-line or via user functions.
In assignment statements such as the below, if B is a **Byte** variable and w is a **Word**:

```
b = w
```

Then b receives the low 8 bits of w.
That is, the value to the right of the "=" is converted to the type of the variable to the left of the "=".

If w is an **Integer** with a value of, say, -2, then b will become &hFE or 254.

In this assignment:

```
w = b
```

the low 8 bits of w receive b, and the high 8 bits of w will be zero.

In type conversions such as these, with signed numbers (integer and long), the programmer is responsible for accommodating the possibility that the sign is ignored and a multi-byte 2's complement number can become fragmented.

Type Conversions – Implicit

In statement with an expression who's results are not assigned to a specified type, such as:

```
Print b+w
```

The run-time code will make the result be the type of the **left-most** name. This name may refer to a variable or a constant.

Type Conversions – Bit to Byte or Word or Integer

The 0 or 1 value of the bit variable is placed in the least significant bit.

Type Conversions – Byte or Word or Integer to Bit

A Bit variable to the left of an "=" receives the least significant bit of the value to the right of the "=".

 To change a Bit variable based on the result of an expression such as X > 0, use an IF statement.

2.6.3. Types Arithmetic

FastAVR supports arithmetic (pronunciation: "arith-metic") types of variables as shown in the table, below. Unsupported types are shown for clarity.
For signed numbers, two's complement arithmetic is used by the AVR chips and by the compiler. Thus, –1 in any signed type is all 1's and –2 has the least significant bit = 0.

Type	Size (Bytes)	Range
Bit	1 bit	0 or 1
Byte (unsigned char)	1	0-255
Integer (signed)	2	-32767 to +32767
Word (unsigned)	2	0 to 65535
Long (signed)	4	-2147483648 to +2147483647
Float	4	-2.350987E-38 to +8.507062E+37

The 32 bit signed integer type is a Long in this compiler.
Use of Longs or Float should be minimized because of the large code sizes required.

2.6.4. Assigning Statements

Arithmetic variables

An arithmetic variable may receive a value from
- A constant
- A variable
- The result of a Function call returning an arithmetic type
- An arithmetic expression which may include a function

The assignment statement is of this form:
<variable> = <expression or constant or function>

Examples:
```
Gamma = 123
Gamma = X+123
Gamma = FunctionName()
```

String Variables

A string variable may receive a value from
- A string constant
- A string constant stored in an array (in FLASH)

The assignment statement is of this form:

<variable> = <expression or constant or function>

Examples:
```
String1 = String2
```

I/O Register Variables, Set and Reset

A special form of assignment statement exists for AVR register bits:
Set <I/O bit name>
Reset <I/O bit name>
Where <I/O bit name> refers to an AVR register and bit.

Examples:
```
Set PORTB.0     'place a 1 in port B bit 0 (least significant bit)
Reset PORTB.7   'place a 0 in port B bit 7 (most significant bit)
```

I/O Register Variables in General Assignment Statements

A special form of assignment statement exists for AVR register bits:
<varname> = <I/O bit name>

Example:
```
b = PORTB.1    'variable b is set to 1 or 0 depending on latched port  B bit 1
b = PINB.1     'variable b is set to 1 or 0 depending on momentary state of port
B bit 1
```

The I/O register variables may also be used in IF statements, but only where the comparison operator is the "=" operator; i.e., IF PORTB.0 > 0 is not permitted.

2.7. Constants

2.7.1. Constants - Scope

Constants apply to all source statements in all files included in the compilation.

2.7.2. Constants- Numbers and Their Syntax

Decimal

Constants which are decimal (base 10) are written as follows:
Decimal positive number: <digits>
Decimal negative number: -<digits>

Where
<digits> are the digits 0-9

Hexadecimal (base 16)

Constants which are hexadecimal (base 16) are written as follows:
&h<digits>

Where
<digits> are the digits 0-9, A-F

Examples

```
&h01 is one, &h10 is sixteen, &h0F is fifteen,
&hFFFF if stored into a Word or Long variable is 65535,
&hFFFF if stored into a variable of  type Integer is -1,
```

Binary (base 2)

Constants which are base (base 2) are written as follows:
&b<digits>

Where
<digits> is 0 or 1

Example

```
&b1001  'is nine.
```

2.7.3. Constants, Arithmetic - Declaring

Constants must be declared at the beginning of the program, before or after declaring variables. The constant's name must be unique from variables and compiler keywords.

Arithmetic constants use this syntax:

```
Const <constname> = <constant>
```

Where

<constname> is a *previously unused* valid variable name (see 6), and

<constant> is one of the following:

1. A decimal constant such as 1 or –1 or 99
2. A hexadecimal number such as &hFFFF
3. A binary number such as &b0110
4. A Scientific notation decimal constant such as -2.1479231E-23
5. A *previously defined* constant's <varname>
6. A simple math expression.

 An expression to the right of the "=" is permitted.

It is the programmer's responsibility to use numeric constants in a manner consistent with the storage variable's type. For example, a Byte variable cannot store a Const x = 256. A compile-time warning cannot be assured.

2.7.4. Constants, Arithmetic Arrays - Declaring

One-dimension arrays of constants may be stored in FLASH memory and accessed from flash at run time. The following syntax is used:

```
Dim <varname> As Flash <type>
```

Establishes the existence of the array. To fill (initialize) the array at compile time, the following is used, after the DIM statement.

Examples:

```
Dim abc As Flash Byte

… later in program …
ABC = 1,2,3,4,
 5,6,7  ' Stores the binary codes for these numeric values (not ASCII)
```

Note: continues on second line due to comma after the "4".

```
Dim DigitsASCII As Flash Byte
DigitsASCII  = "1234567890"     ' access with  x = DigitsASCII(n)
```

The compiler will create different code to access constants in FLASH than for constants stored in RAM.

Constants in Flash MUST be initialized at the END of Program!

See also: Constants, String Arrays

2.7.5. Constants, String

Strings in FLASH memory

Strings may be stored in FLASH memory as follows:

```
Dim String1 As Flash String
String1 = "Hello World"
```

Or

```
Const String1 = "Hello World"
```

The string is stored in FLASH memory with a null byte terminator. The name **String1** refers to that constant in FLASH. Because it is a constant, the DIM statement needed no size declaration.

Constants in Flash MUST be initialized at the END of Program!

2.7.6. Constants, String Arrays

String constants may be placed into arrays in FLASH as follows.

```
Dim Strings1 As Flash String
…
Strings1 = One, Two, Three, "Only Three"
```

This places four null-terminated strings in Flash memory with "Strings1" referring to the first element of the array of strings. The following code will access the strings

Constants in Flash MUST be initialized at the END of Program!

Example 1

```
Dim s As String * 10
s = Strings1(2) ' copy from FLASH to RAM
```

Example 2:

```
Print Strings1(n) ' where N is 0 to 3
```

2.8. Variables

2.8.1. Variables - Scope

Variables declared in the program heading are global to all source statements in all files included in the compilation. BIT variables are stored in processor registers and are thus global.

2.8.2. Variables, Arithmetic - Declaring

Non-dimensioned variables must be declared at the beginning of the program, after declaring constants and before code. The variable's name must be unique from constants and compiler keywords.
It is the programmer's responsibility to use variables in a manner consistent with the storage variable's type. For example, a Byte variable cannot store x = 256. A compile-time warning cannot be assured.
Variables larger than one byte are stored in successive bytes. These types are not aligned on word or long address boundaries. This is because the AVR chips address memory only as bytes.

Arithmetic variables use this syntax:

```
DIM <varname> AS <type> [AT <location>] [, …]
```

Where

<varname> is a *previously unused* valid variable name (see 6), and
<type> is one of the following:

Type	Storage	Bit Usage
Bit	One bit	One bit
Byte (unsigned char)	One byte	8 bits
Integer (signed)	Two bytes	Two's complement, 16 bits
Word (unsigned)	Two bytes	16 bit positive number
Long (signed)	Four bytes	Two's complement, 32 bits
Float	Four bytes	IEEE
String	Len + 1	

<location> if given, is a constant defining the memory address at which the variable may be stored.

Examples:

```
Dim somevariable As Byte
Dim B As Byte
Dim This_123 As Word, delta_gamma As Bit
```

 If the "AT" option is used, all subsequent variables are stored at successive locations. The use of "AT" is error prone if used after other variables have been declared and is not recommended.

2.8.3. Variables, Arithmetic - Run-time Type Conversions:

Some forms of type conversions are done by the compiler at run time, in storing a variable's value. Generally, the programmer should not assign a value to a variable which is from an expression or variable of a different type, or which is too large or differs in signed vs. unsigned. The following table summarizes:

Variable Type	**Assignment Type**	**Result**
Bit	Byte, Integer, Word or Long	Valid for all but Long (LSB used)
Byte (unsigned char)	Integer, Word or Long	Least significant byte
Integer (signed)	Word or Long	Least significant two bytes
Word (unsigned)	Word or Long	Least significant two bytes
Long (signed)	Word	Least significant two bytes
Float		

2.8.4. Variables, Arithmetic - Arrays

One dimension Arrays of arithmetic variables type **Byte, Word and Integer** are supported.

Syntax: DIM <varname>(n) AS <vartype> where <vartype> is as listed above.
"n" must be a numeric or previously defined named constant

Arrays can be initialized with:

```
ArrName=(d0, d1, d2, .....)
```

or with

```
MemLoad (VarPtr(n), 4, 4, 4, 15, &hff, &hff)
```

Example:

```
Dim someBytes(aconstant) As Byte
```

It is the programmer's responsibility to declare and use array sizes so that they will fit into the available RAM.

 The compiler does not create run-time error checking for array index validity. For example, the code x = a(999) is invalid for an AVR 2313 chip.
Do not use the same name for an array and for a non-dimensioned variable.

2.8.5. Variables, String

Strings in FastAVR assume 8 bit ASCII characters. Strings are null-terminated. For string constants, the compiler inserts a null. For variables, the programmer must ensure that the string's declared length allows for one extra character for the null.

String variables (see below) are declared with a fixed length.

 The programmer must not permit a string variable to receive a value longer than the length of the variable, minus one. If this happens, subsequent memory storage will be overwritten and/or a string may not be null-terminated. This causes difficult to debug problems. String length limit checks are not done at run time for reasons of efficiency.

String variables are blocks of RAM into which run-time code places ASCII characters terminated by a null. They are declared as follows:

```
Dim <varname> As String *n
```

where
n is a numeric constant such as 8 or 12

This form is permitted:
DIM <varname> AS STRING * <ConstantName>

where **ConstantName** is the name of a constant.

Examples:

```
Const MAXLEN = 32
Dim LastName As String * MAXLEN
Dim FirstName As String * 16
```

In both cases, the storage allocated is one greater than the value of the constant, to make room for the terminating null.
The "*" does not mean the same thing as it does in other languages (e.g., operating with a pointer).

 For reasons of efficiency, the compiler does not generate string length checking code. The programmer must assure that a string variable's storage size is sufficient, including the necessary terminating null byte. The run-time libraries truncate a string to fit the defined size of a string variable.

2.8.6. Variables, String Arrays

One dimension Arrays of Strings variables are supported.

Syntax: DIM <varname>(n) AS STRING * <ConstantName>
"n" must be a numeric or previously defined named constant

Arrays can be initialized with:

```
ArrName=(StrConst0, StrConst1, StrConst2, .....)
```

or with

```
ArrName=(StrConst)
```

to init all elements to StrConst!

Example:

```
Dim MyString(10) As String * 7   ' ten Strings seven characters each (total 8)
```

If there is only one string constant then the whole array will be initialiyed to this String Constant!

```
MyString=(" ")                   ' will init all array elements to " "

MyString=("123", "ABC", "    ")  ' will init first three elements
```

It is the programmer's responsibility to declare and use array sizes so that they will fit into the available RAM.

 The compiler does not create run-time error checking for array index validity. For example, the code x = a(999) is invalid for an AVR 2313 chip.
Do not use the same name for an array and for a non-dimensioned variable.

Note also that Strings itself acts like arrays of Bytes

`s3(4)` is fifth element from String s3 (starting with index 0).

2.9. Declarations - Procedures and Functions

2.9.1. Declaring Procedures

Procedures are like subroutines. They receive passed variables as parameters and return values only by affecting global variables.
Procedures must be declared in the program heading prior to coding the procedure itself or referencing the procedure in code.

 The code for the procedure itself must appear after the entire heading section of the program.

The syntax is:

```
Declare Sub <name>()
```

Or

```
Declare Sub <name>(<parameter> As <type> [, …..])
```

Where <name> is any valid name (see 6) and matches exactly the name used in coding the procedure (a.k.a. subroutine) itself, later in the program.
And <parameter> is the **dummy name** of a variable,

NOTE: Do not choose a dummy variable name which is the same as the name of a global variable in the program;

And <type> is the type of the dummy variable as expected by the code for the procedure.

Type BIT cannot be passed.

The order of the parameters is fixed. The order used in the DECLARE statement must be the same order used in the actual code for the procedure.

Examples:

```
Declare Sub gamma()
Declare Sub min(a As Integer, b As Integer)
```

 Run-time type checking is not performed. If a variable of a type other than that in the DECLARE is passed, unpredictable results or branching may occur.

2.9.2. Declaring Functions

Functions are like procedures but return values. A function, like a variable, has a particular type.
Functions must be declared in the program heading prior to coding the procedure itself or referencing the procedure in code.

 The code for the function itself must appear after the entire heading section of the program.

Function names are used to the left of an "=" in an assignment statement.
The code for the function itself must appear after the heading section of the program.
The syntax for a function is the same as for a procedure except as noted below. See 7.

```
Declare Function <name>() As <return type>
```

Or

```
Declare Sub <name>(<parameter> As <type> [,….]) As < function type >
```

Where <name> and <parameter> and <type> are as explained for DECLARE SUB,
And <function type> is the type returned by the function, as if the function were an ordinary variable.

Functions may be of type **Byte**, **Word**, **Integer, Long or Float**.

 Functions of type **String** or **Bit** are not permitted.

The value returned by the function is that assigned using the function's name in the left side of an assignment statement which is within the function.
Function names are used to the left of an "=" in an assignment statement.

2.9.3. Declaring Interrupts

Interrupt procedures are subroutines used to process a hardware-generated interrupt.

Interrupt procedures are identical to Procedures (see 7), except:

- They must never be called from statements in the code
- They are invoked by the AVR chip's interrupt hardware

```
Declare Interrupt <interrupt cause name>()
```

Where <interrupt caise name> is the symbolic name of an interrupt cause for the AVR chip targeted by the compilation.

 A list of interrupt cause names is in section 11.

The parentheses after the name are necessary.

Example:

```
Declare Interrupt Ovf1()
```

Means that later in the code there is an interrupt procedure coded for the timer 1 overflow interrupt source.

2.10. Statements

2.10.1. Statements, Arithmetic Expressions

An arithmetic assignment statement takes the following form:

<varname> **=** <expression>
or
Print <expression> ' a special assignment to convert to a string type

where <expression> is
<name**>** <arithmetic operator> <name > […]

where
<name> is the name of an arithmetic variable or arithmetic constant
<arithmetic operator> is an arithmetic operator (see 9)

Parentheses may be used in expressions to control the order of operations.

Examples:

```
A = B
A = B+1
A = B/2
A = B+(C/2)+D*E
```

2.10.2. Statements, String Expressions

An string assignment statement takes the following form:

<varname> **=** <string expression>
or

Print <string expression> ' a special assignment

Where <string expression> is

- A string constant
- A string variable which has received a value
- A built-in (library) string function defined later in this document. User-written functions may not return type string (or Bit).

The variable receiving the result should be large enough to store the entire result. If not, the result is truncated to make room for the automatically-inserted terminating null byte.

 If the name to the right of the "=" is a string constant, or if the Print statement's argument is a string constant, then note: Special code is generated to copy the string from FLASH to RAM.

2.11. Program Flow

2.11.1. Statement, Do - Loop

The DO statement creates a loop. DO/LOOP loops may be nested.
The following forms are permitted:

1. Infinite loop
```
DO
   <statements>
LOOP  ' Only way out is a Exit Do and GoTo
```

2. Conditional loop
```
DO
LOOP WHILE <condition>
```

Where <condition> is an expression evaluating to true or false.

Examples:

```
WHILE x=0
WHILE x < 8
WHILE a >= b
WHILE String1 <> "ABC"
```

In either form, the statement

```
EXIT DO
```

within the loop causes the statement below the LOOP statement to be executed.

Example:

```
DO
   <statements>
   IF <condition> THEN
      EXIT DO  ' cannot be on same line as THEN
   END IF
LOOP
```

As in non-looping code, interrupts may occur and execute an interrupt procedure.

2.11.2. Statement, While - When

The WHILE statement creates a loop, similar to DO.

The form is:

```
WHILE <condition>
   <statements>
WEND
```

The statement

```
EXIT WHILE
```

within the loop causes the statement below the WEND statement to be executed.

As in non-looping code, interrupts may occur and execute an interrupt procedure.

2.11.3. Statement, For - Next

The FOR statement creates a loop as in traditional BASIC. FOR/NEXT loops may be nested.
The forms are:

1. Increment by +1

```
FOR <iteration variable> = <initial value> TO <final value>
```

```
    <statements>
NEXT
```

Where <iteration variable> is a numeric variable large enough to contain the <final value>, and
<initial value> is a constant or variable who's value is stored in <initial value> prior to the first iteration, and
<iteration value> is incremented by one after all <statements have been executed>, and
<final value> is a constant or variable who's value is compared with <iteration variable> after all
<statements> are executed.

At least one iteration is assured.

2. Increment or decrement by arbitrary amount

```
FOR <iteration variable> = <initial value> TO <final value> STEP <delta>
    <statements>
NEXT
```

Where <delta> is a positive or negative numeric of numbers who's value is added to <iteration value>. A named constant or variable may not be used. Instead, use the DO statement.

2.11.4. Statement - If

In this section, the notation
<relation>
means
<expression> <relational operator> <expression>

Example of a <relation>:
A=B

 Note that <expression> can be a special I/O register reference such as "PINB.2".

Single line form

The syntax has these forms:

1. IF <relation>THEN <statement>
   Statement is executed if <relation> is true.
   Example: IF a > b THEN Print "yes"
2. IF <relation> <logical operator> <relation> THEN <statement>
   Statement is executed if all <relation> are true.
   Example: IF a > b AND a < 10 THEN Print "yes
3. Same as above, but with additional logical operators.
   Example: IF a > b AND a < 10 OR a > 100 THEN Print "yes

Parentheses may be used for clarity.

The ":" for multiple statements per line may not be used in the single-line IF.

 The ELSE clause is not permitted in a single-line IF statement.

Multi-line form

The syntax has these forms:

1. IF <conditional>THEN
   <statements>
   END IF
2. IF <conditional>THEN
   <statements>
   ELSE
   <statements>
   END IF
3. IF <conditional>THEN
   <statements>
   ELSEIF <conditional> THEN
   <statements>
   END IF

4. IF <conditional>THEN
   <statements>
   ELSEIF <conditional> THEN
   <statements>
   ELSE
   <statements>
   END IF

2.11.5. Statement - Select Case

Selects a block of statements from a list, based on the value of an expression or a variable.

In the below, the notation
<value>

means

- a variable of the same type as in the SELECT CASE statement, including type String,
- a constant of the same type as in the SELECT CASE statement, including type String,

Syntax forms:

```
SELECT CASE <variable>
   CASE <value>
      <statements>
   CASE <value> TO <value>
      <statements>
   CASE <relational operator> <value>
      <statements>
   CASE ELSE
      <statements>
```

```
END SELECT
```

The CASE ELSE may be omitted.

The keyword "TO", above, is normally used with numeric values. It is the same as
<value> <= <value>.

If <value> is a string, the TO operator tests for equality.

2.11.6. Statement - Goto

Syntax:
GOTO <label>

Where <label> is a valid name as in variable name, and that name is the same as the name used in a label statement.

A label statement is a symbol which first in the line of code and ends with a colon:
<name>:

where the name is not a reserved keyword

Example:

```
Here:
      GOTO Here
      GOTO There
There:
```

Good programming style is to use GOTO sparingly, typically to branch out of a deeply nested conditional or nested loops.

Warning: The scope of a label is global. A GOTO can be coded to jump into a SUB or FUNCTION, or out of such. This will corrupt the stack pointer management. A GOTO in a SUB or FUNCTION should reference only labels in the same SUB or FUNCTION. A GOTO outside any SUB or FUNCTION should not reference a label inside a SUB or FUNCTION. The compiler does not enforce these rules.

2.11.7. Statement - On X Goto

Description:
Jumps to one of listed line labels, depending on value of a numeric variable or numeric constant, shown here as "X". Compiles to code smaller and faster than an IF or Case statement.

Syntax:

```
On x GoTo LabelA, LabelB, LabelC [, …]
```

If numeric variable x is zero, then the program continues at line label "`LabelA`", if 1 then to `LabelB`, and so on.

 The programmer must ensure the validity of the variable (X in this explanation). The value must be in the range of 0 to n-1, where there are n choices in the statement. In the example, X must be in the range of 0 to 2. The line labels

Example:
```
DIM ABC As Byte

ABC = 1
On ABC GoTo Label0, Label1, Label2
<other code>
LabelA:
   Print "ABC was 0"
LabelB:
   Print "ABC was 1"
LabelC:
   Print "ABC was 2"
```

2.11.8. Statement - On X Sub()

Description:
Calls one of the listed Sub(), depending on value x. Compiles to code smaller and faster than an IF or Case statement containing Sub Calls.
This is similar to On X GOSUB lable1, label2, … in other languages.

The action taken is the same as shown for **On X GoTo** except that the Sub() is called whereas On X GOTO does a jump, not a call. On return from the Sub, the code following the
On X Sub() is executed after the Sub() returns.

 The call to the Sub() must include no parameters within the parentheses.

 The programmer must ensure the validity of the variable (X in this explanation). The value must be in the range of 0 to n-1, where there are n choices in the statement.

Example:
```
DIM ABC as Word
DECLARE SUB Foo()
DECLARE SUB Bar()
ABC = 1
DO
   On ABC Foo(), Bar()
   Print "Returned From Sub"
LOOP

SUB Foo()
   Print "ABC was 0"
END SUB

SUB Bar()
   Print "ABC was 1"
END SUB
```

2.12. Compiler and Limitations

FastAVR Basic Compiler translates your Basic source file into assembler code. The assembler file is then assembled with Atmel's free Assembler (AvrAsm32.exe). Of course, the generated assembler file can be edited with additional assembler statements and then recompiled!

