DOCUMENT
Structured Text programming manual
LADV-PM-ST

PROJECT
Wartens LADV virtual PLCs
Rev 1.0

Structured Text programming manual

IEC 61131-3 Structured Text on the Wartens LADV controllers: how a program is laid out, the data types, the statements, tasks and I/O, with listings that compile as printed.

Document	Structured Text programming manual
Code	LADV-PM-ST
Revision	1.0, 1 November 2026
Firmware	V1.0.0
Issued by	Wartens, makers of LADX

> Wartens LADV controllers are virtual PLCs that run in a web browser, for learning, for testing logic and for preparing for commissioning. They are not real time, not safety rated, and they cannot control equipment.

1. A first program

A Structured Text source holds declarations and program organisation units (POUs) in any order. The smallest useful program declares its I/O at direct addresses and has one PROGRAM:

```
VAR_GLOBAL
  Start_PB AT %IX0.0 : BOOL;
  Stop_PB  AT %IX0.1 : BOOL; (* normally closed: TRUE when healthy *)
  Motor    AT %QX0.0 : BOOL;
END_VAR

PROGRAM Main
  Motor := (Start_PB OR Motor) AND Stop_PB;
END_PROGRAM
```

With no CONFIGURATION, every PROGRAM runs in one cyclic task at the program's scan time. Identifiers are not case sensitive. Comments are written (* like this *), /* like this */ or // to the end of the line.

2. Elementary data types

Type	Size	Range
SINT	8 bits, signed	-128 to 127
INT	16 bits, signed	-32,768 to 32,767
DINT	32 bits, signed	-2,147,483,648 to 2,147,483,647
LINT	64 bits, signed	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807, exact to 53 bits
USINT	8 bits, unsigned	0 to 255
UINT	16 bits, unsigned	0 to 65,535
UDINT	32 bits, unsigned	0 to 4,294,967,295
ULINT	64 bits, unsigned	0 to 18,446,744,073,709,551,615, exact to 53 bits
BYTE	8 bits, bit string	16#0 to 16#FF
WORD	16 bits, bit string	16#0 to 16#FFFF

Type	Size	Range
DWORD	32 bits, bit string	16#0 to 16#FFFFFFFF
LWORD	64 bits, bit string	16#0 to 16#FFFFFFFFFFFFFFFF, exact to 53 bits

Type	Meaning
BOOL	TRUE or FALSE
REAL, LREAL	IEC 60559 single and double precision floating point
TIME, LTIME	Durations: T#1h2m3s, T#250ms, LTIME#1.5us
DATE, TOD, DT	Dates and times of day: D#2026-11-01, TOD#08:30:00, DT#2026-11-01-08:30:00
STRING, WSTRING	Text, 80 characters unless declared STRING[n]
CHAR, WCHAR	One character

Integer literals can be written in base 2, 8 or 16 (2#1010, 8#17, 16#FF) and given a type (INT#5, BYTE#16#81). Integer arithmetic wraps at the width of the type, as hardware does: an INT holding 32767 plus 1 is -32768.

3. Derived types

Enumerations, subranges, arrays and structures are declared in a TYPE block. Structures and arrays can be nested and initialised.

```
TYPE
  Mode : (Idle, Filling, Draining);
  Percent : INT (0..1000);
  Recipe : STRUCT
    name : STRING[20] := 'Default';
    setpoint : Percent := 600;
    soak : TIME := T#30s;
  END_STRUCT;
END_TYPE

PROGRAM Main
VAR
  mode : Mode := Idle;
  recipes : ARRAY[1..3] OF Recipe;
  active : INT := 1;
  target : Percent;
END_VAR
  target := recipes[active].setpoint;
  IF mode = Idle AND target > 0 THEN
    mode := Filling;
  END_IF;
END_PROGRAM
```

4. Variables

Section	Where	Meaning
VAR	Any POU	Local memory, kept between calls of a PROGRAM or FUNCTION_BLOCK
VAR_INPUT	FUNCTION, FUNCTION_BLOCK	Passed in by value
VAR_OUTPUT	FUNCTION_BLOCK	Read after the call as instance.name, or with =>
VAR_IN_OUT	FUNCTION, FUNCTION_BLOCK	Passed by reference
VAR_TEMP	Any POU	Scratch memory, initialised on every call
VAR_GLOBAL	Outside POUs	Global memory and I/O, visible everywhere

RETAIN keeps a variable through a warm restart on a retentive CPU. CONSTANT makes it read-only. AT gives a direct address: %IX and %QX for digital channels, %IW and %QW for analog, %M for internal memory.

