TwinCAT
TwinCAT ST Specs and Language Description

Updated Aug 12, 2010 by lucabell...@gmail.com

TwinCAT ST (Structured Text) Language

The structured text consists of a series of instructions which, as determined in high level languages, ("IF..THEN..ELSE") or in loops (WHILE..DO) can be executed.

Expressions

Construction returning a value after its evaluation. Composed of **operators** and **operands**. An **operand** can be:

- A constant
- A variable
- A function Call
- An expression

```
expression = ( constant | variable | functioncall ),
             ( constant | variable | functioncall ) operator expression;
```

Binding strength in expressions:

1. Parentheses `(expression)`
2. Function call `Function name (parameter list)`
3. Exponentiation `EXPT`
4. Negate `NOT`
5. Multiply,Divide,Modulo `*,/,MOD`
6. Add,Subtract `+,-`
7. Compare `<,>,<=,>=`
8. Equal to,Not Equal to `=,<>`
9. Boolean AND `AND`
10. Boolean XOR `XOR`
11. Boolean OR `OR`

Instructions

There are several kind of instructions:

- Assignment

```
A:=B; CV := CV + 1; C:=SIN(X);
```

- Calling a function block and use of the FB version

```
CMD_TMR(IN := %IX5, PT := 300);
A:=CMD_TMR.Q;
```

- Return from function

```
RETURN;
```

- If statement

```
IF D<0.0
THEN C:=A;
ELSIF D=0.0
THEN C:=B;
ELSE C:=D;
END_IF;
```

- Case statement

```
CASE INT1 OF
1: BOOL1 := TRUE;
2: BOOL2 := TRUE;
ELSE
BOOL1 := FALSE;
BOOL2 := FALSE;
END_CASE;
```

- For statement

```
FOR I:=1 TO 100 BY 2 DO
   IF ARR[I] = 70
      THEN J:=I;
      EXIT;
   END_IF;
END_FOR;
```

- While statement

```
WHILE J<= 100 AND ARR[J] <> 70 DO
   J:=J+2;
END_WHILE;
```

- Repeat Until statement

```
REPEAT J:=J+2;
UNTIL J= 101 OR ARR[J] = 70
END_REPEAT;
```

- Exit statement

```
EXIT;
```

- Empty instruction

```
;
```

Data Types

There are 2 groups of data types:

- Standard Data Types
- User Defined Data Types

Standard Data Types

- BOOL {TRUE, FALSE}
- BYTE {0 to 255}
- WORD {0 to 65535}
- DWORD {0 to 4294967295}
- SINT {-128 to 127}
- USINT {0 to 255}
- INT {-32768 to 32767}
- UINT {0 to 65535}
- DINT {-2147483648 to 2147483647}
- UDINT {0 to 4294967295}
- LINT(n.s.)
- ULINT(n.s.)
- REAL {~ -3.402823 x 10^38 to ~ 3.402823 x 1038}
- LREAL {~ -1.79769313486231E308 to ~ 1.79769313486232E308}
- STRING {(dim) 'This is a String';}
- TIME {T#0ms to T#71582m47s295ms} - T#9d8h7m6s5ms
- TIME_OF_DAY (TOD) {TOD#00:00 to TOD#1193:02:47.295} - TOD#00:00:00.001
- DATE {D#1970-01-01 to D#2106-02-06} - D#1972-03-29
- DATE_AND_TIME (DT) {DT#1970-01-01-00:00 to DT#2106-02-06-06:28:15}
- MEMORY LOCATION {rMyVar1 AT %MW0 : REAL;}

A little example about constants:

- 14 (Decimal Number)
- 2#1001_0011 (Binary Number)
- 8#67 (Octal Number)

- 16#A (Hexadecimal Number)
- 7.4 instead of 7,4
- 1.64e+009 instead of 1,64e+009
- $$ Dollar signs
- $' Single quotation mark
- $L or $l Line feed
- $N or $n New line
- $P or $p Page feed
- $R or $r Line break
- $T or $t Tab

In variables defined as SINT, INT, DINT, USINT, UINT, UDINT, BYTE, WORD, DWORD, a single bit can be addressed using dot notation. Not for VAR_IN_OUT variables.