LIMITATIONS:
While testing `Bit` variables of any kind (bit var, port.bit or var.bit) only "=" or "<>" can be used!

```
Dim b As Bit
Dim n As Byte

If b=1 Then       ' OK
If n.5=1 Then     ' OK
If PinD.5=1 Then  ' OK
If PinD.5<>1 Then ' OK

If b>0 Then       ' NOT OK
If n.5<1 Then     ' NOT OK
If PinD.5>0 Then  ' NOT OK
```

Also, if user wishes to use bitwise operators with logic, bitwise must be in parentheses!

```
If (n And 1)>5 Or b=1 Then       ' OK
```

Basic itself does not have a CAST like C does! So if the left side of an assignment is of type "Byte" then only the lower bytes of words and/or Integers from the right side of the expression are processed!

```
Byte = Word / Byte1      'wrong result
Word1 = Word / Byte1
Byte = Word1             'correct result
```

When using an expression with the Print statement, result will be the same type as **first** element in expression:

```
Dim a As Byte
Dim b As Word
Dim c As Word

Print 10+(a*b)             'Byte result - 10 is byte
Print 10+(a*a)             'Byte result

c=10+(a*b)
Print c                    'Correct result
```

2.13. Language Specific

Basic programs are written using the `FastAVR` integrated editor, just as we would write a letter. This letter, your program, is pure ASCII text and can also be opened or edited with any simple (ASCII) editor like Window's Notepad. While writing this "letter," however, we must follow the language syntax understood by the `FastAVR` Basic Compiler. Let us start with some Basic rules, following these simple practical examples. Fortunately, Basic syntax and philosophy are quite easy to understand.
So let us start!
To make the program easier to read, It is recommend that comments be used first. For example:

```
'////////////////////////////////////////////////
'///  FastAVR Basic Compiler for AVR
'///  First program using 4433
'///  Author:
'///  Date  :
'////////////////////////////////////////////////
```

As can be seen the comment starts with a single quote character ('), while the REM keyword is not supported (obsolete).
Later in the program, comments may be added in virtually every line to clarify a line purpose, such as:

```
Statements      ' make pin 4 of portd an output
```

Now we continue with some non-executable statements (also called Meta statements). The following three lines are absolutely necessary:

```
$Device=4433    'tells the compiler which chip we are using.
$Stack=32       'reserves the estimated number of bytes for the stack.
$Clock=8        'defines the crystal frequency in megahertz.
```

All configuration statements start with the character $ ($Lcd, $I2C, $key, $watchdog, ...)
For other Meta statements please refer to the Keywords list.

Our next step is declaring (dimensioning) variables.

```
Dim var As Type
```

Keyword Dim reserves space for a defined variable in SRAM according to the type of variable.
Var is the variable's name. Allowed variable names may contain any alphanumeric characters that do not duplicate Keywords. Variable names are case insensitive.

FastAVR Basic Compiler supports the following element types:
Bit - occupies 1 bit (0 to1), located in r2 and r3 internal registers, (allowing 16 "bit variables" to be defined)
Byte - occupies 1 byte (0 to 255)
Integer - occupies 2 bytes (-32768 to +32767)
Word - occupies 2 bytes (0 to 65335)
String - an additional parameter is needed to specify the length and occupies the length+1 byte because they are terminated with a zero.
Long - occupies 4 bytes (-2147483648 to +2147483647)
Float - occupies 4 bytes (-2.350987E-38 to +8.507062E+37)

```
Dim var as String*6
```

Var can be 6 characters long but occupies 7 bytes in SRAM. The 7th byte contains a zero for termination.

Optionally, the user can specify memory space for variables like:

```
Dim var as Xram Byte
```

var will be placed in External RAM (if available)
In addition, the location can be specified:

```
Dim var as Xram Byte at &h8100
```

var will be placed in External RAM (if available) at address &h8100.
Since I abandoned the Data and Lookup statements, a table of constants can be created in code memory (Flash) using the keyword Dim.

```
Dim TableName as Flash Byte
Dim TableName as Flash String
```

Length could be also declared, since some statements (Find) wants Length of the table.

```
Dim TableName(16) as Flash Byte
```

The table can later be initialized:

```
TableName = 11, 22, 33, 44, 55, 66,
            12, 13, 14, 15, 16, 17,
            23, 24, 25, 26, 27, 28

TableName = "sample string"
```

The Table is finished when no comma is encountered!
Access to table elements:

```
var = TableName(index)
```

Of course, index can be a complex expression or even a function call!
Generaly, Tables in Flash works like an Arrays and they **MUST be initialized at the END of Program!**!

`Dim` declared variables are global, so they can be reached from everywhere in the program and their value is not destroyed.

We continue with declaring Subs and Functions.

```
Declare Sub NameOfSub(parameter list)
Declare Sub Test1(a As Byte, b As Word)

Declare Function NameOfFunc(parameter list) as Type
Declare Function Test2(a As Byte, b As Byte) as Byte
```

Also, Interrupt subroutines must be declared here.

```
Declare Interrupt Ovf1()
```

This forms HEAD of PROGRAM and MUST be in this order: (!)
1. Metastatements,
2. Dims,
3. Declarations,
4. Constants definitions,
5. All other statements.

Now we can finally start with executable statements.
Usually we first initialize the system: assign the initial value of variables and/or internal registers for needed settings, define each port pin direction, etc . . .

We continue by writing the main loop, which is a never-ending loop in most cases.

```
Do
   Body of the program (statements)
Loop
```

This loop is the heart of the program and may consist of:
- other loops
- assignments
- mathematical calculations
- keywords
- calls to subs or functions, etc...

More than one statement can be written on a line, separating each statement with a colon:

```
For n=0 To 15: Print n: Next
```

However, a single statement per line with a comment is preferable for clarity.

```
For n=0 To 15      'n will run from 0 to 15
   Print n         'output n to serial port
Next
```

Many expressions are supported in FastAVR. From very basic assignments like:

```
a=5
```

To more complex like:

```
a=(b+12)*c-3*d
```

FastAVR Basic Compiler performs all math operations in full hierarchal order. This means there is precedence to the operators. Multiplication and division are performed before addition and subtractions. As an example, to ensure the operations are carried out in the order needed, use parentheses to group the operations.

Even calls to system and user functions can be factors in expressions:

```
a=5*Test(15)+Adc8(3)
```

Where Test is your function called with parameter 15 and Adc8(3) is a system function that returns an 8bit value as a result of the analog measurement on channel 3.

List of mathematical operators:

`+` plus sign
`-` minus sign
`*` asterisk (multiplication symbol)
`/` slash (division symbol)
`Mod` modulus operator

List of relational operators:

`=` equality
`<>` inequality
`<=` less than or equal
`>=` greater than or equal
`<` less than
`>` greater than

List of logical operators:

`And` conjunction
`Or` disjunction

List of boolean operators:

`And, &` boolean conjunction, bitwise and
`Or, |` boolean disjunction, bitwise or
`Xor, ^` boolean Xor
`Not` boolean complement

Other operators also have special meanings, such as:

`"` double quotation as string delimiters
`,` comma as a parameter separator
`.` period for ports or variable bit delimiters
`;` semicolon is used when more than one parameter is used (i.e., Print a; b; c)
`'` single quotation mark starts a comment

Numeric constants can be in decimal format:

```
a=33
```

in hexadecimal:

```
a=&h21          'dec 33
```

or even in bynary:

```
a=&b00100001    'dec 33
```

A Label can be used as a line identifier. Label is an alphanumeric combination ending with a colon.

```
If a=0 Then
   Goto ExitLabel
End If

Other statements

ExitLabel:          'this is a Label
```

After the main loop we write all used and previously declared subs and functions, including interrupt subroutines.

The subroutine itself starts with the keyword Sub or Function, followed by the name and parameter list (if one exists)

```
Sub Test1(a As Byte, b As Word)
Function Test2(a As Byte, b As Byte) as Byte
```

Parameter list must be identical to the declaration of the sub!

With the keyword Local we can declare local variables.

```
Local var as Type
```

Bits, Strings and Arrays are always Global!
The use and lifetime of local variables are limited to this subroutine.
The rules for Type are the same as for the Dim.

The body of Sub or Function is a complete program needed to solve a particular problem.
The Function can return a value using the keyword Return.

If you have serious trouble in programming, especially if in doubt about the compiled results, please email source files to the mailing list for support!

FastAVR HINTS!

All internal registers can be accessed direct from basic:

```
XDIV = &h05              'changing clock for Mega
MCUCR = MCUCR or &h38    'enter powerdown mode


Dim Str_0 As String*15
Dim Str_1 As String*15
Dim Str_2 As String*15
Dim Str_3 As String*15
Dim s As String*15         ' working string
Dim n As Byte

MemCopy(16, Str_0+16*n, s)  ' copies n-th string from Str_0 into string s (acts like Array of
Strings!)


Str_2="FastAVR"
n=Str_2(4)    ' n=65  Strings acts like Arrays - elements are accessable also by index
```

Happy programming!

2.14. Interrupts

All AVR interrupts are supported by FastAVR!

```
Interrupt Ovf1(), Save All
```

Interrupt service routines are just like normal subroutines. Of course, instead of using the keyword Sub we will use Interrupt. The table of short names listed below may be used for Interrupt names!
Very important is the Save x directive. Save x determines how many registers will be saved before calling the interrupt. This depends on what variables are used in the routine.
`Save 0`, will save SREG only, could be omitted,
`Save 1`, will save SREG, zl and zh only.
`Save 2`, as Save 0 plus r24 and r25
`Save 3`, as Save 1 plus r0, r1, xl and xh
`Save 4`, as Save 2 plus r0, r1, r20, r21, r22, r23, xl and xh
`Save All` will save SREG and all registers from r0 to r5 and r19 to r31

When the `Interrupt` routine is more complex, use `Save 2`, `Save 3` or `Save All`.

```
'////////////////////////////////////////////////
Interrupt Ovf1()    'simple routine, no save
Timer1=&h7000       'reloads timer1 for 10ms
Toggle PortB.2      'toggles portb.2
End Interrupt
```

When user dont know about using `Save`, start with `All` and then try the minor versions!

Here is a list of available Interrupts

```
Int    Int Type for 2313
INT0   External Interrupt0
INT1   External Interrupt1
ICP1   Input Capture1 Interrupt
OC1    Output Compare1 Interrupt
OVF1   Overflow1 Interrupt
OVF0   Overflow0 Interrupt
URXC   UART Receive Complete Interrupt
UDRE   UART Data Register Empty Interrupt
UTXC   UART Transmit Complete Interrupt
ACI    Analog Comparator Interrupt

Int    Int Type for 4433
INT0   External Interrupt0
INT1   External Interrupt1
ICP1   Input Capture1 Interrupt
OC1A   Output Compare1A Interrupt
OVF1   Overflow1 Interrupt
OVF0   Overflow0 Interrupt
SPI    SPI Interrupt
URXC   UART Receive Complete Interrupt
UDRE   UART Data Register Empty Interrupt
UTXC   UART Transmit Complete Interrupt
ADCC   ADC Interrupt
ERDY   EEPROM Interrupt
ACI    Analog Comparator Interrupt

Int    Int Type for 8515
INT0   External Interrupt0
INT1   External Interrupt1
ICP1   Input Capture1 Interrupt
```

```
OC1A   Output Compare1A Interrupt
OC1B   Output Compare1B Interrupt
OVF1   Overflow1 Interrupt
OVF0   Overflow0 Interrupt
SPI    SPI Interrupt
URXC   UART Receive Complete Interrupt
UDRE   UART Data Register Empty Interrupt
UTXC   UART Transmit Complete Interrupt
ACI    Analog Comparator Interrupt

Int    Int Type for 8535
INT0   External Interrupt0
INT1   External Interrupt1
OC2    Timer2 Compare Interrupt
OVF2   Overflow2 Interrupt
ICP1   Input Capture1 Interrupt
OC1A   Output Compare1A Interrupt
OC1B   Output Compare1B Interrupt
OVF1   Overflow1 Interrupt
OVF0   Overflow0 Interrupt
SPI    SPI Interrupt
URXC   UART Receive Complete Interrupt
UDRE   UART Data Register Empty Interrupt
UTXC   UART Transmit Complete Interrupt
ADCC   ADC Conversion Complete Handle
ERDY   EEPROM Write Complete Handle
ACI    Analog Comparator Interrupt

Int    Int Type for ATmega103
INT0   External Interrupt0
INT1   External Interrupt1
INT2   External Interrupt2
INT3   External Interrupt3
INT4   External Interrupt4
INT5   External Interrupt5
INT6   External Interrupt6
INT7   External Interrupt7
OC2    Output Compare2 Interrupt
OVF2   Overflow2 Interrupt
ICP1   Input Capture1 Interrupt
OC1A   Output Compare1A Interrupt
OC1B   Output Compare1B Interrupt
OVF1   Overflow1 Interrupt
OC0    Output Compare0 Interrupt
OVF0   Overflow0 Interrupt
SPI    SPI Interrupt
URXC   UART Receive Complete Interrupt
UDRE   UART Data Register Empty Interrupt
UTXC   UART Transmit Complete Interrupt
ADCC   ADC Conversion Complete Handle
EEWR   EEPROM Write Complete Handle
ACI    Analog Comparator Interrupt

Int    Int Type for ATmega8
INT0   External Interrupt0
INT1   External Interrupt1
OC2    Output Compare2 Interrupt
OVF2   Overflow2 Interrupt
ICP1   Input Capture1 Interrupt
OC1A   Output Compare1A Interrupt
OC1B   Output Compare1B Interrupt
OVF1   Overflow1 Interrupt
OVF0   Overflow0 Interrupt
SPI    SPI Interrupt
```

```
URXC   UART Receive Complete Interrupt
UDRE   UART Data Register Empty Interrupt
UTXC   UART Transmit Complete Interrupt
ADCC   ADC Conversion Complete Handle
ERDY   EEPROM Write Complete Handle
ACI    Analog Comparator Interrupt
TWI    Two wire interface Interrupt
SPM    Store Program Memory Ready Interrupt

Int    Int Type for ATmega16
INT0   External Interrupt0
INT1   External Interrupt1
OC2    Output Compare2 Interrupt
OVF2   Overflow2 Interrupt
ICP1   Input Capture1 Interrupt
OC1A   Output Compare1A Interrupt
OC1B   Output Compare1B Interrupt
OVF1   Overflow1 Interrupt
OVF0   Overflow0 Interrupt
SPI    SPI Interrupt
URXC   UART Receive Complete Interrupt
UDRE   UART Data Register Empty Interrupt
UTXC   UART Transmit Complete Interrupt
ADCC   ADC Conversion Complete Handle
ERDY   EEPROM Write Complete Handle
ACI    Analog Comparator Interrupt
TWI    Two wire interface Interrupt
INT2   External Interrupt2
OC0    Output Compare0 Interrupt
SPMR   Store Program Memory Ready Interrupt

Int    Int Type for ATmega323
INT0   External Interrupt0
INT1   External Interrupt1
INT2   External Interrupt2
OC2    Output Compare2 Interrupt
OVF2   Overflow2 Interrupt
ICP1   Input Capture1 Interrupt
OC1A   Output Compare1A Interrupt
OC1B   Output Compare1B Interrupt
OVF1   Overflow1 Interrupt
OC0    Output Compare0 Interrupt
OVF0   Overflow0 Interrupt
SPI    SPI Interrupt
URXC   UART Receive Complete Interrupt
UDRE   UART Data Register Empty Interrupt
UTXC   UART Transmit Complete Interrupt
ADCC   ADC Conversion Complete Handle
ERDY   EEPROM Write Complete Handle
ACI    Analog Comparator Interrupt
TWSI   Two wire interface Interrupt
SPMR   Store Program Memory Ready Interrupt

Int    Int Type for ATmega128
INT0   External Interrupt0
INT1   External Interrupt1
INT2   External Interrupt2
INT3   External Interrupt3
INT4   External Interrupt4
INT5   External Interrupt5
INT6   External Interrupt6
INT7   External Interrupt7
OC2    Output Compare2 Interrupt
OVF2   Overflow2 Interrupt
```

```
ICP1   Input Capture1 Interrupt
OC1A   Output Compare1A Interrupt
OC1B   Output Compare1B Interrupt
OVF1   Overflow1 Interrupt
OC0    Output Compare0 Interrupt
OVF0   Overflow0 Interrupt
SPI    SPI Interrupt
URXC0  UART0 Receive Complete Interrupt
UDRE0  UART0 Data Register Empty Interrupt
UTXC0  UART0 Transmit Complete Interrupt
ADCC   ADC Conversion Complete Handle
ERDY   EEPROM Write Complete Handle
ACI    Analog Comparator Interrupt
OC1C   Output Compare1C Interrupt
ICP3   Input Capture3 Interrupt
OC3A   Output Compare3A Interrupt
OC3B   Output Compare3B Interrupt
OC3C   Output Compare3C Interrupt
OVF3   Overflow3 Interrupt
URXC1  UART1 Receive Complete Interrupt
UDRE1  UART1 Data Register Empty Interrupt
UTXC1  UART1 Transmit Complete Interrupt
TWI    Two wire interface Interrupt
SPMR   Store Program Memory Ready Interrupt
```

Devices not listed have the same interrupt names!

2.15. Outputs

FastAVR Basic Compiler compiles the Basic source file in the currently active editor window by pressing the RUN button! An assembler source file will be generated if no errors are encountered!
Then Atmel's free Assembler (AvrAsm.exe) is called to generate an executable file in standard Intel Hex format! Also, Lst and Obj files are generated at the same time! The Obj file can be loaded directly into Atmel's free debugger-simulator AvrStudio!

```
Test.bas ----> Test.asm -> Test.hex, Test.obj and Test.eep (If InitEE is used!)
```

If the compiler is run while an Assembler window is active then only the Assembler will be called!

2.16. Memory Usage

With every declared variable, space is reserved in internal SRAM. The available SRAM memory depends on the chip, from 64bytes in ATiny22 to 4k in ATmega103. Except for the always needed stack space, no SRAMs20 is used by the compiler.

In addition to SRAM, AVR also has a register file from 0 to 31. These are the Compilers working space.

`Dim b As Bit` will occupy one bit from R3 and R4 internal registers! No SRAM locations are needed!
`Dim n As Byte` will occupy one byte, starting at &h60 in SRAM.

`Dim i As Integer` occupies two bytes, next to variable **n** at &h61and &h62
`Dim w As Word` occupies two bytes, next to variable **i** at &h61and &h62
`Dim s As String*5` will occupy six(6) bytes, five for variable **s** and one for the string terminator "zero". In this case **s** starts after variable **w** in position &h63.
`Dim x As Long` occupies four bytes.
`Dim f As Float` occupies four bytes.

Because the entire **AVR** family are 8-bit microcontrollers the most efficient code is obtained by using variables of type Byte.

FastAVR uses two software stacks. The first one for temporary storage and for return addresses while calling Subroutines or Functions. This stack starts at the end of SRAM and grows downward. The second stack is used to store Local variables and variables that are passed to subroutines. This stack is defined by the programmer with the Meta Statement:
`$stack=20`. This means that the stack will start 20 bytes below the top of SRAM and will also grow downward!

Each Local or passed variable to a Sub or Function uses stack.

When using conversion routines that convert a number to a string, the compiler will need additional SRAM space starting from the second stack UP. This is also true when Strings or StringsConstants are passed into Subs or Functions! Sometimes this can overlap the first stack, so some attention will be needed!

With some devices like the 8515, external memory may be added. However, because the XRAM can only start after the SRAM, which is at &H0260, the lower memory locations of the XRAM will not be used.

Most AVR chips have internal EEPROM on board. This EEPROM can be used to store and retrieve infrequently used data.

With **FastAVR,** access to this space is easy using `WriteEE` and `ReadEE` statements!
Note that each address can only be written a maximum of 100,000 times!

Numeric and String Constants do not use any SRAM, they are in code (flash)!

2.17. Assembler Programming

Assembler code may be added at any time. However, assembler programming should not be necessary since **FastAVR** will probably generate smaller code than can be done in assembler!
Also, the generated assembler file can be edited and recompiled to fine tune the whole system!

All variables are reachable from assembler, like:

```
$Asm
   sts   tip,zl
   lds   r24,tip
$EndAsm
```

tip is a global variable!

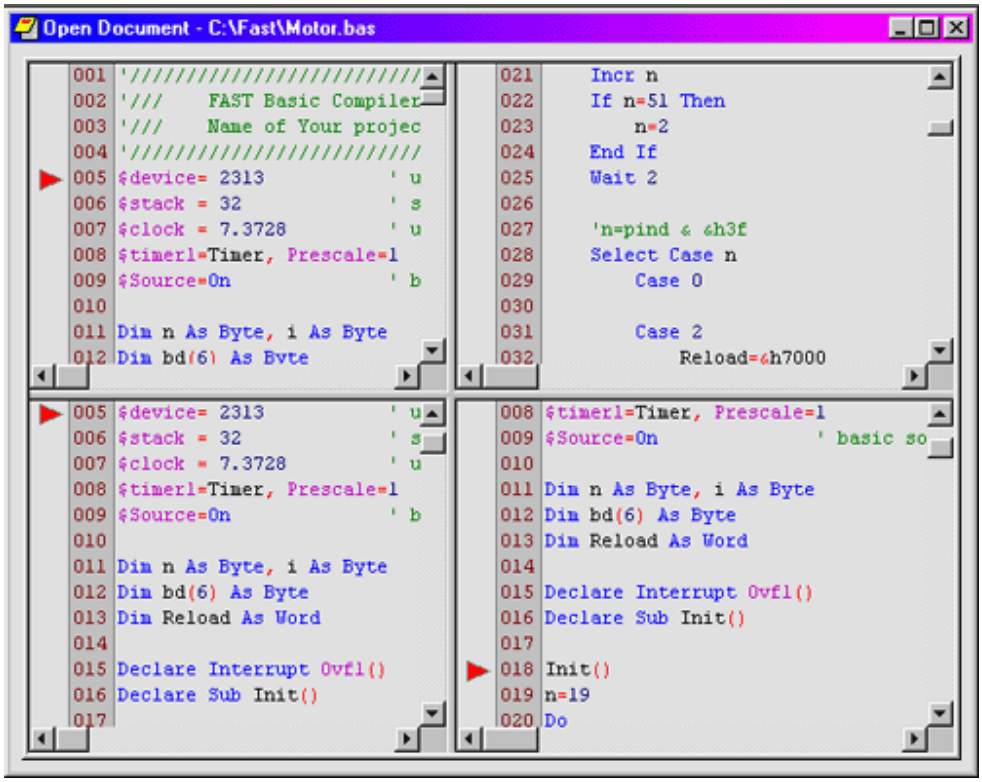
3. FastAVR IDE

3.1. Editor

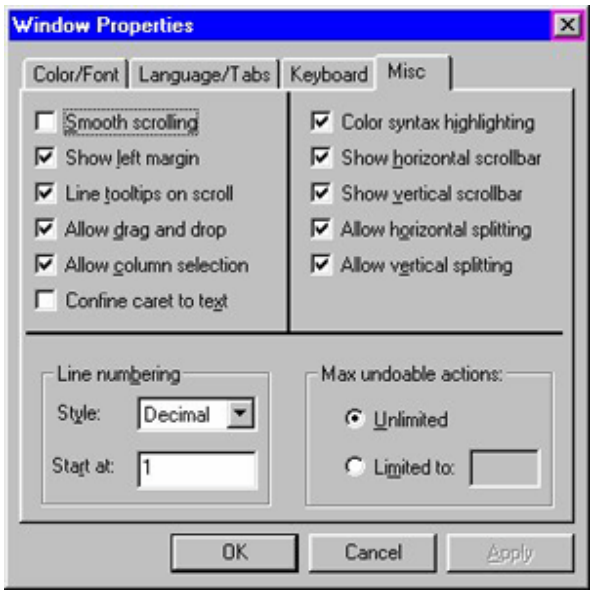
The Editor is the main part of the IDE. This is where your program appears under your fingers! Here is where you spend most of your development time! So the editor should be something very useful and friendly.