5. Expressions

Operators from highest to lowest precedence: parentheses and calls; ** (power); unary - and NOT; * / MOD; + -; < > <= >=; = <>; AND and &; XOR; OR.

Implicit conversion only widens (INT to DINT, REAL to LREAL). Narrowing is a warning and needs a conversion function such as DINT_TO_INT to be explicit. A REAL is never stored in an integer without REAL_TO_INT or TRUNC. Comparing REAL values for equality is a warning, since rounding makes it unreliable.

6. Statements

Statement	Form
Assignment	x := expression;
IF	IF c THEN ... ELSIF c THEN ... ELSE ... END_IF;
CASE	CASE n OF 1: ...; 2, 3: ...; 10..20: ...; ELSE ... END_CASE;
FOR	FOR i := 1 TO 10 BY 1 DO ... END_FOR;
WHILE	WHILE c DO ... END_WHILE;
REPEAT	REPEAT ... UNTIL c END_REPEAT;
EXIT, CONTINUE	Leave a loop, or start its next pass
RETURN	Leave the POU
Block call	timer(IN := start, PT := T#5s); or with outputs: fb(a := 1, q => result);

7. Functions and function blocks

A FUNCTION returns a value and keeps no memory. A FUNCTION_BLOCK keeps its memory between calls and is used through an instance. Recursion is refused at compile time, and calls nest at most 32 deep.

```
FUNCTION Scale : REAL
VAR_INPUT
  raw : INT;
  lo, hi : REAL;
END_VAR
  Scale := lo + (hi - lo) * INT_TO_REAL(raw) / 1000.0;
END_FUNCTION

FUNCTION_BLOCK Debounce
VAR_INPUT
  raw : BOOL;
  settle : TIME := T#50ms;
END_VAR
VAR_OUTPUT
  clean : BOOL;
END_VAR
VAR
  on_delay : TON;
  off_delay : TON;
END_VAR
  on_delay(IN := raw, PT := settle);
  off_delay(IN := NOT raw, PT := settle);
  IF on_delay.Q THEN
    clean := TRUE;
  ELSIF off_delay.Q THEN
    clean := FALSE;
  END_IF;
END_FUNCTION_BLOCK

PROGRAM Main
VAR
  pressure : REAL;
  door : Debounce;
```

```
  door_shut : BOOL;
  raw_pressure : INT := 500;
  door_switch : BOOL;
END_VAR
  pressure := Scale(raw := raw_pressure, lo := 0.0, hi := 10.0);
  door(raw := door_switch, clean => door_shut);
END_PROGRAM
```

The standard functions and function blocks, from timers to string handling, are listed in LADV-IR-ST.

8. Tasks

A CONFIGURATION assigns programs to tasks. A cyclic task has an INTERVAL; an event task has SINGLE, and runs once on each rising edge of a global BOOL. At each scan the tasks that are due run in priority order, 0 first. Tasks do not preempt one another. A task faster than the CPU's fastest scan is refused at download.

```
VAR_GLOBAL
  Batch_Start AT %IX0.0 : BOOL;
  Heater      AT %QX0.0 : BOOL;
  Temp        AT %IW0   : INT;
  ticks : UDINT;
END_VAR

PROGRAM Control
  Heater := Temp < 650;
END_PROGRAM

PROGRAM Housekeeping
  ticks := ticks + 1;
END_PROGRAM

PROGRAM OnStart
VAR RETAIN
  batches : UDINT;
END_VAR
  batches := batches + 1;
END_PROGRAM

CONFIGURATION Cell
  RESOURCE Cpu ON LADV
    TASK Fast (INTERVAL := T#20ms, PRIORITY := 1);
    TASK Slow (INTERVAL := T#500ms, PRIORITY := 5);
    TASK Event (SINGLE := Batch_Start, PRIORITY := 0);
    PROGRAM Loop WITH Fast : Control;
    PROGRAM Stats WITH Slow : Housekeeping;
    PROGRAM Count WITH Event : OnStart;
  END_RESOURCE
END_CONFIGURATION
```