User Defined Data Types

- ARRAY (up to 3 dimensional)

```
<Field_Name>:ARRAY [<LowLim1>..<UpLim1>, *<LowLim2>..<UpLim2>, *<LowLim3>..<UpLim3>] OF <elem. Type>
```

- POINTER (ADR() and SIZEOF() functions are needed; dereferenced by adding the content operator "^" after the pointer identifier)

```
<Identifier>: POINTER TO <Datatype/Functionblock>;
```

- ENUM

```
TYPE <Identifier>:(<Enum_0> ,<Enum_1>, ...,<Enum_m> *<assign>);END_TYPE
```

- STRUCT (referenced using Structurename.Componentname)

```
TYPE <Structurename>:
STRUCT
    <Declaration of Variables 1>
    .
    .
    <Declaration of Variables n>
END_STRUCT
END_TYPE
```

- ALIAS

```
TYPE <Identifier>: <Assignment term>;
END_TYPE
```

- SUB-RANGE DATA TYPES (for defining a sub-range of a standard data type)

```
TYPE <Name> : <Inttype> (<ug>..<og>) END_TYPE;
```

Addresses

Individual memory locations can be displayed using the following syntax:

```
%{I,Q,M}{X,B,W,D,*}{location}
```

Where:

- I: Input
- Q: Output
- M: Memory Location
- X: Single Bit - %QX75.1 (*Bit 1 of output byte 75*)
- B: Byte - %QB7 (*Output byte 7*)
- W: Word - %IW215 (*Input word 215*)
- D: Double Word - %MD48 (*Double word in memory position 48 in the memory location*)
- *: Config Variable