Some features and benefits:

- very fast syntax highlighting
- line numbers can be in decimal, hex or binary format
- bookmarks, Ctrl-F2 for mark, F2 to switch between bookmarks
- horizontal and/or vertical split bars of same file (drag from left-down and/or upper-right scroll bars),



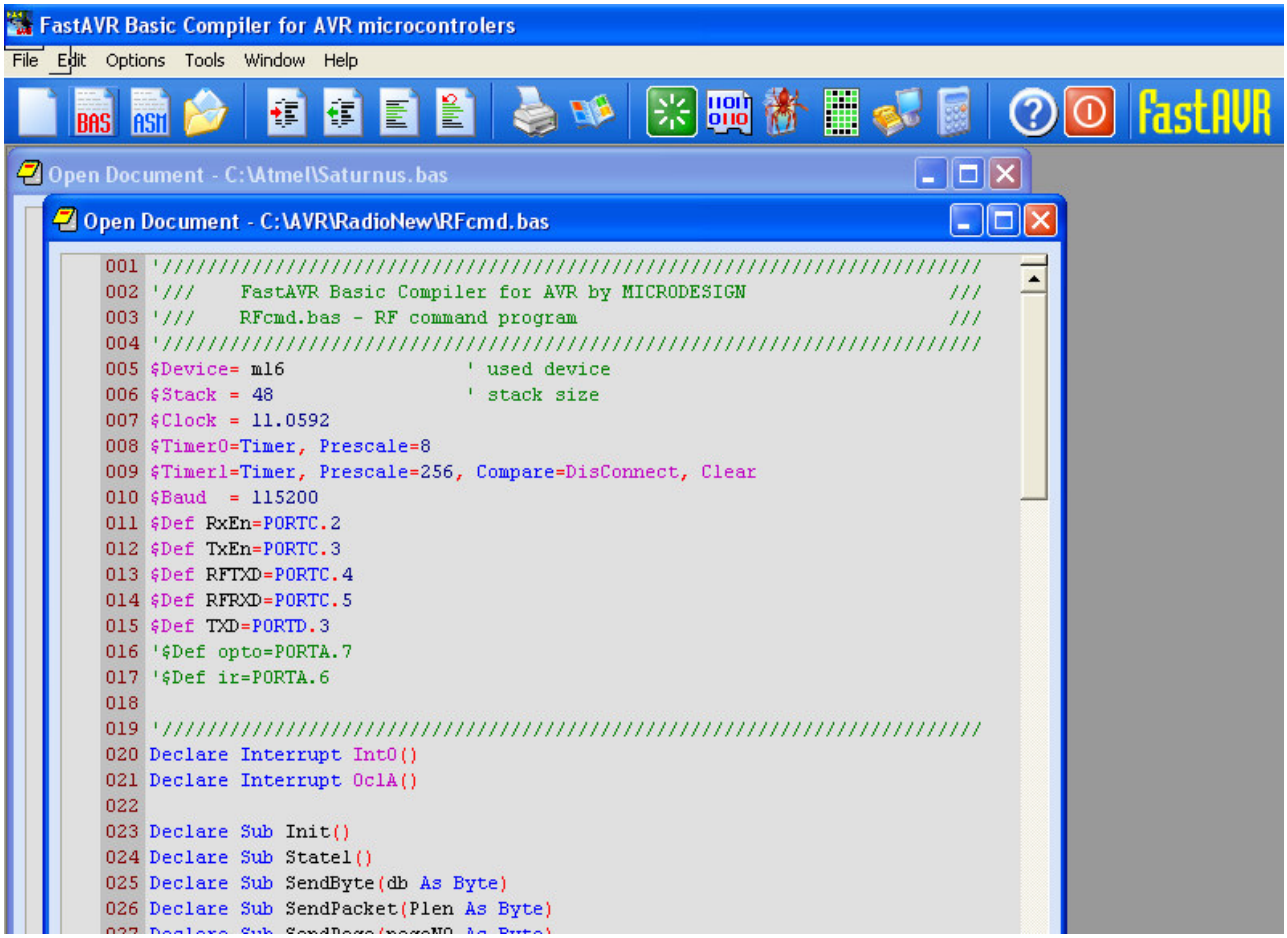
- editor properties window with right-click on editor screen:



- fully configurable keyboard commands
- double click on word to select and enable Find or Replace
- Find and Replace commands inside right-click on editor screen
- automatic reload of last edited or compiled file
- and many more...

3.2. IDE

Integrated Development Environment is your working desktop!
With easy-to-use menus, files and windows can be easily manipulated.
Everything needed during the development process can be found in the ToolBar.
Buttons are self explanatory and very easy to use.



The main screen is used for editing files. More than one file can be open at once.
At the bottom is the Compiler status frame where compiled results can be viewed!

3.3. Keyboard Commands

Command	Keystroke
Compile	F5
Run LCD char gen	F4
Run Programmer	F6
Run Terminal Emulator	F9
Run AVR Calculator	F11
BookmarkNext	F2
BookmarkPrev	Shift + F2
BookmarkToggle	Control + F2
CharLeft	Left
CharLeftExtend	Shift + Left
CharRight	Right
CharRightExtend	Shift + Right
Copy	Control + C
Copy	Control + Insert
Cut	Shift + Delete
Cut	Control + X
CutSelection	Control + Alt + W
Delete	Delete
DeleteBack	Backspace
DocumentEnd	Control + End
DocumentEndExtend	Control + Shift + End
DocumentStart	Control + Home
DocumentStartExtend	Control + Shift + Home
Find	Alt + F3
Find	Control + F
FindNext	F3
FindNextWord	Control + F3
FindPrev	Shift + F3
FindPrevWord	Control + Shift + F3
FindReplace	Control + Alt + F3
GoToLine	Control + G
GoToMatchBrace	Control +]
Home	Home
HomeExtend	Shift + Home
IndentSelection	Tab
LineCut	Control + Y
LineDown	Down
LineDowNextend	Shift + Down
LineEnd	End
LineEndExtend	Shift + End
LineOpenAbove	Control + Shift + N
LineUp	Up
LineUpExtend	Shift + Up
LowercaseSelection	Control + U
PageDown	Next
PageDowNextend	Shift + Next
PageUp	PRIOR
PageUpExtend	Shift + Prior
Paste	Control + V
Paste	Shift + Insert
Properties	Alt + Enter
RecordMacro	Control + Shift + R
Redo	F8
SelectLine	Control + Alt + F8
SelectSwapAnchor	Control + Shift + X
SentenceCut	Control + Alt + K
SentenceLeft	Control + Alt + Left

SentenceRight	Control + Alt + Right
SetRepeatCount	Control + R
TabifySelection	Control + Shift + T
ToggleOvertype	Insert
ToggleWhitespaceDisp	Control + Alt + T
Undo	Control + Z
Undo	Alt + Backspace
UnindentSelection	Shift + Tab
UntabifySelection	Control + Shift + Space
UppercaseSelection	Control + Shift + U
WindowScrollDown	Control + Up
WindowScrollLeft	Control + PageUp
WindowScrollRight	Control + PageDown
WindowScrollUp	Control + Down
WordDeleteToEnd	Control + Delete
WordDeleteToStart	Control + Backspace
WordLeft	Control + Left
WordLeftExtend	Control + Shift + Left
WordRight	Control + Right
WordRightExtend	Control + Shift + Right

3.4. Mouse Use

Mouse Action:	Result:
L-Button click over text	Changes the caret position
R-Button click	Displays the edit menu
L-Button down over selection, and drag	Moves text
Ctrl + L-Button down over selection, and drag	Copies text
L-Button click over left margin	Selects line
L-Button click over left margin, and drag	Selects multiple lines
Alt + L-Button down, and drag	Select columns of text
L-Button double click over text	Select word under cursor
Spin IntelliMouse mouse wheel	Scroll the window vertically
Single click IntelliMouse mouse wheel	Select the word under the cursor
Double click IntelliMouse mouse wheel	Select the line under the cursor
Click and drag splitter bar	Split the window into multiple views
Double click splitter bar	Split the window in half into multiple views

4. FastAVR Tools

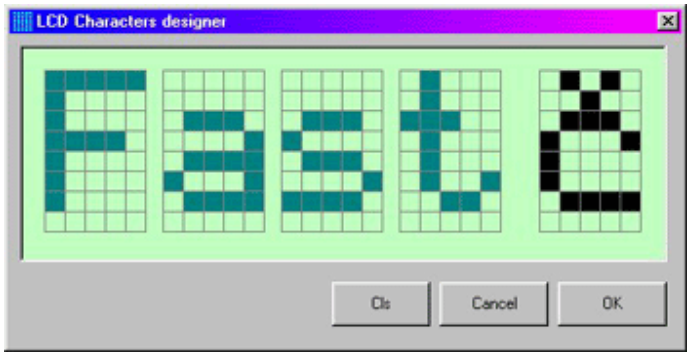
4.1. AVR Studio

You can Debug or Simulate your program at assembler level using Atmel's free AVR Studio.
For this purpose please load Obj file to AVR Studio!

When pressing the **DEBUG** button from the main toolbar for the first time you will be asked to locate the **AVR Studio** software!
Any further click of the Debug button will run AVR Studio!
AVRStudio3 can be downloaded for simulating and/or debugging the assembler output file!

4.2. LCD Character Generator

The alphanumeric LCD can define up to eight special characters numbered from 0 to 7.



First design your character by clicking on LCD pixel blocks (left click- set pixel, right click- reset pixel). By pressing OK, the LCD designer will insert a special code at the current cursor position in the active document window.

```
DefLcdChar 0, &h0A, &h04, &h0E, &h11, &h10, &h10, &h0F, &h00
```

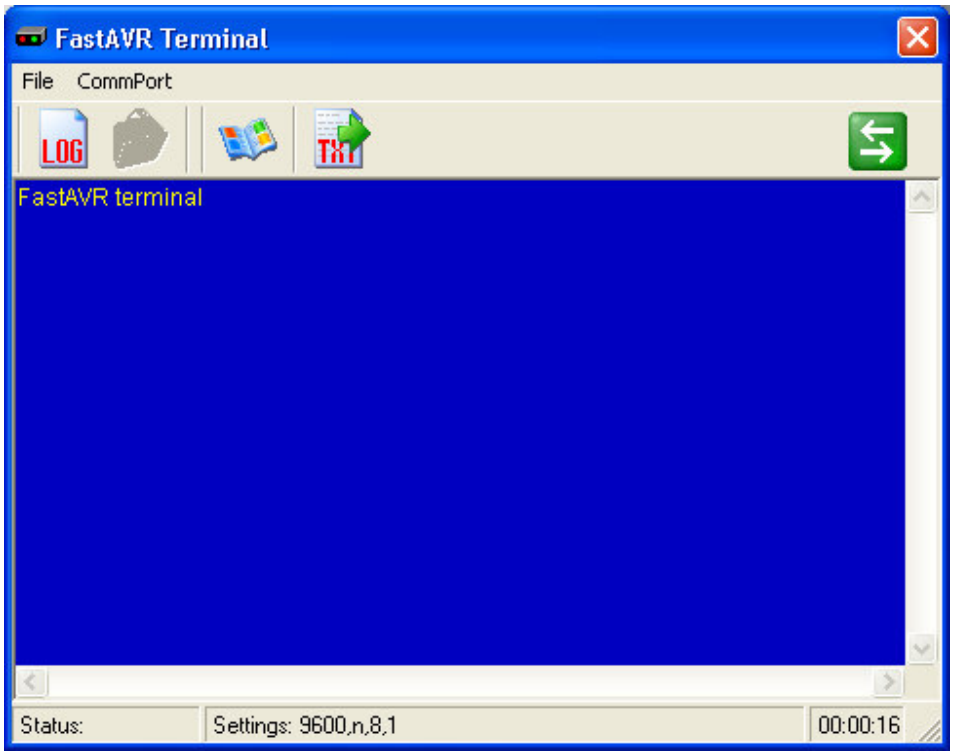
Zero after DefLcdChar is the Character number and must be edited in subsequent character definitions!

The new LCD character can be displayed on the LCD using the statement:

```
Lcd Chr(n)       'where n is the character number from 0 to 7
```

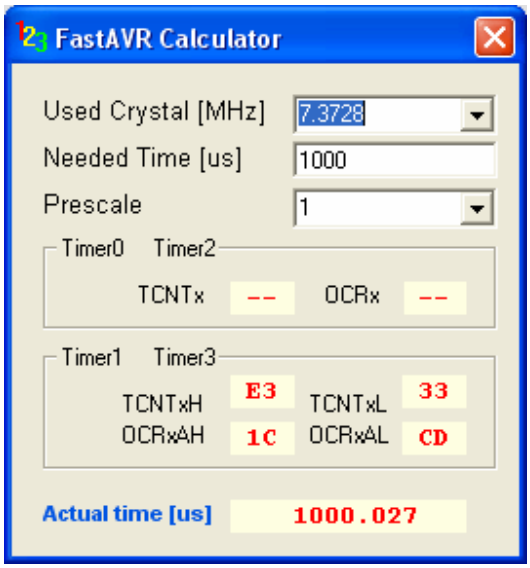
4.3. Terminal Emulator

When testing out the UART (hardware or software type), you may wish to monitor the output from your hardware. Terminal emulator will capture any ASCII output sent using the Print statement.
While typing in Terminal Emulator, all characters are sent to your hardware and can be captured using Input. ComPort must first be configured for the correct Port (Com1, Com2), speed (9600,....) and other parameters! The Terminal Emulator port must be opened by clicking on the RED circle!



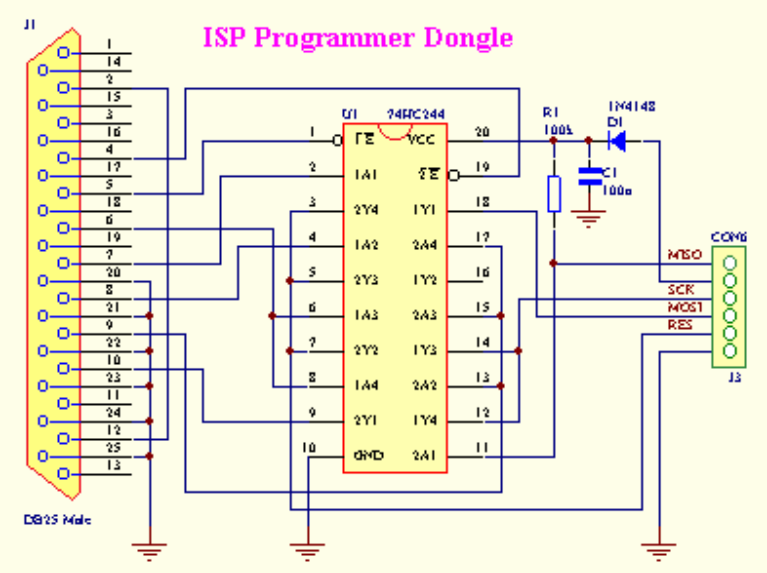
4.4. AVR Calculator

AVR calculator allows quick calculations for timer reload values based on the crystal used, needed time and prescale factor!
Calculated results are for Timer Overflow and for OutputCompare!



4.5. Programmer

FastAVR runs Atmel's free ISP programming software installed on your PC (or any other programming software). Programming can be accomplished using a very simple programming dongle connected to your Parallel port. Here is the schematic to build one:



When pressing the **PROGRAM** button from the main tool bar the first time you will be asked to locate Your prefered programming software!
Any further click on the Program button will run the ISP programmer!

In addition, You can enter special Command line parameters if You are using such a Programmer!

You can download ISP Programmer from Atmels www !

5. AVR fundaments

The best reading about AVR core is AVR data documents at
http://www.atmel.com/dyn/products/datasheets.asp?family_id=607.

6. FastAVR KeyWords

6.1. Meta - Statements

Meta-statements direct the compiler. Most are to configure compiler options. Some cue the compiler about the intended program actions, such as interrupt handling. Thus, some directives cause code from predefined libraries to be included.

Meta-statement keywords begin with "$".

6.1.1. Compiler directives

6.1.1.1. $Angles

Description:
Defines how Angles will be threated in Trigonometric functions.

Syntax:
```
$Angles = Degrees|Radians
```

Remarks:
Default is Radians.

Example:
```
$Angles=Degrees     'Angles are in Deegrres

Dim f1 As Float

f1=Sin(30)     'f1=0.5000000
```

Related topics:
Sin
Cos
Tan
Asin
Acos
Atan

6.1.1.2. $Asm

Description:
Starts an assembler program subroutine.

Syntax:
```
$Asm
```

Remarks:
This allows to use inline assembly code.
Always use $Asm with $EndAsm at the end of a block.

Example:
```
$Asm
  ldi     zl,0x65
  st      c,zl
$EndAsm
```

6.1.1.3. $Include

Description:
Instructs the compiler to include a Basic source file from disk at that position.

Syntax:
```
$Include "Path\BasDoc.bas"
```

Remarks:
The compiler continues with the next statement in the original source file when it encounters the end of the included file. The result is the same as if the contents of the included file were physically present in the original source file.

Example:
```
$Include "C:\FastAVR\Init.bas"
$Include "C:\FastAVR\Font.bas"
```

6.1.1.4. $IncludeAsm

Description:
Instructs the compiler to include a ASM source file from disk at the position.

Syntax:
```
$IncludeAsm "Path\Utils.asm"
```

Remarks:
The compiler just add included file in to generated ASM output, so only Assembler will compile it.

Example:
```
$Include "C:\FastAVR\Init.asm"
```

6.1.1.5. $Source

Description:
Tells the compiler to add Basic statements as comments in the ASM file for easy debugging.

Syntax:
```
$Source=ON|OFF
```

Could be omitted, default is ON.

6.1.2. Processor Configuration

6.1.2.1. $Baud

Description:
Defines the UART (or second UART) baud rate and optional setings. It is similar to bits per second, but includes other non-data bits (start, stop, mark) which add overhead.

Syntax:
```
$Baud = const [, Parity, DataBits, StopBits]
$Baud2 = const [, Parity, DataBits, StopBits]    ' for second UART
```

INPORTANT! If user will use UART in Default mode (No Parity, 8 data bits, 1 Stop bit) use short mode:
```
$Baud = 9600
```

Specifying Parity, ..., will add extra routines for handling this extra features!

Remarks:
const is the baud rate number with standard values:
1200, 2400, 4800, 9600, 19200, 38400, 56600,76800,115200
Higher Baud rates are possible on new Mega devices!

Parity N, O, E, M or S
DataBits 5, 6, 7, 8 or 9
StopBits 1 or 2 (in case of 9 DataBits, must be only 1 StopBit)

 See the AVR datasheets for valid UART settings for the target microprocessor

Example:
```
$Baud = 9600
$Baud2 = 9600
```

Related topics:
Baud
$Clock

6.1.2.2. $Clock

Description:
Defines for the compiler's use the frequency of the microprocessor's crystal input. This is used to calculate compiler-generated constants in the output of a compilation. These constants are used at run time to set the serial port(s) baud rates and for built-in functions which delay by looping.

Syntax:
```
$Clock=const
```

Remarks:
const is the frequency value of crystal used. (In MHz)

Check for max working frequency for specific microcontroler!


If the value given in the $Clock meta-statement is not the actually implemented microprocessor frequency, the serial port baud rate will be in error. The baud rate must be within a few percent in order for successful and reliable communications over time and temperature variations. Also, the software delay loops and hardware timers will be in error.

Example:
```
$Clock = 3.6864   'Our crystal is 3.6864MHz
```

Related topics:
$Baud
Baud

6.1.2.3. $Device

Description:
Declares which microprocessor product is the target for the generated code.

Syntax:
```
$Device=type [, Xram, FirstAdr, XramLength]
```

Remarks:
type refers to a particular microprocessor product.
The other parameters are optional and define external RAM if present, for those chips which support such.

Example:
```
$Device= 4433

$Device= 8515, Xram, 0, 32k

$Device= ATmega16
$Device= mega16
$Device= m16

$Device= tiny13
```

6.1.2.4. $Stack

Description:
Defines the stack size needed for the program based on the program's design, the needs of the run-time libraries created by the compiler, and the interrupt arrangements. The stack space is created by initialization code generated by the compiler.

The stack must include space for the following:

- The worst-case nesting of calls to SUBs and FUNCTIONS (return addresses)
- The worst-case nesting of parameters passed to FUNCTIONS
  - Passed parameters are stored on stack frames, based on their sizes
- The worst-case nesting of the above with their local variables
  - That is, the DIMs inside SUB or FUNCTION or INTERRUPT
- Plus space for interrupt procedures and saving of registers at interrupt


The necessary stack size is often underestimated by the programmer. This error causes all kinds and sorts of non-obvious run-time program failures under statistical probabilities of events. For microprocessors with small memories, estimating optimal stack size is a true challenge for the software engineer.

Syntax:
```
$Stack=num
```

Remarks:
num is the number of memory bytes reserved for stack space.

Example:
```
$Stack = 32  'stack will be 32 bytes deep
```

6.1.3. I/O Configuration

6.1.3.1. $Def

Description:
Gives a symbolic name to a particular AVR I/O port and a bit position within that port. This enables user-defined names based on the purpose of the I/O pins of the microprocessor in any given application.