9. System variables

Name	Meaning
FirstScan	TRUE during the first scan after a start, FALSE after
AlwaysTRUE, AlwaysFALSE	Constants
Clock_10Hz, Clock_5Hz, Clock_2Hz, Clock_1Hz, Clock_0_5Hz	Square waves at those frequencies

10. Runtime errors

An error the compiler cannot see faults the controller, with a code and the line:

Code	Cause
F300	Watchdog: the scan took too long, or a loop never ended
F310	Integer division or MOD by zero

Code	Cause
F311	Array index outside the declared range
F312	MUX selector outside its inputs
F313	BCD_TO_INT given a value that is not BCD
F314	Calls nested more than 32 deep

11. Not supported

- Object-oriented extensions: CLASS, METHOD, INTERFACE, PROPERTY.
- References and pointers: REF_TO, ^, POINTER TO.
- SFC elements in Structured Text source: STEP, TRANSITION, ACTION.
- NAMESPACE, and vendor libraries and system functions.

Each of these is refused with a message that names it, not a parse error.

12. Worked examples

The reference solutions for the lab's twins. Each passes every objective of its twin.

Motor start/stop station

Start runs the motor and it keeps running when Start is released. Stop and E-Stop both stop it, and while E-Stop is pressed Start must do nothing. Light the Running lamp whenever the motor runs. Both stop buttons are wired normally closed, so they read 1 when nobody is pressing them.

```
(* Motor start/stop station: reference solution in IEC 61131-3 Structured Text. *)

VAR_GLOBAL
  Start_PB AT %IX0.0  : BOOL; (* Start pushbutton, normally open *)
  Stop_PB  AT %IX0.1  : BOOL; (* Stop pushbutton, normally closed, 1 when healthy *)
  EStop_OK AT %IX0.2  : BOOL; (* Emergency stop, normally closed, 1 when healthy *)
  Motor    AT %QX0.0  : BOOL; (* Motor contactor *)
  Run_Lamp AT %QX0.1  : BOOL; (* Running indicator *)
END_VAR

PROGRAM Main
  (* Start with a seal-in. Stop_PB and EStop_OK are normally closed, *)
  (* TRUE while healthy, so a broken wire stops the motor too.       *)
  Motor := (Start_PB OR Motor) AND Stop_PB AND EStop_OK;
  Run_Lamp := Motor;
END_PROGRAM
```

Traffic light junction

Run the junction: north-south green, then amber, then red while east-west goes green, amber, red, and round again. Never show two greens, always show amber between green and red, never leave a head dark, and give each direction at least four seconds of green.

```
(* Traffic light junction: reference solution in IEC 61131-3 Structured Text. *)

VAR_GLOBAL
  NS_Red   AT %QX0.0  : BOOL; (* North-south red lamp *)
  NS_Amber AT %QX0.1  : BOOL; (* North-south amber lamp *)
  NS_Green AT %QX0.2  : BOOL; (* North-south green lamp *)
  EW_Red   AT %QX0.3  : BOOL; (* East-west red lamp *)
  EW_Amber AT %QX0.4  : BOOL; (* East-west amber lamp *)
  EW_Green AT %QX0.5  : BOOL; (* East-west green lamp *)
END_VAR

TYPE Stage : (NS_GO, NS_WAIT, EW_GO, EW_WAIT); END_TYPE

PROGRAM Main
VAR
  stage : Stage := NS_GO;
  timer : TON;
```

```
  hold : TIME;
END_VAR
  (* How long each phase lasts. *)
  CASE stage OF
    NS_GO, EW_GO: hold := T#5s;
    NS_WAIT, EW_WAIT: hold := T#2s;
  END_CASE;

  timer(IN := TRUE, PT := hold);
  IF timer.Q THEN
    timer(IN := FALSE);
    CASE stage OF
      NS_GO: stage := NS_WAIT;
      NS_WAIT: stage := EW_GO;
      EW_GO: stage := EW_WAIT;
      EW_WAIT: stage := NS_GO;
    END_CASE;
  END_IF;

  (* One lamp per head, decided only by the stage, so two greens cannot happen. *)
  NS_Green := stage = NS_GO;
  NS_Amber := stage = NS_WAIT;
  NS_Red := stage = EW_GO OR stage = EW_WAIT;
  EW_Green := stage = EW_GO;
  EW_Amber := stage = EW_WAIT;
  EW_Red := stage = NS_GO OR stage = NS_WAIT;
END_PROGRAM
```

Sorting conveyor

Start and Stop run the belt, with a seal-in. Parts arrive on their own. At the pusher station a photo-eye sees every part and an inductive sensor sees only metal ones: push the metal parts into the reject bin and let the plastic parts ride to the end. Ten in a row with no mistakes passes.

```
(* Sorting conveyor: reference solution in IEC 61131-3 Structured Text. *)

VAR_GLOBAL
  Start_PB     AT %IX0.0  : BOOL; (* Start pushbutton, normally open *)
  Stop_PB      AT %IX0.1  : BOOL; (* Stop pushbutton, normally closed, 1 when healthy *)
  PE_Entry     AT %IX0.2  : BOOL; (* Photo-eye at the feeder, 1 when a part is there *)
  PE_Pusher    AT %IX0.3  : BOOL; (* Photo-eye at the pusher station, sees every part *)
  Metal_Sensor AT %IX0.4  : BOOL; (* Inductive sensor at the pusher station, sees metal parts only *)
  Belt         AT %QX0.0  : BOOL; (* Conveyor motor contactor *)
  Pusher       AT %QX0.1  : BOOL; (* Reject pusher solenoid valve, 1 extends *)
END_VAR

PROGRAM Main
VAR
  arrived : R_TRIG;
  parts : UDINT; (* Parts seen at the entry since the last cold restart. *)
END_VAR
  Belt := (Start_PB OR Belt) AND Stop_PB;

  (* A metal part at the pusher station is pushed off. *)
  Pusher := PE_Pusher AND Metal_Sensor;

  arrived(CLK := PE_Entry);
  IF arrived.Q THEN
    parts := parts + 1;
  END_IF;
END_PROGRAM
```

Tank level control

The process draws water all the time. Open the fill valve when the level falls below 40 percent and close it above 80, so it never overflows and never runs dry. Copy the level to the panel meter. When the Auto selector is switched off, close the valve and keep it closed. Level reads 0 to 1000 for 0 to 100 percent.

```
(* Tank level control: reference solution in IEC 61131-3 Structured Text. *)

VAR_GLOBAL
  Auto_Mode   AT %IX0.0  : BOOL; (* Auto/manual selector, 1 is auto *)
  Level       AT %IW0    : INT; (* Level transmitter, 0 to 1000 is 0 to 100 percent *)
  LSH         AT %IX0.1  : BOOL; (* High level switch, 1 above 90 percent *)
  LSL         AT %IX0.2  : BOOL; (* Low level switch, 1 below 10 percent *)
```

```
  Fill_Valve  AT %QX0.0  : BOOL; (* Fill valve solenoid, 1 opens *)
  High_Alarm  AT %QX0.1  : BOOL; (* High level alarm lamp *)
  Level_Meter AT %QW0    : INT; (* Panel meter, 0 to 1000 *)
END_VAR

PROGRAM Main
  (* Level reads 0 to 1000 for 0 to 100 percent. Fill from 40 percent up to 80, *)
  (* and never past the high switch or out of auto.                              *)
  IF Auto_Mode AND Level < 400 THEN
    Fill_Valve := TRUE;
  END_IF;
  IF Level > 800 OR LSH OR NOT Auto_Mode THEN
    Fill_Valve := FALSE;
  END_IF;

  Level_Meter := LIMIT(0, Level, 1000);
  High_Alarm := LSH;
END_PROGRAM
```