TwinCAT PLC Library and Operators

```
'                              String delimiters (e.g. 'string1')
[..]                           Size of Array range (e.g. ARRAY[0..3] OF INT)
:                              Delimiter between Operand and Typ in a declaration
;                              Termination of instruction (e.g. a:=var1;)
```

```
^                              Derefereced Pointer (e.g. pointer1^)
:=                             ST var1 N Store actual result to var1
<Programmname>                 Call program prog1
<Instanzname>                  Call function block instance inst1
<Fktname>(vx,vy,..)            Call function fctname and transmit variables vx, vy
RETURN                         Leave POU and go back to caller
(                              The value following the bracket is handled as operand, the operation before the bracke
)                              Now execute the operation which has been set back
AND                            Bitwise AND
OR                             Bitwise  OR
XOR                            Bitwise  exclusive OR
NOT                            Bitwise  NOT
+                              Addition
-                              Subtraction
*                              Multiplication
/                              Division
>                              Greater than
>=                             Greater or equal
=                              Equal
<                              Less than
<>                             Not equal
<=                             Less or equal
in1 MOD in2                    Modulo Division
INDEXOF(in)                    Internal index of POU in1; [INT]
SIZEOF(in)                     Number of bytes required for the given data type of in
SHL(in,K)                      Bitwise left-shift of operator in by K
SHR(in,K)                      Bitwise right-shift of operator in by K
ROL(in,K)                      Bitwise rotation to the left of operator in by K
ROR(in,K)                      Bitwise rotation to the right of operator in by K
SEL(G,in0,in1)                 Binary selection between 2 operands in0 (G is FALSE) and in1 (G is TRUE)
MAX(in0,in1)                   Returns the greater of 2 values
MIN(in0,in1)                   Returns the lesser of 2 values in0 and in1
LIMIT(Min,in,Max)              Limits the value range
MUX(K, in0,.. in_n)            Selects the Kth value out of a group of values
ADR(in)                        Address of the operand in [DWORD]
BOOL_TO_<type>(in)             Type conv. of the boolean operand
<type>_TO_BOOL(in)             Type conv. to BOOL
INT_TO_<type>(in)              Type conv. of an INT Operand to another elementary type
REAL_TO_<type>(in)             Type conv. of an REAL operand to another elementary type
LREAL_TO_<type>(in)            Type conv. of a LREAL operand to another elementary type
TIME_TO_<type>(in)             Type conv. of a TIME operand to another elementary type
TOD_TO_<type>(in)              Type conv. of a TOD operand to another elementary type
DATE_TO_<type>(in)             Type conv. of a DATE operand to another elementary type
DT_TO_<type>(in)               Type conv. of a DT operand to another elementary type
STRING_TO_<type>(in)           Type conv. of a STRING operand to another elementary type
TRUNC(in)                      Conversion from REAL to INT
ABS(in)                        Absolute value of operand in
SQRT(in)                       Square root of operand in
LN(in)                         Natural logarithm of operand in
LOG(in)                        Logarithm of operand in, base 10
EXP(in)                        Exponential function of operand in
SIN(in)                        Sine of operand in
COS(IN)                        Cosine of operand in
TAN(in)                        Tangent of operand in
ASIN(in)                       Arc sine of operand in
ACOS(in)                       Arc cosine of operand in
ATAN(in)                       Arc tangent of operand in
EXPT(in,expt)                  Exponentation of operand in with expt
LEN(in)                        String length of operand in
LEFT(str, size)                Left inital string of given size of string str
RIGHT(str, size)               Right initial string of given size of string str
MID(str, len, pos)             Partial string of str of given length
CONCAT(str1, str2)             Concatenation of two subsequent strings standard.lib
INSERT(str1, str2, pos)        Insert string str1 in String str2 at position pos
DELETE(str1, len, pos)         Delete partial string (length len), start at pos of str1
REPLACE(str1,str2,len,pos)     Repl. partial string of len by str2, start at pos of str1
FIND(str1, str2)               Search for partial string str2 in str1
SR                             Bistable FB is set dominant
RS                             Bistable FB is set back
SEMA                           FB: Software Semaphor (interruptable)
R_TRIG                         FB: rising edge is detected
F_TRIG                         FB: falling edge is detected
```

```
CTU                             FB: Counts up
CTD                             FB: Counts down
CTUD                            FB: Counts up and down
TP                              FB: trigger
TON                             FB: on-delay timer
TOF                             FB: off-delay timer
```

System Functions

- CheckBounds
- CheckDivByte
- CheckDivReal
- CheckDivWord
- CheckDivDWord
- CheckRangeSigned
- CheckRangeUnsigned

Program Blocks

There are 3 kinds of blocks:

- `FUNCTION_BLOCK FunctionBlockName`, represents a function which returns no data; a call to a FUNCTION_BLOCK needs explicit name of arguments (e.g. `FunctionBlockName(arg1:=1,arg2:=2)`); in order to use a FUNCTION_BLOCK, it is necessary to get an instance, declaring it as variable
- `FUNCTION FunctionName : ReturnDataType`, represents a function which returns a value; in a call to a FUNCTION, name of arguments are implicit (e.g. `Function(1,2)`)
- `PROGRAM ProgramName`, represents the main program

On the top of each one of these blocks there's the variable section. They can be:

- `VAR_INPUT [...] END_VAR`, contains all variables that will be shown as Input lines in FBD
- `VAR_OUTPUT [...] END_VAR`, contains all variables that will be shown as Output lines in FBD
- `VAR_IN_OUT [...] END_VAR`, contains all variables that will be shown as Input-Output lines in FBD
- `VAR [...] END_VAR`, contains all variables that will be shown as Local variables, so cannot be accessed in FBD

Global variables, global network variables (VAR_GLOBAL), and variable configurations (VAR_CONFIG) must be defined in separate objects. Variable sections can have a modifier such as RETAIN, PERSISTENT, CONSTANT.

Notes on source code

There's no way to create TwinCAT project source files from 3rd party applications. Seems that no API exists for TwinCAT.