Syntax:
```
$Def name=PORT.n
```

Remarks:
name is a valid symbol name for the compiler.
PORT is one of the predefined names of the microprocessor I/O port such as PORTA or PORTB
n one digit between 0 and 7 depicting the bit number in the port

Example:
```
$Def Led=PORT.1

.
.
Set Led
```

6.1.3.2. $1Wire

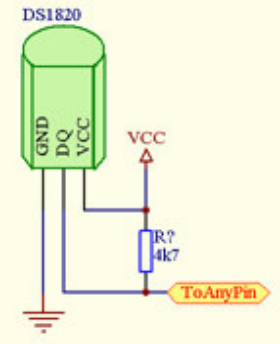
Description:
Tells the compiler which PORT.pin the 1wire bus is connected to.

Syntax:
```
$1Wire=Port.pin [, PORT.pin1, PORT.pin2, ...]
```

Remarks:
`PORT.pin` is the name of the physical pin.

You can have more than one 1Wire bus. Each additional Port.pin has its own index, first is 0!
Of course, You can connect also more 1Wire devices on each bus.

Example:
```
$1Wire=PORTD.2     '1Wire bus is connected to PortD.2
```



Related topics:
1wreset
1wread
1wwrite

6.1.3.3. $DTMF

Description:
Reminds wher is OC1 output for DTMF signal generation.

Syntax:
```
$DTMF = PORT.pin, duration
```

Remarks:
`Port` must be Port where OC1 pin is located.
`pin` is a OC1 pin number.
`duration` in ms, max is 255!

Example:
DTMF

6.1.3.4. $I2C

Description:
Defines the I2C bus pins connections for software single master configuration.

Hardware supported I2C on new Mega devices is not yet supported!

Syntax:
```
$I2C SDA=PORT.pin, SCL=PORT.pin
```

Remarks:
Tells the compiler which port pins SDA and SCL are connected to.

Dont forget pulup resistors on SDA and SCL (4k7 - 10k)!

Example:
```
$I2C SDA=PORTD.5, SCL=PORTD.6     'Defines I2C port pins
```

Related topics:
I2CStart
I2CWrite
I2CRead
I2CStop

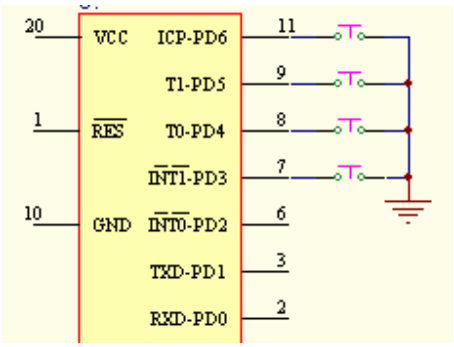
6.1.3.5. $Key

Description:
Defines the user defined keyboard matrix.

Syntax:
```
$Key Rows=PORT &hHexNum [, Cols=PORT &hHexNum] [, Hi] [, deb]
```

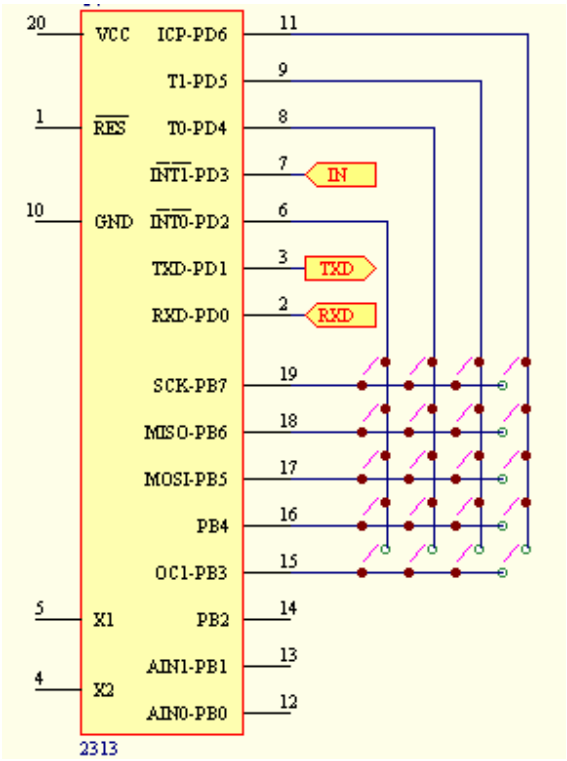
Remarks:
AVRPort is the name of the physical port.
&hHexNum is a two digit hex number representing keyboard wires
Hi this makes keys Active High (not for matrix organized keys)
deb is the debounce time in mseconds. Default is 20ms.

Example1:
```
'Defines four keys keyboard (keys in line)
$Key Rows=PORTD &h78
```



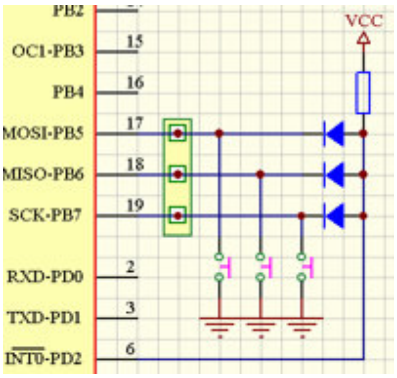
Example2:

```
'Defines matrix 5x4 Kbd connection (keys in matrix)
'debounce time is set to 50ms
$Key Rows=PORTB &hf8, Cols=PORTD &h74, 50
```



Example3:

```
'Defines three keys keyboard (keys in line), every keypress generates also External interrupt
INT0
$Key Rows=PORTB &he0
```



Related topics:

Key()

6.1.3.6. $LeadChar

Description:
Defines Leading Char for Print, Lcd, Tlcd, Glcd and Str(). All this outputs becomes RIGHT justified! Optionaly defines also Format.

Syntax:
```
$LeadChar="single string constant" [, Format(Int,Frac)]
```

Remarks:
"single string constant" becomes Leading Char
Int Number of Integer numbers or Scientific
Frac Number of Fractal numbers

Note: LeadChar could be one for the whole project! Statement Format reserves two bytes of SRAM as system variables!

For Format statements read Format.

Example:
```
n=5
$LeadChar="0", Format(3,1)   'lead char is "0", always will be three places left of decimal
point and one on the right
Lcd n                        'output: 000.5

n=54321
Format(3,1)                  'lead char is "0", always will be three places left of decimal
point and one on the right
Lcd n                        'output: 432.1

n=5
Format(2,0)                  'lead char is "0", always will be two places and no decimal point
Lcd n                        'output: 05
```

Related topics:
Format

6.1.3.7. $Lcd

Description:
Tells the compiler which pins the alphanumeric LCD is connected to.

Syntax:
For 4bit port connection:
```
$Lcd=PORT.pin, rs=PORT.pin, en=PORT.pin, cols, rows
```

RW pin MUST be wired to GND!

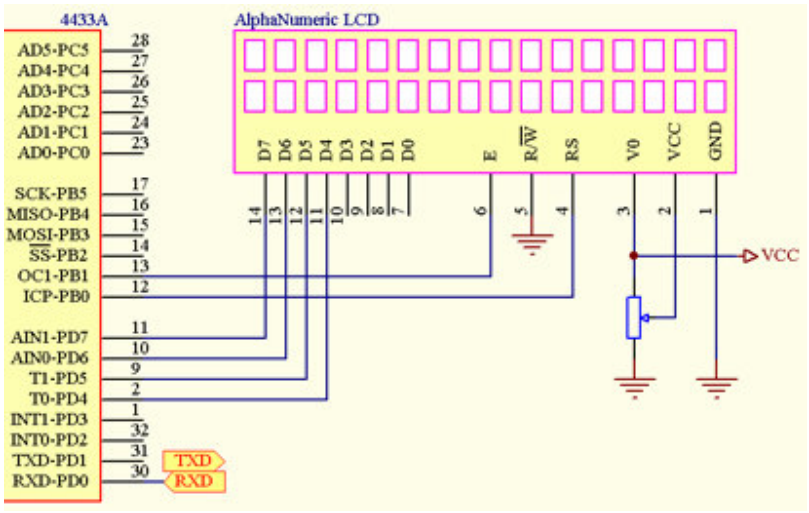
For 8bit BUS connection:
```
$Lcd=Adr, rs=AdrRS, cols, rows
```

ATTENTION! Configuration for STK-200 and STK-300 in bus mode:
```
$Lcd=&h8000, rs=&hc000, cols, rows
```
A15 to generate EN, A14 for RS

Remarks:
Port is the name of the physical port. Any bidirectional or output port can be used!
pin is the name of the physical pin at which Lcd's **D4** starts.
Adr is the Hex Address of the LCD connected in BUS mode.
AdrRS is the Hex Address of the LCD RS signal connected in BUS mode.
cols are the number of columns of the LCD.
rows are the number of rows of the LCD.

ATTENTION! The shortest code wil generate `$Lcd=PORTx.0` or `PORTx.4`, upper or lower nible!
Lcd pin RW MUST be soldered to GND!

Example:
```
$Lcd=PORTD.4, rs=PORTB.0, en=PORTB.1, 16, 2    'LCD Defined as 16x2
```



Related topics:
LCD
Locate
Display
Cursor
InitLcd

6.1.3.8. $PcKey

Description:
Configures AT Keyboard connection

Syntax:
```
$PcKey data=PORT.pin1, clock=PORT.pin2 [, NoInit]
```

Remarks:
data line for PcKey is connected to AVRport.pin1. Any bidirectional PORT.pin can be used!
`NoInit` option for skip Initialize PcKeyboard (usefull if BarCode reader is attached instead of PcKeyboard).

clock line for PcKey is connected to AVRport.pin2

Example:
PcKey()

Related topics:
PcKeySend()

6.1.3.9. $RC5

Description:
Configures Phillips RC5 IR receiving.

Syntax:
```
$RC5 = PORT.pin
```

Remarks:
Port is the name of the physical port.
pin is a pin number where IR receiver is connected.

Example:
RC5

Related topics:
RC5

6.1.3.10. $ShiftOut

Description:
Tells the compiler the name of the AVR pin for ShiftOut or ShiftIn

Syntax:
```
$Shiftout Data=PORT.pin, Clock=PORT.pin, Msb [Lsb]
```

Remarks:
Port is the name of the physical port.
Msb for Most significant bit first, **Lsb** for Less significant bit first

Routines works for both Clock polarityes!

Very usefull for expanding IO lines using shift registers!

Example:
```
$Shiftout Data=PORTB.0, Clock=PORTB.1, 1
```

Related topics:
ShiftOut
ShiftIn

6.1.3.11. $Sound

Description:
Configures AVR pin for Sound output.

Syntax:
```
$Sound = PORT.pin
```

Remarks:
AVRPort is the name of the physical port.
pin is a pin number where (buffered speaker or buzzer) is connected.

Example:
```
$Sound = PORTD.2    ' this pin for sound output
```

Related topics:
Sound

6.1.3.12. $Spi

Description:
Defines the SPI bus parameters.

Syntax:
```
$Spi num, Lsb|Msb, Master|Slave, Hi|Low, Hi|Low
```

Remarks:
num is the Clock division number for setting speed: 4, 16, 64, 128
Lsb or Msb tells which bit will be shifted out first.
First Hi or Low for Clock polarity (see Atmel's data)
Second Hi or Low for Clock Phase (see Atmel's data)

Note: During SPI initialization as Master Clk and MOSI are set to be output. SS must be held high!

Example:
```
$Spi 128, Lsb, Master, Hi, Low
```

Related topics:
SPIIn
SPIOut

6.1.3.13. $Timer

Description:
Defines the mode for Timers.

```
$Timer0=Timer [, Prescale=const, Async, Compare= Set|Reset|Toggle|DisConnected,
Clear]
$Timer0=Counter, Rising|Falling [, Compare= Set|Reset|Toggle|DisConnected, Clear]
$Timer0=PWM, 8|9|10, Normal|Inverted|DisConnected [,Prescale=8]

$Timer1=Timer [, Prescale=const,
```

```
Compare=Set|Reset|Toggle|DisConnected,CompareA=Set|Reset|Toggle|DisConnected,
CompareB=Set|Reset|Toggle|DisConnected, Clear, Capture=Rising|Falling, NoiseCancel]

$Timer1=Counter, Rising|Falling [,Compare=Set|Reset|Toggle|DisConnected,
CompareA=Set|Reset|Toggle|DisConnected, CompareB=Set|Reset|Toggle|DisConnected,
Clear, Capture =Rising|Falling, NoiseCancel]
$Timer1=PWM, 8|9|10, PwmA|PwmB=Normal|Inverted|DisConnected [, Prescale=8]

$Timer2=Timer [, Prescale=8, Async, Compare=Set|Reset|Toggle|DisConnected, Clear]
$Timer2=Counter, Rising|Falling [, Compare= Set|Reset|Toggle|DisConnected, Clear]
$Timer2=PWM, 8|9|10, Normal|Inverted|DisConnected [, Prescale=8]

$Timer3=Timer [, Prescale=const,
Compare=Set|Reset|Toggle|DisConnected,CompareA=Set|Reset|Toggle|DisConnected,
CompareB=Set|Reset|Toggle|DisConnected, Clear, Capture=Rising|Falling, NoiseCancel]

$Timer3=Counter, Rising|Falling [,Compare=Set|Reset|Toggle|DisConnected,
CompareA=Set|Reset|Toggle|DisConnected, CompareB=Set|Reset|Toggle|DisConnected,
Clear, Capture =Rising|Falling, NoiseCancel]
$Timer3=PWM, 8|9|10, PwmA|PwmB=Normal|Inverted|DisConnected [, Prescale=8]
```

Remarks:
const can be 1, 8, 64, 256, 1024, for Timer0 and Timer2 also 32 and 128 (not for all devices!)
Normal Timers are clocked with Non prescaled Clock in PWM and Compare modes. If the user wishes to use lower frequencies just combine statements, such as:

```
$Timer0=Timer, Prescale=256      ' Clock will be divided by 256
$Timer0=PWM, 8, Normal|Inverted    ' PWM will now use prescaled clock
```

In PWM mode, Use special variables: `Pwm0, Pwm1A, Pwm1B, Pwm2`.
In OutCompare mode, Use special variables: `Compare0, Compare1A, Compare1B, Compare2`.

See the manual for Timer usage!

Example:
```
$Timer0=Timer, Prescale=1
$Timer1=PWM, 8, PwmA=Inverted
```

6.1.3.14. $WatchDog

Description:
Defines the WatchDog time constant.

Syntax:
```
$WatchDog=const
```

Remarks:
const is the aprox. time in ms (at Vcc=5V) after which will WatchDog mechanism reset AVR device (16, 32, 64, 128, 256, 512, 1024, 2048).

Example:
```
$WatchDog = 2048   'About two seconds
```

Related topics:
Start
Stop
Reset

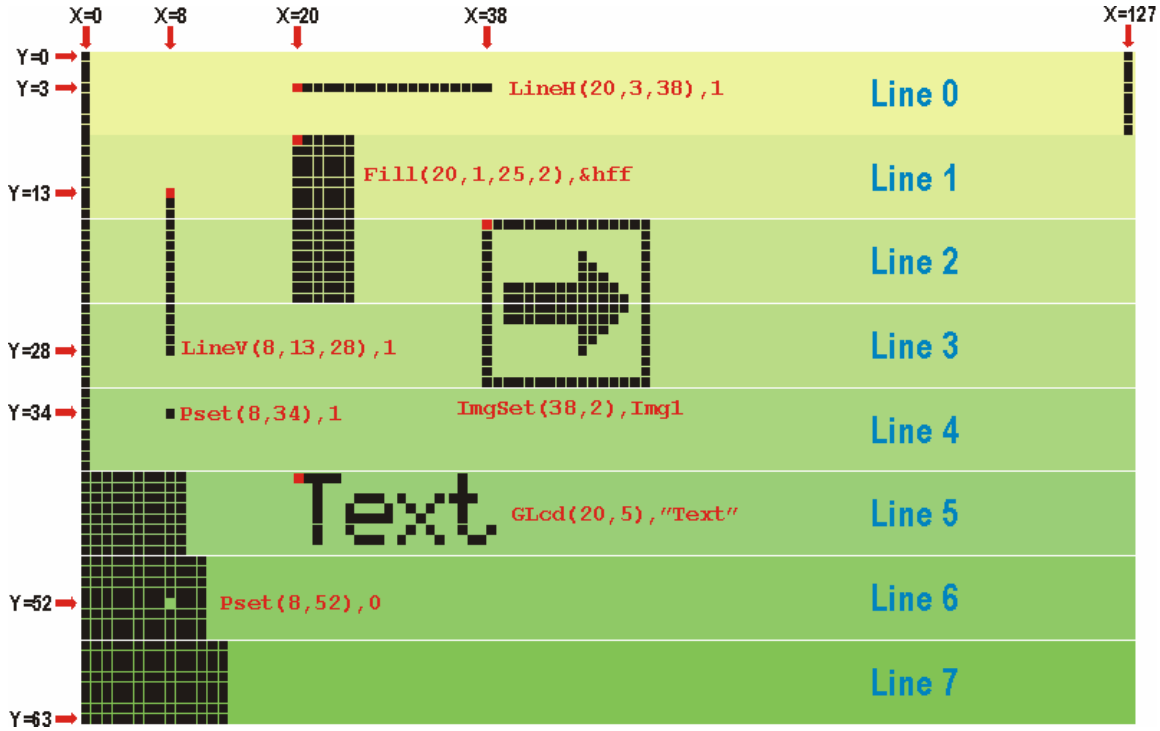
6.2. HD61202, KS0108B and SEP1520 Graphic LCD support

6.2.1. General

Graphic LCD (Hitachi HD61202 and Samsung KS1080B) usage.

Most commonly used graphic LCD has 128 x 64 pixels and it is produced by many manufacturers like Seiko (G1216), Hantronix (HDM64GS12), Samsung, WM-G1206,….

Pages are organized in rows (Lines), each being 8 pixels high. The number of Lines depends on the resolution of the particular display. For example, a 128 x 64 lcd would have 8 Lines, while a 128 x 32 lcd would only have 4 Lines. Some statements are Line oriented, not pixel. For instance, text can be written only on Lines, not in between.



Most displays using the HD61202 and KS0108B chipset are separated into two banks. Each bank is addressed by the use of two chip select lines (CS1 and CS2). Therefore, a 128 x 64 display would be treated like two (64 x 64) displays.
For more information on Lcd Graphic displays please refer to the datasheets.

Support for SEP1520 is done for popular 122 x 32 pix module format. Only $GCtrl is deeferente!

6.2.2. $GLCD, $GCtrl

Description:
Tells the compiler details about Graphic LCD connections.

Syntax:
```
$GLCD HD61202, Data=AVRPort, Ctrl=AVRPort, NumOfXpix, NumOfYpix, i
$GCtrl EN=4, WR=3, DI=2, CS1=0, CS2=1

For SEP1520:
$GCtrl A0=4, RW=3, E1=0, E2=1
```

Remarks:
HD61202 or KS0108B is the graphic controller chip used
Data AVRPort where data bus is connected
Ctrl AVRPort where control lines are connected
AVRPort any valid AVRPort. Any bidirectional port can be used!
NumOfXpix how many Pixels LCD has on X
NumOfYpix how many Pixels LCD has on Y
i type of init routine, could be **1, 2 or 3**
EN, WR, DI, CS1, CS2 valid Control line names for HD61202 and KS0108B
A0, RW, E1, E2 valid Control line names for SEP1520D61202

Note: Because of differences in Graphic Lcds, no provision is made for a hardware reset.
You may, however, assign any valid AvrPort pin that is available or use an appropriate RC setup for the LCD module reset. Please refer to the datasheet or manual for the specific graphic LCD module being used.

Control lines can be declared in any order!

Example:
```
$GLCD HD61202, Data=PORTB, Ctrl=PORTD, 128, 64, 1
$Gctrl EN=4, WR=3, DI=2, CS1=0, CS2=1

'EN is connected to PORTD.4, WR to PORTD.3...
```

6.2.3. Fill

Description:
Fills specified area with a byte pattern.

Syntax:
```
Fill(varX, varL, varX1, varL1), Pat
```

Remarks:
varX LeftMost X coordinate of area, normally between 0 and 126
varL TopMost Line of area, normally between 0 and 6
varX1 RightMost X coordinate of area, normally between 1 and 127
varL1 BottomMost Line of area, normally between 1 and 7
Pat Byte the area will be filled with

varX1 must be greater than **varX** and **varL1** must be greater than **varL.**

Y coordinates are in Lines not in Pixels! Also suitable for clearing a specific area.

Example:
```
Fill(15, 1, 60, 4), &haa    ' Specified area will be filled with &haa
```

Related Topics:
```
Inverse
GCls
```

6.2.4. FontSet

Description:
Selects soft Font.

Syntax:
```
FontSet NameOfFontTable
```

Remarks:
NameOfFontTable Table in Flash that contains individual letter definitions.

NameOfFontTable must be declared first and added into source ($Included)!
Fonts can be edited with the FastLCD utility and saved in **bas** format ready to include in source!
Selected Font is active until another Font is selected with FontSet.

Example:
```
Dim F0HD As Flash Byte
Dim F1HD As Flash Byte
Dim n As Byte
Dim s As String*20

n=15
s="Graphic LCD"
FontSet F1HD                        ' Selects F1
GLcd(15, 0), n                      ' Writes n with F1
GLcd(15, 7), s                      ' Writes w with F1

FontSet F0HD                        ' Selects F0
GLcd(15, 1), "HD61202"              ' Writes txt with F0

$Included "C:\FastAVR\F0HD.bas"     ' Here is 6x8 font definition
$Included "C:\FastAVR\F1HD.bas"     ' Here is 8x8 font definition
```

Related Topics:
```
GLcd
```

6.2.5. Gcls

Description:
Clears the Graphic LCD

Syntax:
```
GCls
```

Example:
```
GCls    ' Graphic LCD is now cleared
```

6.2.6. Glcd

Description:
Writes text on graphic LCD using previously specified soft Font.

Syntax:
```
GLcd(varX, varP), var
```

Remarks:
varX Starting X coordinate, normally between 0 and 127
varP Line to write in, between 0 and 7
var num, string, string constant or hex to write

Y coordinates are in Lines not in Pixels!
Font MUST be set (`FontSet`)prior to using `Glcd`!

If You wish to show just a few words, maybe ImgSet is better (shorter) solution (word is made as an Immage)!

If `$LeadChar` is defined then result will be right justified with Leading Chars as defined. Also, if `Format()` is defined then optional decimal point will be inserted!

Example:
```
GLcd(15, 0), n                  ' Writes num variable on upper Line
GLcd(15, 1), s                  ' Writes string var on second Line
GLcd(15, 2), "This is HD61202"  ' Writes string on third Line
GLcd(15, 3), Chr(61)            ' Writes letter A on 4. Line
GLcd(15, 4), Hex(61)            ' Writes Hex string on 5. Line
```

Related Topics:
```
FontSet
```

6.2.7. GlcdInit

Description:
Initializes the Graphic LCD display

Syntax:
```
GLcdInit
```

Example:
```
GLcdInit
```

Remarks:
$GLCD and $GCtrl must be setup prior to using GLcdInit.
At initial power on or anytime the graphic LCD is powered down, GLcdInit should be called to initialize the LCD before using any graphic statements.

Some LCDs has theirs own internal RESET, for others user MUST generate RESET (active LOW) before Calling GLcdInit!

6.2.8. GRead

Description:
Reads a byte from the graphic LCD at selected X and Line.

Syntax:
```
Var = GRead(varX, varL)
```

Remarks:
varX X coordinate, normally between 0 and 127
varL Line, between 0 and 7
var is assigned the value read

This is the graphic controllers native Read function.
Y coordinates are in Lines not in Pixels!

Example:
```
n = GRead(17, 2)    ' Data from x=17 on Line 2 will be Read into n.
```

Related Topics:
Gwrite

6.2.9. GWrite

Description:
Writes a byte at selected X and Line.

Syntax:
```
GWrite(varX, varL), var
```

Remarks:
varX X coordinate, normally between 0 and 127
varL Line, between 0 and 7
var to be written to desired position.

This is the graphic controllers native Write function.
Y coordinates are in Lines not in Pixels!

Example:
```
GWrite(17, 2), 15    ' Four pixels will be written to x=17 on the Line 2.
```

Related Topics:
GRead

6.2.10. ImgSet

Description:
Displays an Image or a part of ImageArray on the graphic LCD at selected X and Line.

Syntax:
```
ImgSet(varX, varP), NameOfImgTable
```

Or, if You wat to display just a part of an ImageArray:
(Image must be saved as ImageArray, when edited using FastLCD utility!)

```
ImgSet(varX, varP, var), NameOfImgTable
```

Remarks:
varX X coordinate, normally between 0 and 127
varL Line, between 0 and 7
var which part of Image, (index in ImageArray)
NameOfImgTable Table in Flash that contains the bit image.

Y coordinates are in Lines not in Pixels!
NameOfImgTable must be declared first and added into source ($Included)!
Images can be edited with FastLCD image editor which can save Images in **bas** format.
The saved image is then ready to be included in the source program!

Example:
```
Dim Img0 As Flash Byte
Dim Img1 As Flash Byte

ImgSet(15, 2), Img1    ' Image Img1 will be copied to location

$Included "C:\FastAVR\Img0.bas"    ' Img0 bit image definition
$Included "C:\FastAVR\Img1.bas"    ' Img1 bit image definition
```

Second syntax:
Using ImageArray, a large letters, Icons or Sprites can be displayed, all saved in a single Image!
Any part of this Image is accessable by its index, yeaa - this means animations!

Example:
```
Dim Sclk1616HD As Flash Byte
ImgSet(15, 2, 1), Sclk1616HD    ' SandClock with index 1
$Included "C:\FastAVR\Sclk1616HD.bas"    ' SandClock definition
```

Related Topics:
GLcd

6.2.11. Inverse

Description:
Inverses specified area on the screen.

Syntax:
```
Inverse(varX, varL, varX1, varL1)
```

Remarks:
varX LeftMost X coordinate of area, normally between 0 and 126
varL TopMost Line of area, normally between 0 and 7
varX1 RightMost X coordinate of area, normally between 1 and 127
varL1 BottomMost Line of area, normally between 0 and 7

varX1 must be greater than **varX** and **varL1** must be greater than **varL.**

Y coordinates are in Lines not in Pixels!

Example:
```
Inverse(15, 1, 60, 4)    ' Specified area will be Inversed
```

Related Topics:
Fill

6.2.12. LineH

Description:
Draws or Clears a Horizontal Line.

Syntax:
```
LineH(varX, varY, varX1), 0|1
```

Remarks:
varX X coordinate of LeftMost pixel in Line, normally between 0 and 126
varY Y coordinate of Line, normally between 0 and 63
varX1 X coordinate of RightMost pixel in Line, normally between 1 and 127
0|1 0 will Clear Line, 1 will Draw Line

varX1 must be greater than **varX.**

Example:
```
LineH(15, 20, 120), 1    ' Line will be Drawn from X=15 to 120, at y=20
```

Related Topics:
LineV

6.2.13. LineV

Description:
Draws or Clears a Vertical Line.

Syntax:
```
LineV(varX, varY, varY1), 0|1
```

Remarks:
varX X coordinate of Line, normally between 0 and 127
varY Y coordinate of TopMost pixel in Line, normally between 0 and 62
varY1 Y coordinate of BottomMost pixel in Line, normally between 1 and 63
0|1 **0 will Clear Line, 1 will Draw Line**

varY1 must be greater than **varY.**

Example:
```
LineV(15, 20, 60), 1    ' Vertical Line will be Drawn from y=20 to 60, at x=15
```

Related Topics:
LineH

6.2.14. Point

Description:
Tests if specified Pixel location is Set or Reset.

Syntax:
```
Var = Point(varX, varY)
```

Remarks:
varX X coordinate, normally between 0 and 127
varY Y coordinate, between 0 and 63
var is assigned the result, 0 if pixel is Reset, 1 if Pixel is Set

Example:
```
n = Point(15, 2)    ' If n>0 that Pixel is Set
```

Related Topics:
PSet

6.2.15. Pset

Description:
Sets or Resets an individual Pixel at the desired position.

Syntax:

```
Pset(varX, varY), 0|1
```

Remarks:
varX X coordinate, normally between 0 and 127
varY Y coordinate, normally between 0 and 63
0|1 0 will Reset pixel, 1 will Set pixel, (color)

Example:
```
Pset(15, 20), 1    ' Pixel at coordinates 15, 20 will be Set
```

Related Topics:
Point
LineH
LineV

6.3. PCD8544 - NOKIA 3310

6.3.1. General

Nokia 3310 has very nice, not expensive Graphic display 84 x 48 pixels.

Here http://www.myplace.nu/mp3/nokialcd.htm You can find more data.

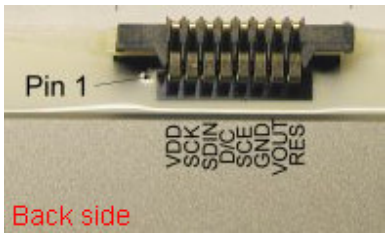
Pages are organized in rows (Lines), each being 8 pixels high. There are 6 Lines. Some statements are Line oriented, not pixel. For instance, text can be written only on Lines, not in between.

Display has serial interface using only four signals plus RESET which could be driven by RC.

Unfortunely, display has not possibility to Read from. This dont allow as to have standard graphic functions like Pset, Line and similar. To avoid this limitation a Buffer of 504 bytes could be reserved and all Drawings must be performed in this Buffer. This is for upgrading - if there will be enought interest!

Another possibility is to use SPI as serial interface - this is much faster transfer.

Display Coonection



Dont forget a 470nF capacitor beetween Vout and GND!

Look at **Nokia3310.bas**

6.3.2. $GLCD

Description:
Tells the compiler details about Graphic LCD connections.

Syntax:
```
$GLCD PCD8544, SDIN=PORTx.n, SCLK=PORTx.n, DC=PORTy.n, SCE=PORTy.n
```

Remarks:
PCD8544 or NOKIA3310 is the graphic controller chip used
SDIN, SCLK, DC, SCE Nokia display signals names.

SDIN and **SCLK** MUST be on the same PORT!
DC and **SCE** MUST be on the same PORT!

Control signals can be declared in any order!

You may also assign any valid AvrPort pin that is available for RESET Graphic module or use an appropriate RC setup for the LCD module reset like 10k PullUp resistor with 100nF capacitor to GND.

Example:
```
$GLCD NOKIA3310, SDIN=PORTA.0, SCLK=PORTA.1, DC=PORTA.2, SCE=PORTA.3
```

6.3.3. Contrast

Description:
Sets Nokia LCD module contrast.

Syntax:
```
Contrast=numeric expression
```

Remarks:
`numeric expression` Value from 0 to 127 to set contrast (defailt is 72).

Example:
```
For n=50 To 120
   Contrast=n     ' finding best contrast
   Wait 1
Next
```

6.3.4. FontSet

Description:
Selects soft Font.

Syntax:
```
FontSet NameOfFontTable
```

Remarks:
`NameOfFontTable` Table in Flash that contains individual letter definitions.

NameOfFontTable must be declared first and added into source ($Included)!
Fonts can be edited with the FastLCD utility and saved in **bas** format ready to include in source!
Selected Font is active until another Font is selected with FontSet.

Example:
```
Dim F0HD As Flash Byte
Dim F1HD As Flash Byte
Dim n As Byte
Dim s As String*20

n=15
s="Graphic LCD"
FontSet F1HD                    ' Selects F1
GLcd(15, 0), n                  ' Writes n with F1
GLcd(15, 7), s                  ' Writes w with F1

FontSet F0HD                    ' Selects F0
GLcd(15, 1), "HD61202"          ' Writes txt with F0

$Included "C:\FastAVR\F0HD.bas"   ' Here is 6x8 font definition
$Included "C:\FastAVR\F1HD.bas"   ' Here is 8x8 font definition
```

Related Topics:
`GLcd`

6.3.5. Glcd

Description:
Writes text on graphic LCD using previously specified soft Font.

Syntax:
```
GLcd(varX, varP), var
```

Remarks:
varX Starting X coordinate, normally between 0 and 83
varP Line to write in, between 0 and 5
var num, string, string constant or hex to write

Y coordinates are in Lines not in Pixels!
Font MUST be set (`FontSet`)prior to using `Glcd`!

If You wish to show just a few words, maybe ImgSet is better (shorter) solution (word is made as an Immage)!

If `$LeadChar` is defined then result will be right justified with Leading Chars as defined. Also, if `Format()` is defined then optional decimal point will be inserted!

Example:
```
GLcd(15, 0), n                    ' Writes num variable on upper Line
GLcd(15, 1), s                    ' Writes string var on second Line
GLcd(15, 2), "This is HD61202"    ' Writes string on third Line
GLcd(15, 3), Chr(61)              ' Writes letter A on 4. Line
GLcd(15, 4), Hex(61)              ' Writes Hex string on 5. Line
```

Related Topics:
FontSet

6.3.6. Gwrite

Description:
Writes a byte at selected X and Line.

Syntax:
```
GWrite(varX, varL), var
```

Remarks:
varX X coordinate, normally between 0 and 83
varL Line, between 0 and 5
var to be written to desired position.

This is the graphic controllers native Write function.
Y coordinates are in Lines not in Pixels!

Example:
```
GWrite(17, 2), 15    ' Four pixels will be written to x=17 on the Line 2.
```

6.3.7. ImgSet

Description:
Displays an Image or a part of ImageArray on the graphic LCD at selected X and Line.

Syntax:
```
ImgSet(varX, varP), NameOfImgTable
```

Or, if You wat to display just a part of an ImageArray:
(Image must be saved as ImageArray, when edited using FastLCD utility!)

```
ImgSet(varX, varP, var), NameOfImgTable
```

Remarks:
varX X coordinate, normally between 0 and 83
varL Line, between 0 and 5
var which part of Image, (index in ImageArray)
NameOfImgTable Table in Flash that contains the bit image.

Y coordinates are in Lines not in Pixels!

NameOfImgTable must be declared first and added into source ($Included)!
Images can be edited with FastLCD image editor which can save Images in **bas** format.
The saved image is then ready to be included in the source program!

Example:
```
Dim Img0 As Flash Byte
Dim Img1 As Flash Byte

ImgSet(15, 2), Img1    ' Image Img1 will be copied to location

$Included "C:\FastAVR\Img0.bas"    ' Img0 bit image definition
$Included "C:\FastAVR\Img1.bas"    ' Img1 bit image definition
```

Second syntax:
Using ImageArray, a large letters, Icons or Sprites can be displayed, all saved in a single Image!
Any part of this Image is accessable by its index, yeaa - this means animations!

Example:
```
Dim Sclk1616HD As Flash Byte
ImgSet(15, 2, 1), Sclk1616HD    ' SandClock with index 1, second sub-Image will be
displayed

$Included "C:\FastAVR\Sclk1616HD.bas"    ' SandClock definition
```

Related Topics:
GLcd

6.3.8. Inverse

Description:
Sets NORMAL or INVERSE screen.

Syntax:
```
Inverse(var)
```

Remarks:
var 0 - normal, 1 - inverse

Only whole display can be Inversed!

Example:
```
Inverse(1)    ' Screen is Inversed
```

6.3.9. Gcls

Description:
Clears the Graphic LCD

Syntax:
```
GCls
```

Example:
```
GCls    ' Graphic LCD is now cleared
```

6.4. T6963C Graphic LCD support

6.4.1. $GLCD, $GCtl

Description:
Tells the compiler details about Graphic LCD connections.

Syntax:
```
$GLCD T6963C, Data=AVRPort, Ctrl=AVRPort, NumOfXpix, NumOfYpix, i
$GCtrl WR=4, RD=3, CE=2, CD=1, FS=1
```

Remarks:
T6963C is the graphic controller chip used
Data AVRPort where data bus is connected.
Ctrl AVRPort where control lines are connected.
AVRPort any valid AVRPort. Any bidirectional port can be used!
NumOfXpix how many Pixels LCD has on X

NumOfYpix how many Pixels LCD has on Y
i **1** for single, **2** for double scan graphic LCD display module (low pix modules has single scan, larges double scan. Look datasheets for details)
WR, RD, CE, CD valid Control line names for T6963C
FS 1 for 6x8 and 0 for 8x8 fonts - tells how Bytes will be using (6 or 8 bits)

Note: Because of differences in Graphic Lcds, no provision is made for a hardware reset.
You may, however, assign any valid AvrPort pin that is available or use an appropriate RC setup for the LCD module reset. Please refer to the datasheet or manual for the specific graphic LCD module being used.

FontSelect (FS) MUST be fixed to 1 for 6x8 internal Font and 6 pix wide column or fixed to 1 for 8x8 internal Font and 8 pix wide column.

Control lines can be declared in any order!

Example:
```
$GLCD T6963C, Data=PORTB, Ctrl=PORTD, 128, 64, 1
$Gctrl WR=4, RD=3, CE=2, CD=1, FS=1

'WR is connected to PORTD.4, RD to PORTD.3...
```

6.4.2. Box

Description:
Draws or Clears a box.

Syntax:
```
Box(varX0, varY0, varX1, varY1), 0|1
```

Remarks:
varX0 X coordinate of Upper Left corner
varY0 Y coordinate of Upper Left corner
varX1 X coordinate of Lower Right corner
varY1 Y coordinate of Lower Right
0|1 0 will Clear Line, 1 will Draw Line

Example:
```
Circle(0, 0, 239, 127), 1    ' Rectangle around 240x128 pix LCD
```

Related Topics:
Pset
Line
Circle
LineH
LineV

6.4.3. Circle

Description:
Draws or Clears a Circle.

Syntax:
```
Circle(varX, varY, radius), 0|1
```

Remarks:
varX X coordinate of center
varY Y coordinate of center
radius of the circle
0|1 0 will Clear Line, 1 will Draw Line

Example:
```
Circle(120, 64, 60), 1    ' Circle on center 240x128 pix LCD, r=60
```

Related Topics:
Line
Box
LineH
LineV
Pset

6.4.4. Fill

Description:
Fills specified area on the screen.

Syntax:
```
Fill(varX, varY, varX1, varL1), Pat
```

Remarks:
varX LeftMost X coordinate of area, between 0 and NofColumns
varL TopMost coordinate, between 0 and Ymax
varX1 number of columns to Inverse
varL1 number of lines (pixels) to Inverse
Pat Byte the area will be filled with

X coordinates are in Columns not in Pixels! Also suitable for clearing a specific area.

Example:
```
Fill(15, 1, 6, 40), &haa    ' Specified area will be filled with &haa
```

Related Topics:
GCls
Inverse

6.4.5. FontSet

Description:
Selects soft Font.

Syntax:
```
FontSet NameOfFontTable
```

Remarks:
NameOfFontTable Table in Flash that contains individual letter definitions.

NameOfFontTable must be declared first and added into source (`$Included`)!
Fonts can be edited with the FastLCD utility and saved in **bas** format ready to include in source!
Selected Font is active until another Font is selected with `FontSet`.

Example:
```
Dim F0T As Flash Byte
Dim F1T As Flash Byte
Dim n As Byte
Dim s As String*20

n=15
s="Graphic LCD"
FontSet F1T                          ' Selects F1T
GLcd(15, 0), n                       ' Writes n with F1
GLcd(15, 7), s                       ' Writes w with F1

FontSet F0T                          ' Selects F0T
GLcd(15, 1), "T6963c"                ' Writes txt with F0

$Included "C:\FastAVR\F0T.bas"    ' Here is 6x8 font definition
$Included "C:\FastAVR\F1T.bas"    ' Here is 8x8 font definition
```

Related Topics:
GLcd
Tlcd

6.4.6. GCls

Description:
Clears the graphic area on Graphic LCD.

Syntax:
```
GCls
```

Example:
```
GCls    ' Graphic area is now cleared
```

6.4.7. GCommand

Description:
Directly controls T6963C graphic LCD controler. Used for several setings.

Syntax:
```
GCommand var
```

Remarks:
var appropriate value to set needed command according to tabel:

Graphic Mode Set:
&h80 Logycally OR of Text with Graphics
&h81 Logycally XOR of Text with Graphics
&h83 Logycally AND of Text with Graphics
&h84 Text only, with Attribute data in Graphic Area
&h90 Display OFF
&h92 Text Cursor ON, Blink OFF
&h93 Text Cursor ON, Blink ON
&h94 Text ON, Graphic OFF
&h98 Text OFF, Graphic ON
&h9a Text ON, Graphic ON

Text Cursor on Text Page Shape:
&ha0 1 line Cursor
&ha1 2 line Cursor
&ha2 3 line Cursor
&ha3 4 line Cursor
&ha4 5 line Cursor
&ha5 6 line Cursor
&ha6 7 line Cursor
&ha7 8 line Cursor

Text Page and Cursor must be turned ON and Cursor shape defined with GCommand!

Example:
```
GCommand &h80    ' OR with Txt and Grph
GCommand &h98    ' turn Txt OFF, Grp ON
```

6.4.8. GCursor

Description:
Sets the shape of Text curcor on Text Page.

Syntax:
```
GCursor varX, varY
```

Remarks:
varX X coordinate in Characters (Columns)
varY Y coordinate in Lines

Text Page and Cursor must be turned ON and Cursor shape defined with GCommand!

Example:
```
GCursor 5, 2    ' Cursor is positioned
```

Related Topics:
GCommand

6.4.9. General

User can find a lot of LCD graphic modules based on Toshiba's T6963C graphic controler. Dimensions runs from 128x64 pix to 256x128 pix.
Generaly they has Text page and Graphic page, both organized in 6 or 8 pix wide columns, depends on pin FS (Font Select). This pin MUST be fixed in the project! User can use Graphic, Text or both together. Text screen coordinates are in characters, while Graphic page coordinates are in pixels, starting in upper left corner (0,0). Some graphics X coordinates are in Columns, while Y are in pixels.
User can select from 6x8 and 8x8 font driving pin FS (hard wired). Unfortunately they are the same font with large interspace in case of 8x8. Writing on Text page is as with any standard alphanumeric LCD using it's own built-in character generator.

Useful links:
http://doc.semicon.toshiba.co.jp/noseek/us/td/03frame.htm
http://ourworld.compuserve.com/homepages/steve_lawther/t6963c.pdf
http://202.76.113.1/varitronix/htm/product.htm
http://www.hantronix.com/index.html

6.4.10. GLcd

Description:
Writes text on graphic LCD's Graphic page. This mode uses soft characters generator in Flash table. User can edit Fonts using FastLCD.
FS pin setting MUST be on 1 for 6 pix Column wide or 0 for 8 pix column.

Syntax:
```
GLcd(varC, varL), var
```

Remarks:
varC Starting X coordinate in columns
varL Pixel to write in
var num, string, string constant or hex to write

X coordinates are in Columns, Y in Pixels!
Font MUST be set (FontSet)prior to using Glcd!

If You wish to show just a few words, maybe ImgSet is better (shorter) solution (word is made as an Immage)!

If $LeadChar is defined then result will be right justified with Leading Chars as defined. Also, if Format() is defined then optional decimal point will be inserted!

Example:
```
GLcd(0, 0 ), n                    ' Writes num variable
GLcd(1, 10), s                    ' Writes string var
GLcd(2, 30), "This is T6963c"     ' Writes string
GLcd(3, 60), Chr(61)              ' Writes letter A
GLcd(4, 90), Hex(61)              ' Writes Hex string
```

Related Topics:
FS pin configuration
FontSet
TLcd

6.4.11. GlcdInit

Description:
Initializes the Graphic LCD display

Syntax:
```
GLcdInit
```

Example:
```
GLcdInit
```

Remarks:
`$GLCD` and `$GCtrl` must be setup prior to using `GLcdInit`.
At initial power on or anytime the graphic LCD is powered down, `GLcdInit` should be called to initialize the LCD before using any graphic statements.

Some LCDs has theirs own internal RESET, for others user MUST generate RESET (active LOW) before Calling `GLcdInit`!

6.4.12. GRead

Description:
Reads a byte from the graphic LCD at selected Column and Y on the Graphic page.

Syntax:
```
Var = GRead(varX, varL)
```

Remarks:
varC X coordinate in columns
varY Y coordinate
var is assigned the value read

This is the graphic controllers native Read function.
X coordinates are in Columns, not in Pixels!

Example:

```
n = GRead(17, 2)    ' Data from Column=17 on Y=2 will be Read into n.
```

Related Topics:
GWrite

6.4.13. GrpAreaSet

Description:
Sets the Graphic Page width in Columns.

Syntax:
```
GrpAreaSet col
```

Remarks:
col width of Graphic page

Graphic page can be wider then LCD visible area!

Example:
```
GrpAreaSet 20    ' Graphic page is 20 colums wide
```

Related Topics:
TxtAreaSet
TxtHomeSet
GrpHomeSet

6.4.14. GrpHomeSet

Description:
Sets the Graphic Page origin.

Syntax:
```
GrpHomeSet adr
```

Remarks:
adr address of Graphic page

Example:
```
GrpHomeSet &h0200    ' Graphic page is at &h0200 of T6963C RAM
```

Related Topics:
TxtAreaSet
TxtHomeSet
GrpAreaSet

6.4.15. GWrite

Description:
Writes a byte at selected Column and Y on the Graphic page.

Syntax:
```
GWrite(varX, varL), var
```

Remarks:
varX X coordinate, normally between 0 and 127
varL Line, between 0 and 7
var to be written to desired position.

This is the graphic controllers native Write function.
X coordinates are in Columns not in Pixels!

Example:
```
GWrite(17, 2), 15   ' Four pixels will be written to x=17 on the Line 2.
```

Related Topics:
GRead

6.4.16. ImgSet

Description:
Displays an Image or a part of ImageArray on the graphic LCD at selected Column and Y.
FS pin setting MUST be on 1 for 6 pix Column wide or 0 for 8 pix column.

Syntax:
```
ImgSet(varC, varY), NameOfImgTable
```

Or, if You wat to display a element of an ImageArray:
(Image must be saved as ImageArray, when edited using FastLCD utility!)

```
ImgSet(varC, varY, var), NameOfImgTable
```

Remarks:
varC coordinate in Column
varY Y coordinate (in Pix)
var which part of Image, (index in ImageArray)
NameOfImgTable Table in Flash that contains the bit image.

X coordinates are in Columnc not in Pixels!

NameOfImgTable must be declared first and added into source (`$Included`)!
Images can be edited with FastLCD image editor which can save Images in **bas** format.
The saved image is then ready to be included in the source program!

Example:
```
Dim Img0 As Flash Byte
Dim Img1 As Flash Byte
```

```
ImgSet(5, 12), Img1    ' Image Img1 will be copied to location

$Included "C:\FastAVR\Img0.bas"    ' Img0 bit image definition
$Included "C:\FastAVR\Img1.bas"    ' Img1 bit image definition
```

Second syntax:
Using ImageArray, a large letters, Icons or Sprites can be displayed, all saved in a single Image!
Any part of this Image is accessable by its index, yeaa - this means animations!

Example:

```
Dim Sclk1616HD As Flash Byte
ImgSet(15, 2, 1), Sclk1616HD    ' SandClock with index 1, second sub-Image will be
displayed

$Included "C:\FastAVR\Sclk1616HD.bas"    ' SandClock definition
```

Related Topics:
GLcd

6.4.17. Inverse

Description:
Inverses specified area on the screen.

Syntax:

```
Inverse(varX, varY, varX1, varL1)
```

Remarks:
varX LeftMost X coordinate of area, between 0 and NofColumns
varL TopMost coordinate, between 0 and Ymax
varX1 number of columns to Inverse
varL1 number of lines (pixels) to Inverse

X coordinates are in Columns not in Pixels!

Example:

```
Inverse(15, 1, 6, 40)    ' Specified area will be Inversed
```

Related Topics:
Fill

6.4.18. Line

Description:
Draws or Clears a Line of any angle.

Syntax:
```
Line(varX, varY, varX1, varY1), 0|1
```

Remarks:
varX X coordinate of start pixel
varY Y coordinate of start pixel
varX1 X coordinate of end pixel
varY1 Y coordinate of end pixel
0|1 0 will Clear Line, 1 will Draw Line

Example:
```
Line(0, 0, 239, 127), 1    ' Diagonal Line on 240x128 pix LCD
```

Related Topics:
LineH
LineV
Pset
Circle
Box

6.4.19. LineH

Description:
Draws or Clears a Horizontal Line.
FS pin setting MUST be on 1 for 6 pix Column wide or 0 for 8 pix column.

Syntax:
```
LineH(varX, varY, varX1), 0|1
```

Remarks:
varX X coordinate of LeftMost pixel in Line
varY Y coordinate of Line
varX1 X coordinate of RightMost pixel in Line
0|1 0 will Clear Line, 1 will Draw Line

varX1 must be greater than **varX**.

Example:
```
LineH(15, 20, 120), 1    ' Line will be Drawn from X=15 to 120, at y=20
```

Related Topics:
LineV

6.4.20. LineV

Description:
Draws or Clears a Vertical Line.
FS pin setting MUST be on 1 for 6 pix Column wide or 0 for 8 pix column.

Syntax:
```
LineV(varX, varY, varY1), 0|1
```

Remarks:
varX X coordinate of Line
varY Y coordinate of TopMost pixel in Line
varY1 Y coordinate of BottomMost pixel in Line
0|1 0 will Clear Line, 1 will Draw Line

varY1 must be greater than **varY.**

Example:
```
LineV(15, 20, 60), 1    ' Vertical Line will be Drawn from y=20 to 60, at x=15
```

Related Topics:
LineH

6.4.21. Point

Description:
Tests if specified Pixel location is Set or Reset.

Syntax:
```
Var = Point(varX, varY)
```

Remarks:
varX X coordinate
varY Y coordinate
var is assigned the result, 0 if pixel is Reset, 1 if Pixel is Set

Example:
```
n = Point(15, 2)    ' If n>0 that Pixel is Set
```

Related Topics:
PSet

6.4.22. Pset

Description:
Sets or Resets an individual Pixel at the desired position.
FS pin setting MUST be on 1 for 6 pix Column wide or 0 for 8 pix column.

Syntax:
```
Pset(varX, varY), 0|1
```

Remarks:
varX X coordinate
varY Y coordinate
0|1 0 will Reset pixel, 1 will Set pixel, (color)

Example:
```
Pset(15, 20), 1    ' Pixel at coordinates 15, 20 will be Set
```

Related Topics:
Point
LineH
LineV

6.4.23. Tcls

Description:
Clears the text area on Graphic LCD.

Syntax:
```
TCls
```

Example:
```
TCls    ' Just text area is now cleared
```

Related Topics:
Fill

6.4.24. TLcd

Description:
Writes text on graphic LCD's Text page defined on `TxtHomeSet`. This mode uses build-in characters generator. User can select beetween 6x8 font (FS=1) ans 8x8 font (FS=0).

Syntax:
```
TLcd(varC, varL), var
```

Remarks:
varC Starting X coordinate in columns
varL Line to write in
var num, string, string constant or hex to write

X and Y coordinates are in Columns and Lines, not in Pixels!

If `$LeadChar` is defined then result will be right justified with Leading Chars as defined. Also, if `Format()` is defined then optional decimal point will be inserted!

Example:
```
TLcd(0, 0), "This is T6963C"    ' Writes string on upper Line
```

Related Topics:
FS pin configuration
GCursor
GCommand

6.4.25. TxtAreaSet

Description:
Sets the Text Page width in Columns.

Syntax:
```
TxtAreaSet col
```

Remarks:
col width of Text page

Text page can be wider then LCD visible area!

Example:
```
TxtAreaSet 20    ' Text page is 20 colums wide
```

Related Topics:
TxtHomeSet
GrpHomeSet
GrpAreaSet

6.4.26. TxtHomeSet

Description:
Sets the Text Page origin. Can be anywhere in the T6963C RAM.

Syntax:
```
TxtHomeSet adr
```

Remarks:
adr address of Text page

Example:
```
TxtHomeSet &h0000    ' Txt page is at the begining of T6963C RAM
```

Related Topics:
TxtAreaSet
GrpHomeSet
GrpAreaSet

6.5. 1WWrite

Description:
1WReset, 1WRead and 1WWrite are the commands used to communicate with Dallas 1 Wire devices.

Syntax:
```
1WWrite [n,] var|exp|func

or block version

1WWrite [n,] var1, m
```

Remarks:
1WWrite writes a variable to the bus (var), the result of an entire expression (exp) or a function result (func)

n is index if more than one 1Wire bus are used, 0 is default for single 1Wire bus or first 1Wire bus!

Example:
```
1wwrite &hcc; &h44      ' writing on first 1Wire bus
1wwrite 2, &hcc; &h44   ' writing on 1Wire bus with index 2
```

Related topics:
$1Wire
1WReset
1WRead

6.6. 1WReset

Description:
1WReset, 1WRead and 1WWrite are the commands used to communicate with Dallas 1 Wire devices.

Syntax:
```
var=1WReset [n]
```

or

```
1WReset [n]
```

Remarks:

1WReset resets the bus and returns the status in var (byte), 0 = there is no 1Wire devices on bus!

n is index if more than one 1Wire bus are used, 0 is default for single 1Wire bus or first 1Wire bus!

Example:
```
a=1wreset, 1   ' reseting secont (index 1) 1Wire bus
```

Related topics:
$1Wire
1WRead
1WWrite

6.7. 1WRead

Description:
1WReset, 1WRead and 1WWrite are the commands used to communicate with Dallas 1 Wire devices.

Syntax:
```
var=1WRead [n]

or block version

1WRead  [n,] var1, m
```

Remarks:
1WRead reads from the 1WIRE device and stores the result in var
Second syntax is special block read, m bytes will be read and stored from var1 up in SRAM. var1 MUST be global!

n is index if more than one 1Wire bus are used, 0 is default for single 1Wire bus or first 1Wire bus!

Example:
```
$1wire=PORTD.3

1wread n, 8     ' block 1Wread, n must be global
x=1wread        ' 1Wread in variable x
```

Related topics:
$1Wire
1WReset
1WWrite

6.8. Abs

Description:
Returns the absolute value of its argument.

Syntax:
```
var=Abs(numeric expression)
```

Remarks:
Var will contain the positive value of the signed **numeric expression**. (integer or Long)

Example:
```
n=-15        'n contains -15
n=Asc(n)     'n will contain 15
```

6.9. Ac

Description:
Defines the type of Analog Comparator Interrupt.

Syntax:
```
Ac type
```

Remarks:
type can be:
Rising
Falling
Toggle

Attention! Default setings is Toggle!

Example:
```
Ac Rising     ' Aci will be triggered on the rising edge.
```

Related topics:
Start
Stop
Enable
Disable

6.10. Acos

Description:
Calculates Inverse Cosine.

Syntax:
```
var=Acos(numeric expression)
```

Remarks:
var receives a Acos of numeric expression
numeric expression must be positive

Acos can return value in Degrees or Radians depends on $Angles Metastatement!

Example:
```
Dim n As Float

n=Acos(0.5)     'n=60, (if $Angles=Deegrees)
```

Related topics:
Sin
Cos
Tan
Asin
Atan

6.11. Adc

Description:
Reads the converted analog value from the ADC (valid only for AVR devices with built in ADC).

Syntax:
```
var=ADC(channel)     ' 10 bits conversion
var=ADC8(channel)    ' 8  bits conversion
```

Remarks:
channel is the number of the ADC channel (mux).
var is a variable that stores the ADC value read.
Adc8(ch) returns 8 bit value.

Note that ADC must be started first!

Example:
```
Start Adc
n=Adc8(i)    ' n = 8 bit ADC value
w=Adc(i)     ' W = 10 bit ADC value
```

Related topics:
Start
Stop

6.12. Asc

Description:
Returns the ASCII code of a character in a string argument.

Syntax:
```
var=Asc(string or string constant [, numeric expression])
```

Remarks:

Returns the ASCII code of the first character or any character that the second optional **numeric expression** is pointing to.

Example:
```
s="A"
n=Asc(s)     'n will contain 65
s="12345"
n=Asc(s, 3) 'n will contain 51 (ASCII code for "4")
```

Related topics:
Chr

6.13. Asin

Description:
Calculates Inverse Sine.

Syntax:
```
var=Acos(numeric expression)
```

Remarks:
`var` receives a Asin of `numeric expression`
`numeric expression`

Asin can return value in Degrees or Radians depends on $Angles Metastatement!

Example:
```
Dim n As Float

n=Asin(0.5)     'n=30, (if $Angles=Deegrees)
```

Related topics:
Sin
Cos
Tan
Asin
Acos
Atan

6.14. Atan

Description:
Calculates Inverse Tangens.

Syntax:
```
var=Atan(numeric expression)
```

Remarks:
`var` receives a Atan of `numeric expression`
`numeric expression`

Atan can return value in Degrees or Radians depends on $Angles Metastatement!

Example:
```
Dim f1 As Float

f1=ATan(1)     'f1=45, (if $Angles=Deegrees)
```

Related topics:
Sin
Cos
Tan
Asin
Acos

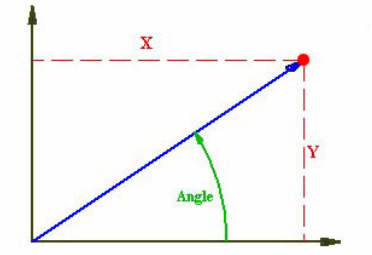
6.15. Atan2

Description:
Returns the angle from the X axis to a Point (x,y) in units defined in `$Angles` (default: Radians)

Syntax:
```
var=Atan2(x,y)
```

Atan2 can return value in Degrees or Radians depends on $Angles Metastatement!

Remarks:
`var` receives an angle Atan2
`y` is a number argument cooresponding to **Y** of point (y,x)
`x` is a number argument cooresponding to **X** of point (y,x)



Example:
```
Dim f1 As Float

f1=ATan2(6,6)     'f1=45, (if $Angles=Deegrees)
```

Related topics:
Sin
Cos
Tan
Asin
Acos
Atan

6.16. Baud

Description:
Overrides the $Baud command.

Syntax:
```
Baud = const [, Parity, DataBits, StopBits]
Baud2 = const              ' for second UART
```

Remarks:
const is the baud rate number - standard values:
1200, 2400, 4800, 9600, 19200, 38400, 56600,76800,115200
but can be any value

Parity N, O, E, M or S (if Parity is set then DataBits must be 9!)
DataBits 8 or 9
StopBits 1 or 2 (in case of 9 DataBits, must be only 1 StopBit)

Example:
```
Baud=1200    'default = N, 8, 1
Baud2=9600    'default = N, 8, 1
```

Related topics:
Print
PrintBin
Start
Stop
Input
InputBin

6.17. Bcd

Description:
Returns the BCD value of a variable.

Syntax:
```
var1=Bcd(var2)
```

Remarks:
var1 is the target variable.
var2 is the source variable.

Example:
```
n=18
m=Bcd(n)    ' m=&h18
```

Related topics:
FromBcd
Chr

6.18. BitWait

Description:
Waits for a specified `PINx.bit` to become 1 or 0.

Syntax:
```
BitWait name 1|0
BitWait PINx.pin 1|0
```

Remarks:
name is the name of `PORT.pin defined with $Def`.
PINx.pin is name of the physical pin.

Note: BitWait waits for specified bit value until this value is reached!

Example:
```
$Def sig=PORTD.5

BitWait sig, 1          'the program waits for 1
BitWait PIND.4, 0      'the program waits for 0
```

6.19. Case

Select

6.20. Chr

Description:
Converts ASCII code into the corresponding ASCII character.

Syntax:
```
var1=Chr(var2)
```

Remarks:
var1 is the target variable.
var2 is the source variable.

Example:
```
n=65
Print Chr(n)          'Displays A
```

Related topics:
Asc
BCD

6.21. Const

Description:
Declares a constant.

Syntax:
```
Const name=NumConst [, simple expression]
```

Remarks:
name is a name of your choice.
NumConst is the value of the constant.

Example:
```
Const True=1, False=0

Const time=250
Const uppr=time+50   ' uppr is 300
Const s="FastAVR"    ' String constants also
```

Related topics:
$Def

6.22. Cls

Description:
Clears the LCD and sets the cursor to home position.

Syntax:
```
Cls
```

Example:
```
Cls     'Clears the LCD
```

Related topics:
LCD
Locate
Cursor
Display

6.23. Cos

Description:
Returns the trigonometric Cosine of its argument.

Syntax:
```
var=Cos(numeric expression)
```

Remarks:
`var` type Float receives a Cosine of `numeric expression`
`numeric expression` can be in Radians or Deegrees, depending on `$Angles` metastatement!

Example:
```
Dim n As Byte
Dim f1 As Float

n=30          ' assuming $Angles=Degrees
f1=Cos(n)     ' f1=0.8660254
```

Related topics:
Sin
Tan
Asin
Acos
Atan

6.24. Cosh

Description:
Returns the Cosine Hiperbolicus of its argument.

Syntax:
```
var=Cosh(numeric expression)
```

Remarks:
`var` type Float receives a Cosh of `numeric expression`
`numeric expression`

Example:
```
Dim f1 As Float

f1=Cosh(5)     'f1=74.20995
```

Related topics:
Sinh
Tanh

6.25. CPeek

Description:
Returns a Byte from program memory (flash).

Syntax:
```
var=CPeek(adr)
```

Remarks:
var The variable that is assigned.
adr The address in program memory.

Example:
```
m=CPeek(n)
```

Related topics:
Poke
Peek

6.26. Crc8

Description:
Calculates 8bit crc value in SRAM, starting at adr of var.

Syntax:
```
var1=Crc8(var, n)
```

Remarks:
var1 is the calculated Crc value.
var itself is the starting address in SRAM (If here is variable, then address of this variable is used.
n is the number of bytes to calculate Crc.

Example:
```
Dim n(8) As Byte
Dim Crc As Byte

Crc=Crc8(n, 8)       'calculate 8bit crc 8bytes from n up

Crc=Crc8(VarPtr(n)+3 ,8)       'we are adding here 3 to the address of variable n
```

6.27. Cursor

Description:
Controls the LCD cursor behavior.

Syntax:
```
Cursor On|Off|Blink|NoBlink
```

Remarks:
Default is Off and NoBlink

Example:
```
Cursor Off      'Cursor is not visible
Cursor On       'Cursor is visible
Cursor Blink    'Cursor is blinking
```

Related topics:
LCD
Locate
Cls
Display

6.28. Data

Look at Dim

6.29. Declare

Description:
Explicitly declares a user Subroutine or Function.

Syntax:
```
Declare Sub SubName([par1 As type] [, par2 As Type])
```

```
Declare Function FuncName([par1 As type] [, par2 As type]) As rType
Declare Interrupt IntType()
```

Remarks:
SubName is a subroutine name of your choice.
FuncName is a function name of your choice.
parx is a name of passing parameters to the Sub or Function
rType is type of the returned value of function, (Byte, Integer, Word or Long)
IntType is the type of Interrupt (look at Interrupts)

Very important: Declared .... belongs to the header of the program, before any other normal statements!
Parameter's names used in Declare statements MUST be the same as names in the actual Subs or Functions!

Example:
```
Declare Sub Test(n As Byte)                 'declares a Sub Test
Declare Function Test1(n As Byte) As Byte   'declares a Function Test1
Declare Interrupt Ovf1()                   'declares a Timer1 Overflow Interrupt routine
```

6.30. Decr

Description:
Decrements var by 1

Syntax:
```
Decr var
```

Remarks:
var is a numeric variable.

Example:
```
Decr a    'a=a-1
```

Related topics:
Incr

6.31. DefLcdChar

Description:
Defines one of eight LCD special characters.

Syntax:
```
DefLcdChar n, byte1, byte2, byte3, byte4, byte5, byte6, byte7, byte8
```

Remarks:
n number of special character (0-7)
bytex 8 bytes defining special character

Graphic editor for LCD characters will automaticaly insert this statement at current cursor position, but user has to

modify **n** from 0 to range 0-7!

Example:
```
DefLcdChar 0, &h08, &h10, &h1C, &h00, &h00, &h04, &h18, &h00
Lcd Chr(0)
```

6.32. Dim

Description:
Declares and dimensions arrays and variables and their types.

Syntax:
```
Dim VarName As [Xram|Flash] Type [At &h1000]
Dim VarName(n) As Type
```

Remarks:
VarName is the variable name.
type is one of the following variable types:

Bit	uses one of 16 reserved bits (R2 and R3)
Byte	uses one byte of RAM memory
Integer	uses one two of RAM memory
Word	uses two bytes of RAM memory
Long	uses four bytes of RAM memory
Float	uses four bytes of RAM memory
String * Length	uses "length" Bytes of RAM memory, plus one more for termination of the string.

Length is the number of string variable elements(characters).
n is the number of array elements
Xram var will be placed in external RAM at address specified after **At** in hex.
Flash constants will be placed in Flash at address specified by VarName.

Very important: Dim .... belongs to the header of the program, just after Declares and BEFORE any other normal statements!

Attention:
Data and Lookup keywords were removed because this mechanism didn't allow the whole range of data types to be built!
Here is the new implementation for table use.

```
Dim TableName As Flash Type
```

TableName is table of specific type of constant in Flash.
User can fill table:

```
TableName = 11,22,33,44,
55,66,77,88
```

User can fill also a table of strings:

```
TableName = London, Newyork, Paris, Roma,
Berlin, Ljubljana, Madrid, Amsterdam
```

As you can see, data can continue in the next line and stops where the comma is missing!
Access to table:

```
var=TableName(index)
```

Tables in Flash MUST be initialized at the END of Program!

Example:

```
Dim a As Byte             'global byte variable named a
Dim w As Word             'global word variable named w
Dim f As Float           'global word variable named w
Dim db(10) As Byte        'global array of ten bytes named n
Dim s1 As String * 8      'global string variable named s1, length must be specified
Dim s2 As String * 9      'global string variable named s2, length must be specified
Dim a As Xram Byte        'global byte variable named a in Xram
Dim w(16) As Flash Word 'global word constant in Flash (table), number of elements could be
also declared
Dim s As Flash String    'global string constants in Flash (table), without strings length
```

Array of Strings Example:

```
Dim MyString(10) As String * 7  ' ten Strings seven characters each (total 8)

MyString=(" ")                  ' will init all array elements to " "
MyString=("123", "ABC", "    ")  ' will init first three elements
```

Arrays, Bits and Strings can not be Local variables!

Related topics:
Local

6.33. Disable

Description:
Disables Global Interrupts and/or individual Interrupts.

Syntax:

```
Disable Interrupts
Disable int
```

Remarks:
int is a valid Interrupt type

Look at Interrupts

Example:

```
Disable Interrupts  'disables Interrupts (Global)
Disable Ovf1        'from now on Ovf1 is disabled
```

Related topics:
Enable
Interrupts

6.34. Display

Description:
Controls the LCD ON or OFF.

Syntax:
```
Display On|Off
```

Remarks:
Default is On.

Example:
```
Display On      'Display is ON
Display Off     'Display is OFF
```

Related topics:
LCD
Locate
Cls
Display

6.35. DegToRad

Description:
Converts Deegrees to Radians.

Syntax:
```
var=DegToRad(var1)
```

Remarks:
var The var that is assigned Radians.
var1 The Deegrees to convert.

Example:
```
Dim n As Byte
Dim f1 As Float, f2 As Float

f1=180
f2=DegToRad(f1)    'f2=3.141593
```

Related topics:
RadToDeg
$Angles

6.36. Do

Description:
Defines a loop of statements that are executed while a certain condition is thrue. Because test for condition is at the end of the loop, the loop itself will be executed at least once!

Syntax:
```
Do
   statements
   Exit Do              'you can EXIT from the loop at any time
Loop [While condition]
```

Remarks:
condition The Numeric or string expression that evaluates to True or False.

Statements within loop are executed at least one time, because test for condition is at the end of loop.

Useful for never ending loop.

Example:
```
Dim i As Byte

Do                  ' never ending loop
   For i=0 To 5
      Print Adc8(i)
      Waitms 250
   Next
Loop
```

Related topics:
While-Wend

6.37. DTMF

Description:
Generates DTMF tone.

Syntax:
```
DTMF(var)
```

Remarks:
var, expression or constant must be beetween 0 and 15

Key var and Tones Table

0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8

```
 9  9
10  A
11  B
12  C
13  D
14  *
15  #
```

ATTENTION!
Timer1 and OVF1 interrupt are used. User can NOT use this timer and interrupt for other purposes!

User MUST enable global interrupts

```
Enable Interrupts
```

and Ovf0 interrupt

```
Enable Ovf1
```

Note also that some filtering is necesary at the output (pin OC1)!

Example:

```
Enable Interrupts  'user must enable interrupts

DTMF(5)
```

Related topics:
$DTMF

6.38. Enable

Description:
Enables Global Interrupts and/or individual Interrupts.

Syntax:

```
Enable Interrupts
Enable int
```

int is a valid Interrupt type

Remarks:
Check Interrupt types for each microcontroller used!

Example:

```
Enable Interrupts   'enables global Interrupts
Enable Ovf1         'enables Timer1 Ovf1 Interrupt
```

Related topics:
Disable
Interrupts

6.39. End

Description:
Ends program execution.

Syntax:
```
End
```

Remarks:
It is not necessary to use this statement if You are using a never-ending loop.

6.40. Exit

Sub
Function
For-Next
Do
While

6.41. Exp

Description:
Calculates a base number raised to a power.

Syntax:
```
var=Exp(numeric expression)
```

Remarks:
`var` receives a Exp of `numeric expression`
`numeric expression`

Example:
```
Dim n As Float

n=Exp(3.45)     'n=3.150039E+01
```

Related topics:
Log
Log10
Pow

6.42. Find8

Description:
Finds a value in a Table of bytes.

Syntax:
```
var=Find8(var1, TableName)
```

`var` returned value
`var1` search value
`TableName` Table which will be searched,

Remarks:
Function returns position of searched element or 0 if there is no such element!
Table must be declared as data in flash with number of elements and initialized (at the end of program).

Example:
```
Dim n As Byte
Dim Table(9) As Flash Byte  ' number of elements MUST be also declared

n=Find(55, Table)           ' n will be 5

Table = 11,22,33,44,55,66,77,88,99  'table of 9 elements
```

Related topics:
Find16

6.43. Find16

Description:
Finds a value in Table of Words or Integers.

Syntax:
```
var=Find16(var1, TableName)
```

`var` returned value
`var1` search value
`TableName` Table which will be searched

Remarks:
Function returns position of searched element or 0 if there is no such element!
Table must be declared as data in flash with number of elements and initialized (at the end of program).

Example:
```
Dim n As Word
Dim Table1(9) As Flash Word  ' number of elements MUST be also declared

n=Find16(5555, Table1)       ' n will be 5

Table1 = 1111,2222,3333,4444,5555,6666,7777,8888,9999  'table of 9 elements
```

Related topics:
Find8

6.44. For

Description:
Defines a loop of program statements whose execution is controlled by a loop counter.

Syntax:
```
For counter=start To stop [Step [-] StepValue]
      statements
      [Exit For]  'you can EXIT from the loop at any time
Next
```

Remarks:
counter numeric variable - **Byte, Integer or Word!**
start numeric expression specifying initial value for counter
stop numeric expression giving the last counter value
stepvalue numeric constant, default is 1, can be negative for decrement

Attention! Counter MUST be Byte, Integer or Word!

Example:
```
Dim i As Byte

Do
   For i=0 To 5
      Print Adc8(i)
      WaitMs 250
   Next
Loop
```

Related topics:
Do-Loop
While-Wend

6.45. Format

Description:
Defines Format for Print, Lcd, Tlcd, Glcd and Str().

Syntax:
```
Format(Int|Scientific,Frac)
```

Remarks:
Int Number of Integer numbers, - for Float max 10, min 1. `Scientific` (or 0) for scientific format, `-x.xxxxxE+yy`
Frac Number of Fractal numbers, - for Float min 1.

Floating point numbers can be presented in normal: `6532.818  'Format(4,3)`
or in Scientific Format: `6.532818E+03  'Format(Scientific,7)`

Scientific format is Default for Float numbers!

Floating point numbers of Single precision has max 7 nums but can lay beetween 10E+38 and 10E-38.
The last (7th) num is already rounded!

Note that Format MUST be first mentioned under metastatement $LeadChar (for Byte, Integer, Word and Long - not for Float)!
Numbers are RIGHT justified!

Example:
```
Dim n As Byte, x As Word, f As Float
Dim s As String*5

n=65: w=1234: f=-1.234E+02

Format(3,0)
Print n
'output will be 065 (assume that "0" is defined as LeadChar)

Format(4,1)
Print w
'output will be 0123.4 (assume that "0" is defined as LeadChar)

Format(5,3)
Print f
'output will be -00123.400 (assume that "0" is defined as LeadChar)

Format(Scientific,3)
Print f
'output will be -1.234E+02
```

Related topics:
$LeadChar

6.46. Fract

Description:
Returns Fractional part of Float argument as Float.

Syntax:
```
var=Fract(numeric expression)
```

Remarks:
`var` receives a Fractional part of `numeric expression`
`numeric expression`

Example:
```
Dim n As Float

n=Fract(3.45)     'n=0.45
```

Related topics:
Int

6.47. FromBcd

Description:
Calculates value from BCD format.

Syntax:
```
var1=FromBcd(var2)
```

Remarks:
var1 is the target variable.
var2 is the source BCD variable.

Example:
```
m=FromBcd(n)
```

Related topics:
BCD

6.48. Function

Description:
Defines a Function procedure.

Syntax:
```
Function NameOfFunc(parameters list) As Type
```

Remarks:
NameOfFunc is the name of Function
parameters list is the name and type of parameters, comma delimited (can not be Bit) - MUST has the same names as is in theirs Declare statements!
As Type is type of returned value (Byte, Integer, Word, Long or Float)

Function must first be declared with Declare keyword.

Example:
```
Declare Function Mul(a As Byte, b As Byte) As Byte
Dim n As Byte

n=Mul(5, 7)      'n=35

'////////////////////////////////////////////////////
Function Mul(a As Byte, b As Byte) As Byte

Return a*b
[Exit Function]    ' optionally exit from Function
End Function       ' end of Function
```

Related topics:
Declare
Return
Sub

6.49. GoTo

Description:
Transfers program execution to the statement identified by a specified label.

Syntax:
```
Goto label
```

Remarks:
label is a line identifier indicating where to jump

Example:
```
Point:       'a label must end with a colon

Goto Point
```

6.50. I2CRead

I2CStart

6.51. I2CStart

Description:
I2CStart starts the I2C transfers.
I2CStop stops the I2C transfers
I2CRead receives a single byte through I2C bus
I2CWrite sends a single byte through I2C bus

Syntax:
```
I2CStart adr
var1=I2CRead, Ack [, Nack]
I2CWrite var2
I2CStop
```

Remarks:
adr The address of the I2C-device.
var1 The variable that receives the value from the I2C-device.
var2 The variable or constant to write to the I2C-device

Ack - tells that there will be more Readings
Nack - this is last Reading (MUST be present)

Dont forget pulup resistors on SDA and SCL (4k7 - 10k)!
Notes about I2Cread: The first call to I2Cread will isue also device address for Read (last bit is automaticaly set), last **I2Cread** command MUST be used with **Nack** option!

Example1:
```
' reading time from Philips PCF85x3
I2cstart &ha0       'generate start
```

```
I2cwrite 2          'select second register
s=I2cread, Ack
m=I2cread, Ack
h=I2cread, Nack
I2cstop             'generate stop
```

Example2:

```
'//////////////////////////////////////////////////
Sub WritePage(Padr As Word)
Local i As Byte

I2Cstart &ha0           ' 24C64 address
I2Cwrite Msb(Padr)      ' write adr MSB
I2Cwrite Padr           ' write adr LSB
For i=0 To 31
    I2Cwrite buff(i)    ' write data from array buff
Next
I2Cstop
WaitMs 6                ' wait to write (10 max)
End Sub
```

Example3:

```
'//////////////////////////////////////////////////
Sub ReadPage(Padr As Word)
Local i As Byte, tmp As Byte

I2Cstart &ha0           ' 24C64 address
I2Cwrite Msb(Padr)      ' write adr MSB
I2Cwrite Padr           ' write adr LSB

tmp=I2Cread, Ack        ' first read deefers from others
For i=0 To 31
    buff(i)=tmp         ' we read into buff array
    tmp=I2Cread, Ack
Next
tmp=I2Cread, Nack       ' dummy read - we need Nack!
I2Cstop
End Sub
```

Related topics:

I2CStop
I2CWrite
I2CRead

6.52. I2CStop

I2CStart

6.53. I2CWrite

I2CStart

6.54. If

Description:
Conditionally executes a group of statements, depending on the value of an expression(s).

Syntax:
```
If expression Then statement

or

If expression Then
   statements
ElseIf expression Then
   statements
.
.

Else
   statements
End If
```

Remarks:
While testing bit variables of any kind (bit var, port.bit or var.bit) only "=1", "=0" or nothing can be used!
Conditions and statements may be contained on one line or multiple lines.
Instead of using many `ElseIfs`, `Select Case` may be used!

Example:
```
If a>5 And a<10 Then
   Print a; " a is Between 5 and 10"
ElseIf a=5 Then
   Print a; " a is 5"
Else
   Print b; " a has other value"
End If

If a Then   'If a>0 generates extra comact code
   Print a
End If

If PINB.5 Then
   Print a
End If

If a.3 Then
   Print a
End If
```

Related topics:
Select

6.55. Incr

Description:
Increments var by 1

Syntax:
```
Incr var
```

Remarks:
var variable to increment

Example:
```
Incr a    ' a=a+1
```

Related topics:
Decr

6.56. InitLcd

Description:
Re initialize alphanumeric LCD.

Syntax:
```
InitLcd
```

Remarks:
If LCD was turned OFF because of entering in one of the PowerDown modes it needs to be ReInitialized after waken up.

Example:
```
InitLcd
```

Related topics:
$Lcd
Lcd

6.57. InitEE

Description:
Initialize EPROM data to be written during device programming.

Syntax:
```
InitEE = 11, 22, 33, 44,
         55, 66, 77, 88
```
Remarks:
`InitEE` will produce a hex file named BasName.eep for EPROM programming starting at adr 0!
Numeric constants are comma delimited and can be placed in more than one line.

Related topics:
ReadEE
WriteEE

6.58. Input

Description:
Returns the value or string from the RS-232 port.

Syntax:
```
Input ["prompt"], var1, var2, ....
Input2 ["prompt"], var1, var2, ....   ' for second UART
```
Remarks:
`prompt` is an optional string constant printed before the prompt character.
`varX` is/are the variable(s) to accept the input value or a string, not for Longs!

With the built-in terminal emulator this statement makes the PC keyboard an input device.

Example:
```
Input s
Input n, w
Input "n="; n; "w="; w

Input2 s
```
Related topics:
Print
PrintBin
InputBin

6.59. InputBin

Description:
Returns a binary value(s) from the RS-232 port.

Syntax:
```
InputBin var1; var2;...
InputBin var, n

InputBin2 var1; var2;...    ' for second UART
```

Remarks:
var, var1, var2 variables that receive a binary value from serial port
n number of bytes to receive. Bytes will be stored from var up!

The number of bytes to read depends on the variable lenth You use, 1 for byte, 2 for integer or word.

Example:
```
InputBin a; w     ' waits three bytes
InputBin a, 12    ' waits for 12 bytes (from a up)

InputBin2 a; w     ' waits three bytes
```

Related topics:
PrintBin

6.60. Int

Description:
Returns Integer part of Float argument as Float.

Syntax:
```
var=Int(numeric expression)
```

Remarks:
`var` receives an Integer part of `numeric expression`
`numeric expression`

Example:
```
Dim n As Float

n=Int(372.41855)     'n=372.0
```

Related topics:
Fract

6.61. IntX

Description:
Defines the type of external Interrupt trigger.

Syntax:
```
Intx type
```

Remarks:
x interrupt number 0-7
type can be:
Rising
Falling
Low

Attention! Default setings is Low!

Example:
```
Int0 Rising    ' Int0 will be triggered on the rising edge.
```

6.62. Instr

Description:
Return the first occurrence of a specified string into another string.

Syntax:
```
var=Instr([var1,] mainString, matchString)
```

Remarks:
`var` receives possition of matchString in mainString or **0** if matchString is not in mainString or any error.
`var1` starting search position, can be omitted.
`mainString` source String (MUST be string var, not string constant)
`matchString` can be String Constant.

Example:
```
Dim Name As String*15
Dim Part As Byte

Name="Mona Lisa"
Part=Instr(Name, "Li")    'Part=6
```

Related topics:

6.63. Key()

Description:
Returns a byte in var representing a pressed key in the line or matrix keyboard!

Syntax
```
var=Key()
NoKey()  only for line switches, waits until user releases keys.
```

Remarks:
var contains the pressed key, returns 0 if no key is pressed.

Note: If your main loop (or other loops) is very bussy then calling Key must be done in EXternal interrupt routine.

Example:
```
a=Key()
NoKey()     'waits until user releases keys
```

Related topics:
PcKey
RC5

6.64. LCase

Description:
Returns an all-lowercase version of its string argument.

Syntax:
```
var=LCase(var1)
```

Remarks:
var lowercase string version of var1
var1 original string variable.

Argument MUST be single String variable!

Example:
```
Dim Name As String*15
Dim Part As String*10

Name="Mona Lisa"
Part=LCase(Name)    'Part="mona lisa"
```

Related topics:
UCase

6.65. Lcd

Description:
Prints to standard ASCII LCD modules.

Syntax:
```
Lcd var1; var2;...
Lcd Hex(var1)
```

Remarks:
var1, var2 are vars to be printed on LCD
Hex(var1) var1 will be printed in hexadecimal format

If `$LeadChar` is defined then result will be right justified with Leading Chars as defined. Also, if `Format()` is defined then optional decimal point will be inserted (even with Integer variables!)!

Example:
```
Lcd "FastAVR Basic Compiler!"
Locate 2, 1: Lcd "n="
Do
    Locate 2, 3: Lcd n
    Incr n
    WaitMs 250
Loop
```

Related topics:
LCD
Locate
Display
Cursor
DefLcdChar
InitLcd

6.66. Left

Description:
Returns the leftmost n characters of a string.

Syntax:
```
var=Left(var1, n)
```

Remarks:
var string that Left chars are assigned.
var1 original string (MUST be string var, not string constant).
n number of characters to be returned from left.

Example:
```
Dim Name As String*15
Dim Part As String*10

Name="Mona Lisa"
Part=Left(Name, 4)    'Part="Mona"
```

Related topics:
Right
Mid

6.67. Len

Description:
Returns the length of a string.

Syntax:
```
var=Len(string var)
```

Remarks:
var string that receives Legth in chars of string var.
string var original string.

Example:
```
Name="Mona Lisa"
n=Len(Name)      'n=9
```

Related topics:
Left
Right
Mid
Str

6.68. Local

Description:
Declares and dimensions Local variables, seen only inside of Subs and/or Functions.

Syntax:
```
Local VarName As Type
```

Remarks:
VarName is the variable name.
type is one of the following variable types:

Byte	uses one byte of RAM memory
Integer	uses one two of RAM memory
Word	uses two bytes of RAM memory
Long	uses four bytes of RAM memory
Float	uses four bytes of RAM memory

Length is the number of string variable elements(characters).
n is the number of array elements

Example:
```
'////////////////////////////////////////////////
Sub Test(n As Byte)
Local a As Byte             'local byte variable named a
```

```
Local w As Word             'local word variable named w
```

Body of Sub

```
End Sub
```

Arrays, Bits and Strings can not be Local variables!

Related topics:
Dim

6.69. Locate

Description:
Locates the position for the next character to be printed.

Syntax:
```
Locate row, var1
Locate adr
```

Remarks:
row is a numeric constant representing the row to print in.
var1 is a requested column value
adr is an alternative absolute address for positioning on the LCD. See LCD data sheets for actual addressing!

Example:
```
Locate 2, 3: Lcd n   'n will be printed in second row at position 3
```

Related topics:
LCD
Locate
Display
Cursor

6.70. Log

Description:
Calculates Natural Logarithm (base e).

Syntax:
```
var=Log(numeric expression)
```

Remarks:

`var` receives a Natural Logarithm of `numeric expression`
`numeric expression` must be positive

Example:
```
Dim n As Float
Dim w As Word

w=12345
n=Log(w)     'n=9.421006E+01
```

Related topics:
Exp
Log10
Pow

6.71. Log10

Description:
Calculates base 10 Logarithm.

Syntax:
`var=Log10(numeric expression)`

Remarks:
`var` receives a Log10 of `numeric expression`
`numeric expression`

Example:
```
Dim n As Byte
Dim w As Word

w=12345
n=Log10(w)     'n=4.09149E+01
```

Related topics:
Exp
Log
Pow

6.72. Lookup

Look at Dim

6.73. Loop

Do

6.74. MakeWord

Description:
Makes Word or Integer from two bytes.

Syntax:
```
MakeWord(var1, var2)
```

`var1` MSB byte
`var2` LSB byte

Remarks:
Very suitable for copying a portion of SRAM.

Example:
```
Dim n As Byte
Dim m As Byte
Dim w As Word

n=&h12
m=&h34

w=MakeWord(n, m)   ' w=&h1234
```

Related topics:
Msb

6.75. MemLoad

Description:
Quickly loads some SRAM locations.

Syntax:
```
MemLoad (var, const1, const1,...)
```

`var` SRAM will be loaded from `var` on.
`constx` constants to load with.

Remarks:
Very suitable for initializing variables in SRAM.

Example:
```
MemLoad (VarPtr(n), 4, 4, 4, 15, &hff, &hff)
MemLoad (&h90, "String constants also!", "Test")
```

Related topics:
MemCopy

6.76. MemCopy

Description:
Quick SRAM block copy from Source locations to Destination.

Syntax:
```
MemCopy (var1, var2, var3)
```

`var1` number of bytes to copy
`var2` we will copy from here - Source
`var3` to here - Destination

Remarks:
Very suitable for copying a portion of SRAM.
Var2 and Var3 MUST be address (note that Strings "value" are already Addresses)!

Example:

```
MemCopy(12, VarPtr(Buff1), VarPtr(Buff2))                    ' copies 12 Bytes from Buff1 to
Buff2
```

Related topics:
MemLoad

6.77. Mid

Description:
Return a specified number of characters in a string.

Syntax:
```
var=Mid(var1, n1, n2)
```

Remarks:
var string that Mid chars are assigned.
var1 source string (MUST be string var, not string constant).
n1 starting position of characters from left.
n2 number of characters.

Example:
```
Dim Name As String*15
Dim Part As String*10

Name="Mona Lisa"
Part=Mid(Name, 2, 5)    'Part="ona L"
```

Related topics:
Right
Left

6.78. MSB

Description:
Returns the most significant byte of the Word var.

Syntax:
```
var=Msb(var1)
```

Remarks:
var byte variable that is assigned.
var1 word variable.

Example:
```
Dim n As Byte
Dim x As Word

n=x        'n holds Lsb byte of x
n=Msb(x)   'n holds Msb byte of x
```

Related topics:
MakeWord

6.79. Next

For

6.80. Nokey()

Key()

6.81. Nop

Description:
Generates one or more Nop assembler statements.

Syntax:
```
Nop[n]
```

Remarks:
n optional numeric for more Nops

Usefull for very short delays. Nop takes 1/Q seconds.

Example:
```
Nop      ' generates 1 Nop
Nop 5    ' generates 5 Nops
```

6.82. On x GoTo

Description:
Jumps to one of listed Labels or executes one of listed Subs, depending on value x.

Syntax:
```
On x GoTo Label0, Label1, Label2.....
On x Sub0(), Sub1(), Sub2() ....
```

Remarks:
X is a test variable or expression.
LabelN labels to jumt to.
SubN subs to call.

Example 1:

```
On n GoTo Test, Lab1, Label3
Back:
.
.
.
End

Test:
   DoTest()
   GoTo Back

Lab1:
   DoSomething()
   GoTo Back

Label3:
   DoOther()
   GoTo Back
```

Example 2:

```
On n Test(), Lab1(), Label3()
.
.
.
End


Sub Test()
   DoTest()
End Sub

Sub Lab1()
   DoSomething()
End Sub

Sub Label3()
   DoOther()
End Sub
```

Related topics:
Select

6.83. Open COM

Description:
Opens up to four software UART channels.

Syntax:
```
Open Com=PORT.pin, speed [, Inv] For Input|Output As #n
```

Remarks:
speed is the baud rate
n is Com number from 1 to 4
Inv option for inverted signal

Interrupts must be disabled during transmiting or receiving throu this software routins!

Example:
```
Open Com=PORTD.0, 9600 For Input As #1
Open Com=PORTD.1, 9600 For Output As #1
Open Com=PORTD.2, 19200, Inv For Output As #2

Do
   InputBin #1, a, 3  ' input three bytes thru Com1
   Print #1, a; b; c  ' print vars on Com1
   Print #2, "test"   ' print inverted string constant on Com2
Loop
```

6.84. PcKey()

Description:
Returns a scan code of pressed key on standard AT-PC keyboard.

Syntax
```
var=PcKey()
```

Remarks:
var contains the scan code of pressed key

Connected AT-PC keyboard works with Scan Code Set 3, so only one byte (make) is received! (default mode for keyboard when connected to PC is Scan Code Set 2)
See file PCcode.pdf!

Note: PcKey function will pool CLOCK line until a scan code will be transmitted! This call can be used in External interrupt routine if this is not acceptable (because some tasks in main loop must be proccessed).

Example:
```
PcKeySend(&hf9)    ' turn autorepeat off
a=PcKey()
```

Related topics:
PcKeySend()

6.85. PcKeySend()

Description:
Send a command or data to standard AT-PC keyboard.

Syntax
```
PcKeySend(const)
```

Remarks:
const is a valid command or data

Connected AT-PC keyboard works with Scan Code Set 3, so, only one byte (make) is received! (default mode for keyboard is Scan Code Set 2)
See file PCcode.pdf!

Scan codes select:
This two-byte command selects the scan code set. Scan code set 2 is selected by default after a reset. However, scan code set 3 is selected by PCkey() init routine since set 3 is recommended for microcontroller applications.
Command: `&hf0`
Command: `&b000000xx`
`01`: scan code set 1
`10`: scan code set 2
`11`: scan code set 3

Set all keys:
This commands assign attributes to the keys, as fallows:
Command: `&hf7` all keys have the repeat function (default is no repeat)
Command: `&hf8` all keys produce Make and Break codes
Command: `&hf9` all keys produce only a Make code (no repeat)
Command: `&hfa` all keys have the repeat function and produce Make and Break code

This two-byte command controls the behavior of the LEDs.
Command: `&hED`
Command: `&b00000xxx`
Bit 0: Scroll lock
Bit 1: Num lock
Bit 2: Caps lock

Reset Command: `&hff`

Set Spermatic Rate/Delay:
Command: &hf3
Command: &b0xxxxxxx

```
Bit6 Bit5 Delay
0    0    150ms
0    1    500ms
1    0    750ms
1    1    1  s

Bit4 Bit3 Bit2 Bit1 Bit0 Autorepeat
0    0    0    0    0    30hz
0    1    1    1    1     8hz
1    1    1    1    1     2hz
```

Example:
PcKey()

See also:
PcKey()

6.86. Peek

Description:
Reads a byte from internal or external SRAM.

Syntax:
```
var=Peek(var1)
```

Remarks:
var The string that is assigned.
var1 The address to read the value from.

Example:
```
Adr=&h70
n=Peek(Adr)        ' read value from SRAM address &h70
```

Related topics:
Poke
Cpeek

6.87. Poke

Description:
Writes a byte to internal or external SRAM.

Syntax:
```
Poke(var1, var2)
```

Remarks:
var1 The address in internal or external SRAM.
var2 The value to be placed in SRAM.

Example:
```
Adr=&h70
Poke(Adr, 5)    ' write 5 to SRAM address &h70
```

Related topics:
Peek
Cpeek

6.88. Pow

Description:
Returns the Power of its argument.

Syntax:
```
var=Pow(numeric expression, exponent)
```

Remarks:
`var` type Float receives a Power of `numeric expression`
`numeric expression` any numeric type
`exponent` any numeric type

Example:
```
Dim n As Byte
Dim f1 As Float, f2 As Float

f2=5.24
n=7

f1=Pow(f2, n)     'f1=1.084723E+05
```

Related topics:
Exp
Log
Log10

6.89. PowerModes

Description:
Forces processor into one of Power Saving Mode.

Sleep modes enable the application to shut down unused modules in the MCU, thereby saving power. The AVR provides various sleep modes allowing the user to tailor the power consumption to the application's requirements.

Note that not all PowerModes are available in each AVR device! Check datasheets!

Syntax:
```
Idle
PowerDown
PowerSave
Standby
ExtStandby
```

Remarks:
`Idle` CPU sleeps after this statement, but the Timers, Watchdog and Interrupt system continue to operate. This power-saving mode is terminated with reset or when an interrupt is received.

`PowerDown` CPU draws only a few micro amperes because the external oscillator is stopped. Only an external reset, a watchdog reset, an external level interrupt or a pin change interrupt can wake up the CPU.

`PowerSave` This mode is identical to PowerDown but the CPU can be also be awakened with Timer2.

`Standby` This mode is identical to Power-down with the exception that the Oscillator is kept running. From Standby mode, the device wakes up in six clock cycles.

`ExtStandby` This mode is identical to Power-save mode with the exception that the Oscillator is kept running. From Extended Standby mode, the device wakes up in six clock cycles..

Example:
```
PowerDown
```

6.90. Print

Description:
Send a variable or constant to the RS-232 port.

Syntax:
```
Print var1; var2; ....
Print2 var1; var2; ....    ' for second UART
```

Remarks:
var1 variable or constant to print
var2 variable or constant to print

You can use a semicolon `;` to print more than one variable on a line.
When you end a line with a semicolon, no linefeed will be added.

With the built-in terminal emulator, you can easily monitor print statements.

If `$LeadChar` is defined then result will be right justified with Leading Chars as defined. Also, if `Format()` is defined then optional decimal point will be inserted!

Example:
```
Dim n As Byte, x As Word
Dim s As String*5

n=65: w=1234: s="Test"

Print n
Print w
Print s
Print n; w
Print "n="; n; "w="; w
Print Bcd(n)
Print Hex(w)
Print Chr(n)

Print2 n

End
```

Related topics:

Input
PrintBin
InputBin

6.91. PrintBin

Description:
Sends a binary value(s) to the serial port.

Syntax:
```
PrintBin var1; var2;...
PrintBin var, n

PrintBin2 var1; var2;...    ' for second UART
```

Remarks:
var, var1, var2 byte or word sent to the serial port
n number of bytes to send from var up! With this statement you can send the whole SRAM byte by byte!

The number of bytes to send depends on the variable you use, 1 for byte, 2 for word.

Example:
```
Dim a As Byte, w As Word

a=5: w=&h3f12

PrintBin a; w     ' three bytes will be sent
PrintBin a, 12    ' 12 bytes will be sent (from a up)

PrintBin2 a; w     ' three bytes will be sent
```

Related topics:
InputBin

6.92. Pulse

Description:
Generates a pulse on the specified AVR port pin.

Syntax:
```
Pulse Port.pin, 0|1, var
```

Remarks:
0 pulse from 1 to 0 and back to 1
1 pulse from 0 to 1 and back to 0
var defines pulse length according to formula: t=(3*var+8)/clock
For clock 8MHz and var=1 pulse will be 1.375us.
AVR port pin must first be configured as output.

Example:
```
Pulse PortB.2, 1, 10     'pulse pin high for 10.3us
                        'then return to low
```
Related topics:
Set
Reset
toggle

6.93. RadToDeg

Description:
Converts from Radians to Deegrees.

Syntax:
```
var=RadToDeg(var1)
```

Remarks:
var The var that is assigned Deegrees.
var1 The Radians to convert.

Example:
```
Dim n As Byte
Dim f1 As Float, f2 As Float

f1=3.14159265
f2=RadToDeg(f1)    ' f2=180
```

Related topics:
DegToRad
$Angles

6.94. RC5

Description:
Receives the Philips RC5 standard remote IR code.

Syntax:
```
Rc5(sysadr, command)
```

Remarks:
sysadr is a RC5 family address (Byte)
command is the code of the pressed key (Byte)
Sysadr and Command vars must be declared with `Dim` first!
TOGGLE BIT is `sysadr.5`
Command is six bits long, sysadr is five bits!
In case of bad reception RC5 returns 255 in Command, garbage in sysadr!

ATTENTION!

Timer0 and OVF0 interrupt are used. User can NOT use this timer and interrupt for other purposes!
User MUST enable global interrupts and Ovf0 interrupt!

Example:
```
Dim Adr As Byte
Dim Com As Byte

Enable Interrupts  'user must enable interrupts
Enable Ovf0        'user must enable Timer0 overflow interrupt


Do
    RC5(Adr, Com)
    Print Adr; " "; Com
Loop
```

Related topics:
$RC5

6.95. Randomize

Description:
Initialize Rnd generator

Syntax:
```
Randomize(seed)
```

Remarks:
seed is initial value for random generator, (numeric constant 0-255).

Rnd

6.96. ReadEE

Description:
Returns a value from internal EEPROM..

Syntax:
```
var=ReadEE(adr)
```

Remarks:
var holds a value previously stored in EEPROM at address adr.

Example:
```
WriteEE(i, i)    ' with counter (omit loc 0)
n=ReadEE(i)
```

Related topics:
WriteEE()
InitEE

6.97. Reset

Description:
Resets the Bit variable, variable.bit, Port.pin, WatchDog timer or External interrupts flags.

Syntax:
```
Reset BitVar
Reset Var.bit
Reset PORT.pin
Reset WatchDog
Reset Intx
```

Remarks:
Port pin must first be configured as an output.

Example:
```
Reset b           'b is Bit var
RESet n.2         'n is byte var

$Def Led=PORTB.3
Set DDRB.2     'configured for output

Reset PORTB.2   'PortB=0
Reset Led

Set PORTb.2
Set Led

Reset WatchDog 'resets WatchDog

Reset Int0     'resets Int0 flag
```

Related topics:
Set
Toggle

6.98. Return

Description:
Defines from Function returned value.

Syntax:
```
Return numeric expression
```

Remarks:
Return is from Function returned value (Byte, Integer, Word or Long)

Example:
```
Declare Function Mul(a As Byte, b As Byte) As Byte
Dim n As Byte

n=Mul(5, 7)      'n=35

'//////////////////////////////////////////////////////
```

```
Function Mul(a As Byte, b As Byte) As Byte

Return a*b

End Function
```

Related topics:
Declare
Function
Sub

6.99. Right

Description:
Return the rightmost n characters in a string.

Syntax:
```
var=Right(var1, n)
```

Remarks:
var string that right chars are assigned.
var1 source string (MUST be string var, not string constant).
n number of characters from the right.

Example:
```
Dim Name As String*15
Dim Part As String*10

Name="Mona Lisa"
Part=Right(Name, 4)   'Part="Lisa"
```

Related topics:
Left
Mid

6.100. Rnd

DDescription:
Returns a pseudo random number between 0 and 255 (type Byte).

Syntax:
```
var=Rnd()
```

Remarks:
var variable that receives the random number

Example:
```
Randomize(5)    'initialize Rnd generator
n=Rnd()
```

Related topics:
Randomize

6.101. Rotate

Description:
Rotate variable left or right n number of places.

Syntax:
```
Rotate(left|right, var1, var2)
var3=Rotate(left|right, var1, var2)
```

Remarks:
var1 is number of places to rotate
var2 is actual variable to be rotated
var3 is var to which rotated var2 is assigned

Example:
```
Rotate (Right, 1, n)    'rotates var n right one place
m=Rotate (Left, 4, n)   'rotates var n left four places and assign it to var m
```

Related topics:
Shift

6.102. Select

Description:
Selects a block of statements from a list, based on the value of an expression.

Syntax:
```
Select Case var
   Case val1
      statements
   Case val2 To val3
      statements
   Case <val4
      statements
   Case val5, val6, val7
      statements
   Case Else
      statements
End Select
```

Remarks:
var is a test variable (Byte, Integer or Word).
val1, val2, ... are different possible variable values.

If one "Case" matches, no subsequent "Cases" will be tested !

The code lenght under one Case is limited to 64 words!

Example1:

```
Select Case n
   Case 32
      Print "SPACE"
   Case 13
      Print "ENTER"
   Case 65
      Print "A"
   Case 49
      Print "1"
   Case 50
      Print "2"
   Case 120
      Print "X"
   Case Else
      Print "Miss!"
End Select
```

Example2:

```
Dim s As String*1
Select Case s
   Case "a", "A"              ' a or A
      Print "This is a or A"
   Case "b", "c", "d"         ' b, c or d
      Print "others"
   Case Else
      Print "Miss!"
End Select
```

Related topics:

If

6.103. Set

Description:

Sets the Bit variable, variable.bit or Port.pin.

Syntax:

```
Set BitVar
Set Var.bit
Set PORT.pin
```

Remarks:

Port pin must first be configured as an output.

Example:

```
Set b           'b is Bit var
Set n.2         'n is byte var
```

```
Set DDRB.2      'PORTB.2 is output
Set PORTB.2      'PORTB.2=1
Set Led          'sets PORT.bit defined as LED

Reset PORTB.2    'PORTB.2=0
Reset Led        'resets PORT.bit defined as LED
```

Related topics:
toggle
Reset

6.104. Shift

Description:
Shift var left or right n number of places.

Syntax:
```
Shift(left|right, var1, var2)
var3=Shift(left|right, var1, var2)
```

Remarks:
var1 is number of places to shift
var2 is actual variable to be shifted
var3 is var to which shifted var2 is assigned

Example:
```
Shift(Right, 1, n)     'shift var n right one place
m=Shift(Left, 4, n)    'shift var n left four places and assign it to var m
```

Related topics:
Rotate

6.105. ShiftOut

Description:
ShiftOut variable(s) on a PORTx.pin, usually to fill shift registers.

Syntax:
```
ShiftOut var1; var2;....
ShiftOut var1, n
```

var1, var2 vars to be shifted out on port.pin defined by $ShiftOut
n number of bytes to shift out

Remarks:
Very suitable for expanding output ports by adding shift registers like 74HC4094, TIC 2965 etc.

Example:
```
ShiftOut n, 10   'ShiftOut the whole array
ShiftOut i; w    'ShiftOut i and w
```

Related topics:
$ShiftOut
ShiftIn

6.106. ShiftIn

Description:
Shift IN variable on a Pinx.pin.

Syntax:
```
ShiftIn var1
```

`var` var to be shifted into from PINx.pin defined by `$ShiftOut data`

Example:
```
n=ShiftIn   'Load n from outher shift register
```

Related topics:
$ShiftOut
ShiftOut

6.107. Sin

Description:
Returns the trigonometric sine of its argument.

Syntax:
```
var=Sin(numeric expression)
```

Remarks:
`var` type Float receives a Sine of `numeric expression`
`numeric expression` can be in Radians or Deegrees, depending on $Angles metastatement!

Example:
```
Dim n As Byte
Dim f1 As Float

n=30          ' assuming $Angles=Degrees
f1=Sin(n)     ' f1=0.5000000
```

Related topics:
Cos

Tan
Asin
Acos
Atan

6.108. Sinh

Description:
Returns the Sine Hiperbolicus of its argument.

Syntax:
```
var=Sinh(numeric expression)
```

Remarks:
`var` type Float receives a Sinh of `numeric expression`
`numeric expression`

Example:
```
Dim f1 As Float

f1=Sinh(5)     'f1=74.20995
```

Related topics:
Cosh
Tanh

6.109. Sort

Description:
Sorts values in SRAM buffer.

Syntax:
```
Sort(SRAMbuff, length)
```

Remarks:
`SRAMbuff` is name of Buffer in SRAM (starting address in SRAM)
`length` is number of bytes to Sort

Example:
```
Dim DB(16) As Byte

Sort(DB, 16)     'sorts 16 bytes in array DB
```

6.110. Sound

Description:
Makes a sound.

Syntax:
```
Sound var1, var2
```

`var1 * 10` is half of perion in us, f = 1/(10 * var1 * 2) (frequency in Hz)
`var2 * 10` is number of periods in sound (duration)

Example:
```
Sound 50, 100    '1 kHz beep, 1 sec in duration
Sound 25, 100    '2 kHz beep, 0.5 sec in duration
```

Related topics:
$Sound

6.111. SpiIn

Description:
Receives a value from the SPI-bus in SLAVE mode.

Syntax:
```
var = SpiIn
```
var variable to receive data from the SPI bus

Remarks:
SS pin is input and must be driven by MASTER!

Example:
```
n=SpiIn
```

Related topics:
SPIOut

6.112. SpiOut

Description:
Sends the value of a variable to the SPI-bus in MASTER mode.

Syntax:
```
SpiOut var
SpiOut var1; var2;....,wait
SpiOut var1, n, wait
```

var, var1, var2 variables to be shifted out
n number of bytes from SRAM to send via SPI bus, starting with `var1`

Remarks:
SS pin is set to OutPut (user can use this pin to select SLAVE)!

Example:
```
SpiOut i              'ShiftOut i (9)
SpiOut n; 10, Wait    'ShiftOut the whole array
```

Related topics:
SPIIn

6.113. Sqr

Description:
Calculates Float Square of its argument.

Syntax:
`var=Sqr(numeric expression)`

Remarks:
`var` Float, receives a Square of `numeric expression`
`numeric expression`

Example:
```
Dim f1 As Float
Dim f2 As Float

f1=12.345
f2=Sqr(f1)     ' f2=f1*f1=152.399
```

Related topics:
Pow
Sqrt

6.114. Sqrt

Description:
Calculates Square root.

Syntax:
`var=Sqr(numeric expression)`

Remarks:
`var` receives a Square root of `numeric expression`
`numeric expression` must be positive

Example:
```
Dim n As Byte
Dim w As Word
```

```
w=12345
n=Sqrt(w)     'n=111
```

6.115. Start

Description:
Starts or enables one of the specified devices.

Syntax:
```
Start device
Start Adc [, Vref=Ext|Int|Vcc]
```

Remarks:
device can be:
- `Adc` supply for AD converter (default is stopped), user can specify type of used Vref!
- `Ac` supply for analog comparator (default is started)
- `WatchDog`
- `Timer0, Timer1, Timer2`

Example:
```
Start Ac             ' switch supply to Ac
Start Adc, Vref=Int  ' switch supply to Adc, Internal (2.56V) Vref is used
Start WatchDog       ' enables WatchDog
Start Timer1         ' start Timer1
```

Related topics:
Stop

6.116. Stop

Description:
Stops or disables one of the specified devices.

Syntax:
```
Stop device
```

Remarks:
device can be:
- `Adc` supply for AD converter (default is stopped)
- `Ac` supply for analog comparator (default is started)
- `WatchDog`
- `Timer0, Timer1, Timer2`

Example:

```
Stop Ac                ' switch supply from Ac
Stop Adc               ' switch supply from Adc
Stop WatchDog          ' disables WatchDog
Stop Timer1            ' stops Timer1
```

Related topics:
Start

6.117. Str

Description:
Converts a number to a string.

Syntax:
```
var=Str(numeric expression)
```

Remarks:
var string variable

If `$LeadChar` is defined then result will be right justified with Leading Char as defined. Also, if `Format()` is defined then optional decimal point will be inserted!

Example:
```
Dim n As Byte
Dim s As String*5

n=123
s=Str(n)    's="123"
```

Related topics:
Val

6.118. Sub

Description:
Defines a subroutine procedure.

Syntax:
```
Sub NameOfSub(parameters list)
```

Remarks:
NameOfSub is the name of the subroutine
parameters list is the name and type of parameters, comma delimited (can not be Bit)

Sub must first be declared using the Declare keyword (names of passing parameters must be the same as later in Sub).

Example:

```
Declare Sub Test(a As Byte, b As Byte) 'declares a Sub Test

Test(2, 6)     '22 will be Printed

End

'//////////////////////////////////////////////////
Sub Test(a As Byte, b As Byte)
Local d As Byte  ' local var is declared

d=10
[Exit Sub] ' optionally exit from Sub

Print a*b+d
End Sub      ' here is end of Sub
```

Related topics:
Declare
Function

6.119. Swap

Description:
Swaps variable(s), depending on type of variable.

Syntax:
```
Swap(var)
Swap(var1, var2)
```

Remarks:
var if var is `Byte` then nibles will be swaped, if var is `Word` or `Integer` then bytes will be swaped.
var1 this variable will be swaped with var2

Example:
```
Dim a As Byte, b As Byte
Dim w As Word

a=&h25
b=&h34
Swap(a)      ' a=&h52

w=&h1234
Swap(w)      ' w=&h3412

Swap(a, b)   ' a=&h34, b=&h25
```

6.120. Tan

Description:
Returns the trigonometric Tangent of its argument.

Syntax:
```
var=Tan(numeric expression)
```

Remarks:
`var` type Float receives a Tangent of `numeric expression`
`numeric expression` can be in Radians or Deegrees, depending on $Angles metastatement!

Example:
```
Dim n As Byte
Dim f1 As Float

n=30          ' assuming $Angles=Degrees
f1=Tan(n)     ' f1=0.5773503
```

Related topics:
Sin
Cos
Asin
Acos
Atan

6.121. Tanh

Description:
Returns the Tangent Hiperbolicus of its argument.

Syntax:
```
var=Tanh(numeric expression)
```

Remarks:
`var` type Float receives a Tanh of `numeric expression`
`numeric expression`

Example:
```
Dim f1 As Float

f1=Tanh(5)     'f1=0.999909
```

Related topics:
Sinh
Cosh

6.122. Toggle

Description:
Toggles the state of an AVR port pin.

Syntax:
```
Toggle PORT.pin
```

Remarks:
AVR port pin must first be configured as an output.

Example:
```
Toggle PORTB.2     'toggles PortB.2
Toggle Led         'toggles Port.Pin named Led (defined using $Def)
```

Related topics:
Set
Reset

6.123. UCase

Description:
Returns an all-uppercase version of its string argument.

Syntax:
```
var=UCase(var1)
```

Remarks:
var uppercase string version of var1
var1 original string variable.

Example:
```
Dim Name As String*15
Dim Part As String*10

Name="Mona Lisa"
Part=UCase(Name)    'Part="MONA LISA"
```

Related topics:
LCase

6.124. Val

Description:
Returns the numeric equivalent of a string.

Syntax:
```
var=Val(string)
```

Remarks:
var variable to store the string value.
`string` string variable

Example:
```
Dim s As String * 8
Dim n As Byte

s="123"    'init string
n=Val(s)   'n=123
```

Related topics:
Str

6.125. VarPtr

Description:
Returns the SRAM or XRAM address of a variable (pointer).

Syntax:
```
var1=VarPtr(var2)
```

Remarks:
var1 variable that will pointing to var2.
var2 variable to retrieve the address from.

Var1 mast be declared as `Word` for devices with more than 256 bytes of SRAM!

Example:
```
Dim n As Byte             'global byte variable named a
Dim w As Word             'global word variable named w
Dim f As Float            'global word variable named w
Dim db(10) As Byte        'global array of ten bytes named n
Dim s1 As String * 8      'global string variable named s1, length must be specified

adr=VarPtr(n)              'adr=&h60
adr=VarPtr(w)              'adr=&h61
adr=VarPtr(f)              'adr=&h63
adr=VarPtr(db)             'adr=&h67
adr=VarPtr(s1)             'adr=&h71
```

6.126. Wait, Waitms, Waitus

Description:
Waits seconds, milliseconds or microseconds*10.

Syntax:
`Wait` var - waits var seconds
`WaitMs` var - waits var milliseconds

`WaitUs` var - waits var microseconds*10

Remarks:
var - `Byte` var type

`Wait`, `WaitMs` and `WaitUs` are not very precise, especially `WaitUs` at lower values!

All enabled Interrupts are active during Waiting!

Example:
```
Wait 2       ' waits 2seconds
WaitMs 25    ' waits 25ms
WaitUs 3     ' wait 30us
```

6.127. Wend

While

6.128. While

Description:
Executes a series of statements as long as a given condition is True.

Syntax:
```
While integer expression
      statements
      Exit While      'you can EXIT from the loop at any time
Wend
```

Remarks:
condition is a boolean expression that evaluates to True or False.

If condition is True, all statements are executed until the Wend statement is encountered. Control then returns to the While statement and the condition is checked again. If condition is still True, the process is repeated, otherwise execution resumes with the statement following the Wend statement.

Example:
```
While i<6          ' for all ADC inputs
   Print Adc8(i)
   Incr i
Wend
```

Example:
Do-Loop
For-Next

6.129. WriteEE

Description:
Writes a value into internal EEPROM at location adr.

Syntax:
```
WriteEE(adr, var [, var1, var2,...varn])
```

Remarks:
adr the address in EEPROM that var will be stored at. (adr can be a constant, variable or expression)
var must be variable or const to be stored in EEPROM at address adr.
var1-n can be expressions or constants to initialize EEPROM starting at address adr.(must be Bytes)

Example:
ReadEE

See also:
ReadEE
InitEE

Index

Notes: