


Declaration

CoDeSys

Smart Software Solutions

We software Automation.

Declaration

Announce the existence of your elements to the compiler.

Smart Software Solutions

We software Automation.

Introduction

After this module you will know how

- to organize everything from simple data structures up to complex elements in CoDeSys.
- to protect data against reboot and download.

Agenda

- elementary data types
- variables
- user defined data types (DUT)
- RETAIN and PERSISTENT

Why data types?

0100 0001

$2^0 + 2^6 = 65$
Codification : decimal

41 H
Codification : hexadecimal

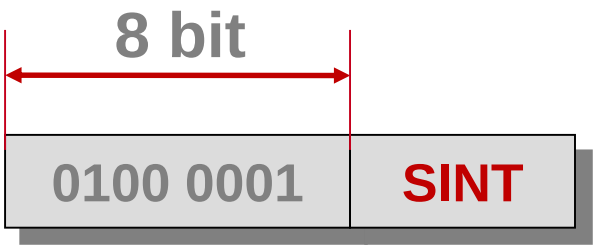
Input 1 and 7 high
Codification : binary

What does this mean?

0100 0001 SINT $2^0 + 2^6 = 65$

Why data types?

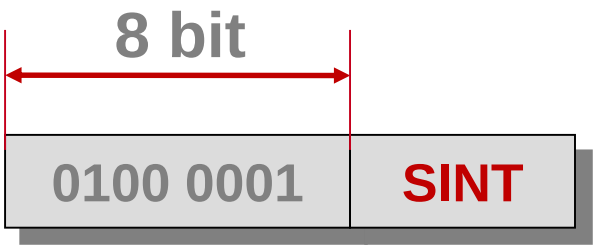
- The compiler must know…
  - What is the data size?
  - What kind of data is it?
  - What can we do with it?



$$2^0 + 2^6 = 65$$

How to declare a variable?

- We need
  - a variable name
  - a colon
  - a data type
  - an optional an initial value
  - a semicolon
  - an optional comment




```
siMyVariable    : SINT  := 65            ;  (* dec  *)
                        := 16#41            (* hex  *)
                        := 2#0100_0001      (* dual *)
```

Use variables according to the data type

- BINARY to make logical operation

data type	*lower limit*	*upper limit*	*information conten*	*prefix*
BOOL	FALSE	TRUE	1 bit (but 1 byte in memory)	x
				b
BYTE	0	255	8 bit	by
WORD	0	65535	16 bit	w
DWORD	0	4294967295	32 bit	dw
LWORD	0	$0 .. 2^{64}-1$	64 bit	lw

Use variables according to the data type

- NUMBERS to do arithmetical operation

data type	*lower limit*	*upper limit*	*information content*	*Prefix*
SINT	-128	127	8 bit	si
USINT	0	255	8 bit	usi
INT	-32.768	32.767	16 bit	i
UINT	0	65.535	16 bit	ui
DINT	-2.147.483.648	2.147.483.647	32 bit	di
UDINT	0	4.294.967.295	32 bit	udi
LINT	-263	$2^{63} - 1$	64 bit	li
ULINT	0	$2^{64} - 1$	64 bit	uli
REAL			32 bit	r
LREAL			64 bit	lr

Use variables according to the data type

- What can I do?

	DATA TYPE			
	BINARY		NUMBERS	
operation	logic	AND ANDN OR ORN XOR XORN	arithmetic	ADD DIV MUL SUB
			comparison	EQ NE GE GT LE LT

What about variables of different types?

- Operations can only be done with variables of the same data type.
- However, sometimes we have operands of different data types.
- So we use data type conversions.

Source		Destination
BOOL	→	INT

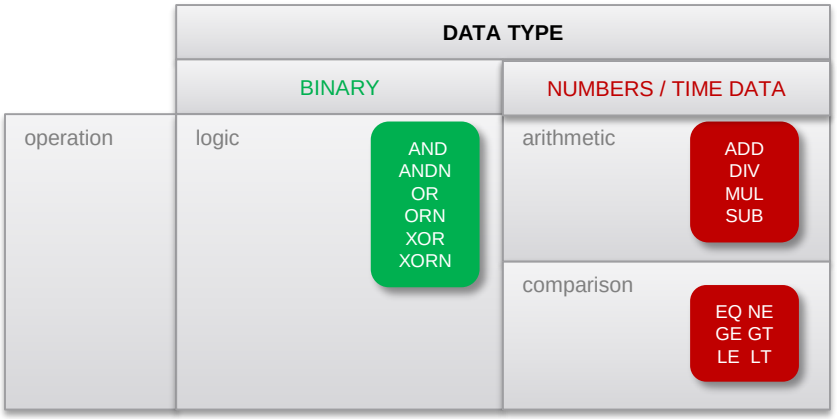
BOOL_TO_INT

More element data types

- Time data types

data type	*lower limit*	*upper limit*	*information content*	*Prefix*
TIME	0ms	1193h2m47s295ms	32 bit	tim
LTIME	0ns	213503d23h34m33s709ms551us615ns	64 bit	ltim
TIME_OF_DAY	00:00:00	23:59:59	32 bit	tod
DATE	01.01.1970	to approx. 06.02.2106	32 bit	dt
DATE_AND_TIME			64 bit	date

Usage of operation similar to numbers



More element data types

- In CoDeSys V3 there are two kinds of strings

STRING	**1 BYTE for each character**	**+**	**1 BYTE more in total**

- UNICODE

WSTRING	**2 BYTE for each character**	**+**	**2 BYTE more in total**

- Example

```
sText  : STRING   := 'CoDeSys' ;
wsText : WSTRING  := "CoDeSys" ;
```

STRING / WSTRING

- Limit the string length to save memory

```
sText   : STRING (10)   := 'CoDeSys'   ; (* 11 BYTE *)
wsText  : WSTRING (10)  := "CoDeSys"   ; (* 22 BYTE *)
```

without any limitation
=> default is 80 character

- STRING functions can only process strings of 1-255 characters!
- WSTRING functions are stored in the Standard64 library

Syntax rules for variables

- letters and numbers
- must start with a letter
- only single underscores
- no spaces
- no IEC keywords/ operands, also: +,-,*,/,…
- case insensitive
- no length limitation

```
xSecurityDoor1Open : BOOL;    (* TRUE ^= Door is open *)
```

Syntax rules for variables

- base name
  - meaningful, preferably short, English name
  - first letter uppercase (example: FileSize)
  - Prefix (lowercase) corresponding to data type
  - information of Boolean state

xSecurityDoor1Open: BOOL;

Prefix

Base name

(* TRUE ^= Door is open *)

Syntax rules for variables

- CONSTANT - variables
  - have to start with the constant-prefix c and an underscore

```
VAR CONSTANT
    c_uiSyncID : UINT := 16#80;
END_VAR
```

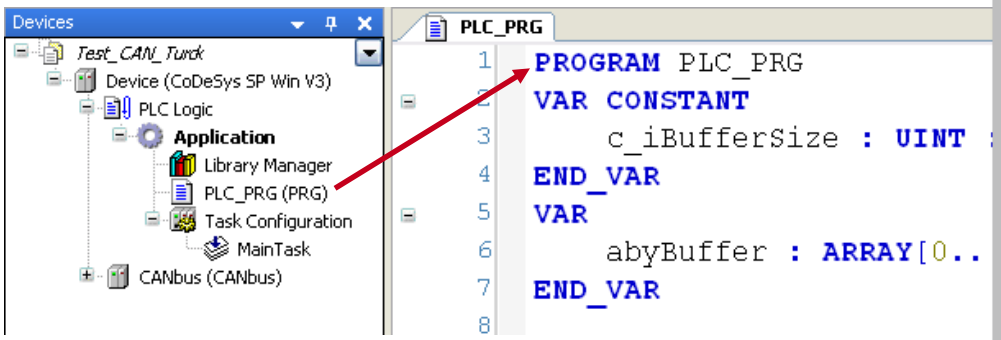
Syntax rules for variables

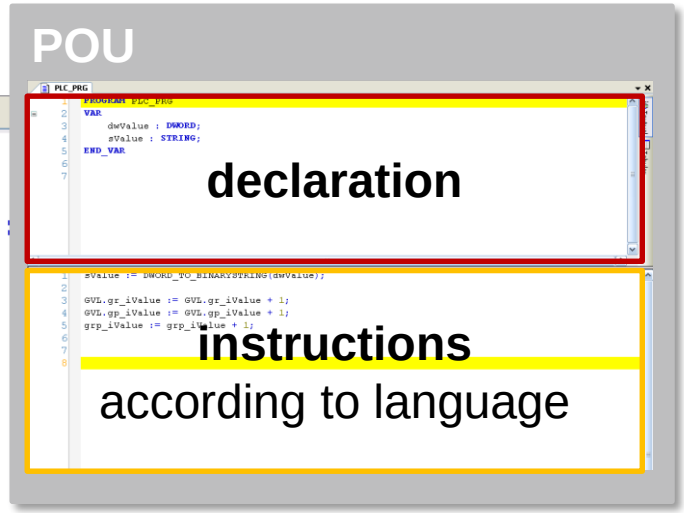
- GLOBAL Variables (g) and Global Constants (gc)
  - have an additional prefix:

```
VAR_GLOBAL CONSTANT
    gc_iBufferSize : INT := 100;
END_VAR
VAR_GLOBAL
    g_abyBuffer : ARRAY[0.. gc_iBufferSize – 1] OF BYTE;
END_VAR
```

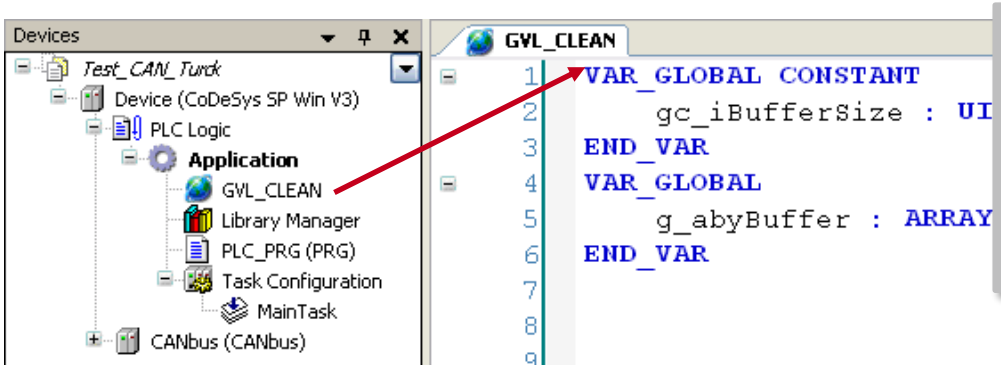
How to declare variables in CoDeSys V3

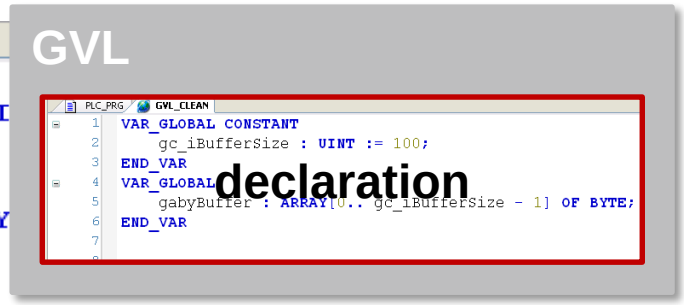
- Local variables in the POUs itself





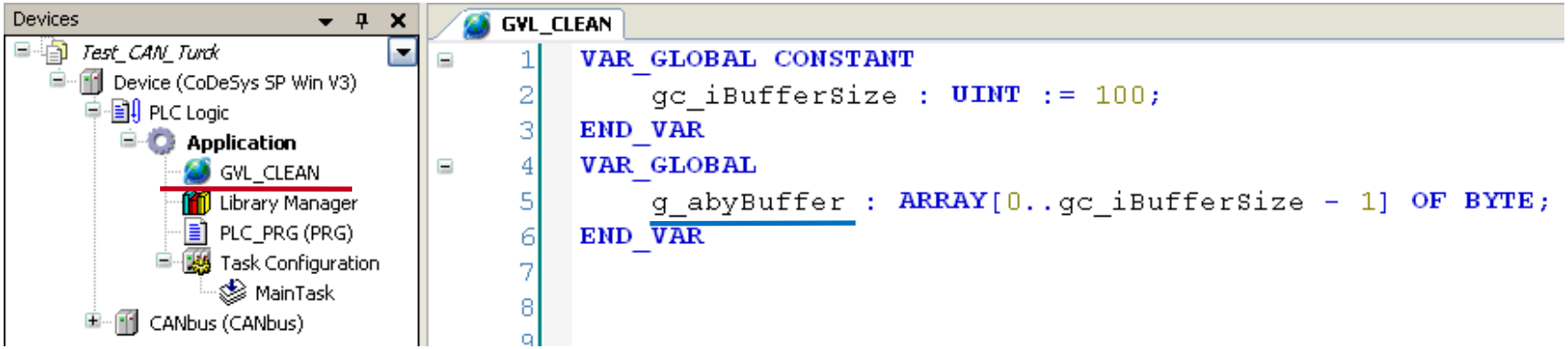
- Global variables in GVL- objects





How to use global variables in CoDeSys V3

- global variables



- Use global variables with namespace.

GVL_CLEAN . g_abyBuffer[0] := 10;

With a pragma *{attribute 'qualified_only'}* on top of a global variable list, only variables with namespace can be accessed with (e.g. GVL_CLEAN.gc_iBufferSize)

Displaymode

Evaluate variables with a specified display mode

GVL_DemoDisplayMode

```
VAR_GLOBAL
{attribute 'displaymode':='bin'}
    g_iValueInBinary : WORD;
{attribute 'displaymode':='hex'}
    g_iValueInHex : WORD;
{attribute 'displaymode':='dec'}
    g_iValueInDecimal : WORD;
END_VAR
```

Devices

- Test_CAN_Turck
  - Device [connected] (CoDeSys SP Win V3)
    - PLC Logic
      - **Application [run]**
        - GVL_CLEAN
        - GVL_DemoDisplayMode
        - Library Manager
        - PLC_PRG (PRG)

GVL_DemoDisplayMode

Device.Application.GVL_DemoDisplayMode

Expression	Type	Value	Prepared value
g_iValueInBinary	WORD	2#0011100000100101	
g_iValueInHex	WORD	16#3825	
g_iValueInDecimal	WORD	14373	

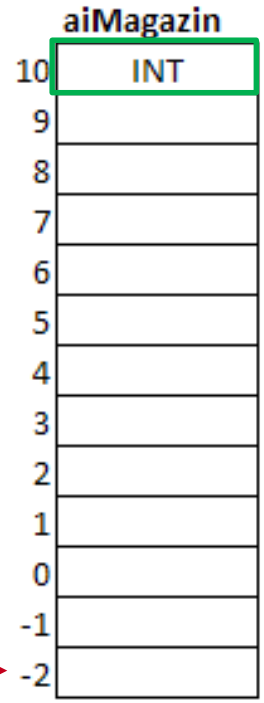
ARRAY

An Array is a collection of elements of the same data type

- Example

PLC_PRG

```
PROGRAM PLC_PRG
VAR
    aiMagazin : ARRAY[-2..10] OF INT;
END_VAR
```



Array dimension from low to high number

Initialization

- Example

PLC_PRG

```
PROGRAM PLC_PRG
VAR
    aiMagazin : ARRAY[-2..10] OF INT := [1, 5(2), 3];
END_VAR
```

Expression	Type	Value
aiMagazin	ARRAY [-2..10] OF INT	
aiMagazin[-2]	INT	1
aiMagazin[-1]	INT	2
aiMagazin[0]	INT	2
aiMagazin[1]	INT	2
aiMagazin[2]	INT	2
aiMagazin[3]	INT	2
aiMagazin[4]	INT	3
aiMagazin[5]	INT	0
aiMagazin[6]	INT	0
aiMagazin[7]	INT	0
aiMagazin[8]	INT	0
aiMagazin[9]	INT	0
aiMagazin[10]	INT	0

5(2)

How many times

Fill value

Two dimensional array

- Example

aiMagazin

	1	2	3	4
10				
9				
8				
7				
6				
5				
4				
3				
2				
1				
0				
-1				
-2				

PLC_PRG

```
PROGRAM PLC_PRG
VAR
    aiMagazin : ARRAY[-2..10, 1..4] OF INT;
END_VAR
```

1st dimension

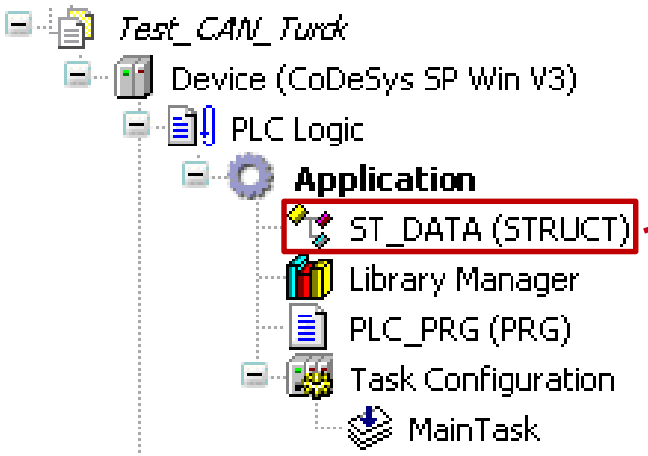
2nd dimension

An array of up to three dimensions is possible

STRUCT

A structure is a collection of elements

- Structures are created as “DUT” objects



ST_DATA

```
1  TYPE ST_DATA :
2  STRUCT
3      sArticleName : STRING(20);
4      uiArticleNo : UINT;
5      diArticleQuantity : DINT;
6  END_STRUCT
7  END_TYPE
```

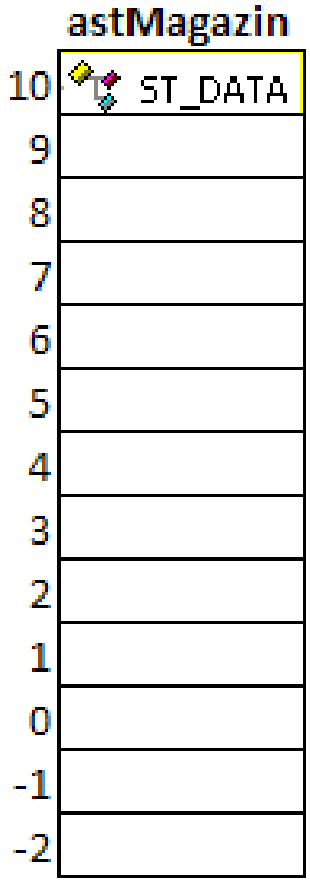
PLC_PRG

```
1  PROGRAM PLC_PRG
2  VAR
3      stArticle : ST_DATA;
4  END_VAR
```

Expression	Type	Value
stArticle	ST_DATA	
sArticleName	STRING(20)	'CoDeSys'
uiArticleNo	UINT	111
diArticleQuantity	DINT	200

Combine an ARRAY with STRUCT

- Example




```
PROGRAM PLC_PRG
VAR
    astArticle : ARRAY[-2..10] OF ST_DATA;
END_VAR
```

Expression	Type	Value
astArticle	ARRAY [-2..10] OF ST_DATA	
astArticle[-2]	ST_DATA	
sArticleName	STRING(20)	''
uiArticleNo	UINT	0
diArticleQuantity	DINT	0
astArticle[-1]	ST_DATA	
sArticleName	STRING(20)	''
uiArticleNo	UINT	0
diArticleQuantity	DINT	0
astArticle[0]	ST_DATA	
astArticle[1]	ST_DATA	
astArticle[2]	ST_DATA	
astArticle[3]	ST_DATA	

Combine an ARRAY with STRUCT

- Initialization

```
PROGRAM PLC_PRG
VAR
    astArticle : ARRAY[-2..10] OF ST_DATA
        :=  [   (   sArticleName:= 'CoDeSys V2.3',
                    uiArticleNo:= 23,
                    diArticleQuantity:= 200 ),
                (   sArticleName:= 'CoDeSys V3.4',
                    uiArticleNo:= 34,
                    diArticleQuantity:= 150 )
            ];
END_VAR
```

	Type	Value
	ARRAY [-2..10] OF ST_DATA	
	ST_DATA	
	STRING(20)	'CoDeSys V2.3'
	UINT	23
diArticleQuantity	DINT	200
astArticle[-1]	ST_DATA	
sArticleName	STRING(20)	'CoDeSys V3.4'
uiArticleNo	UINT	34
diArticleQuantity	DINT	150
astArticle[0]	ST_DATA	
sArticleName	STRING(20)	''
uiArticleNo	UINT	0
diArticleQuantity	DINT	0

Exercise

Create a variable “astCleaningPrg” as shown below

- Setup

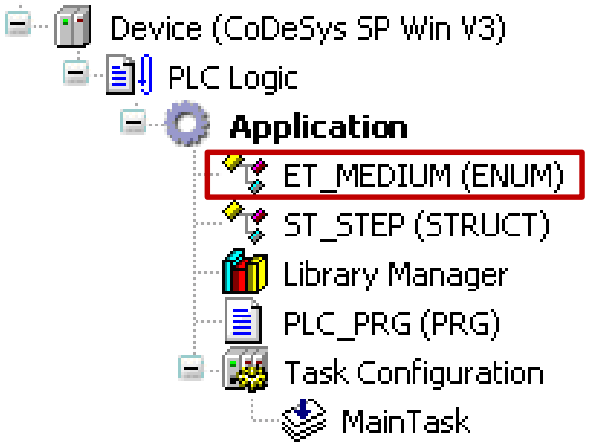
astCleaningPrg

	iMedium INT	tTimeDuration TIME	dwValvePosition DWORD	sStateText STRING(20)
0	2	T#2s	16#00001234	'First'
1	1	T#500ms	16#10000000	'Second'
2	0	T#10s	16#01010000	'Third'
3	-1	T#0ms	16#00000000	'Finish'
.				
.				
.				
31				

Enumerations

An enumeration is a user-defined data type that is made up of a number of string constants

- Setup




```
TYPE ET_MEDIUM :
(                          //  Value
    NoMedium := 0,         //  0
    Water,                 //  1
    Caustic,               //  2
    Acid,                  //  3
    HotAir   := 10,        //  10
    CoolAir                //  11
)INT;
END_TYPE
```

base data type

Exercise

Create a variable “astCleaningPrg” as shown below

- Setup

astCleaningPrg

	eMedium ET_MEDIUM	tTimeDuration TIME	dwValvePosition DWORD	sStateText STRING(20)
0	Caustic	T#2s	16#00001234	'First'
1	Water	T#500ms	16#10000000	'Second'
2	NoMedium	T#10s	16#01010000	'Third'
3	-1	T#0ms	16#00000000	'Finish'
.				
.				
.				
31				

Exercise

Create a variable “astCleaningPrg” as shown below

- Online view

PLC_PRG

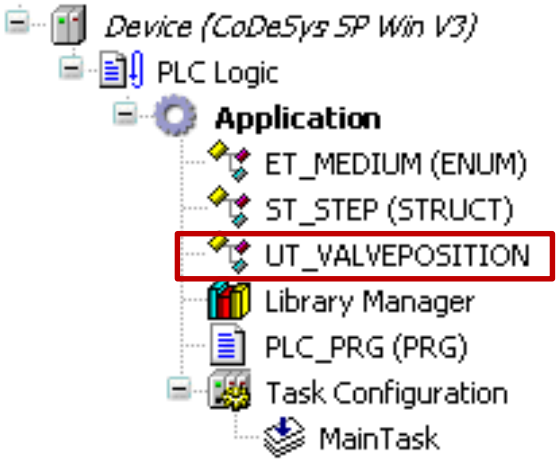
Device.Application.PLC_PRG

Expression	Type	Value
astCleaningPrg	ARRAY [0..31] OF ST_STEP	
astCleaningPrg[0]	ST_STEP	
eMedium	ET_MEDIUM	Caustic
tTimeDuration	TIME	T#2s
dwValvePosition	DWORD	16#00001234
sStateText	STRING(20)	'First'
astCleaningPrg[1]	ST_STEP	
eMedium	ET_MEDIUM	Water
tTimeDuration	TIME	T#500ms
dwValvePosition	DWORD	16#10000000
sStateText	STRING(20)	'Second'
astCleaningPrg[2]	ST_STEP	
eMedium	ET_MEDIUM	NoMedium
tTimeDuration	TIME	T#10s
dwValvePosition	DWORD	16#01010000
sStateText	STRING(20)	'Third'
astCleaningPrg[3]	ST_STEP	
eMedium	ET_MEDIUM	-1
tTimeDuration	TIME	T#0ms
dwValvePosition	DWORD	16#00000000
sStateText	STRING(20)	'Finish'

Union

In a union all elements have the same offset and they all occupy the same storage location

- Setup




```
TYPE UT_VALVEPOSITION :
UNION
    dwValue : DWORD;
    abyValue : ARRAY[0..3] OF BYTE;
END_UNION
END_TYPE
```

same data size

Union

In a union all elements have the same offset and all occupy the same storage location

- Example

PLC_PRG

Device.Application.PLC_PRG

Expression	Type	Value
astCleaningPrg	ARRAY [0..31] OF S...	
astCleaningPrg[0]	ST_STEP	
eMedium	ET_MEDIUM	Caustic
tTimeDuration	TIME	T#2s
uValvePosition	UT_VALVEPOSITION	
dwValue	DWORD	16#00001234
abyValue	ARRAY [0..3] OF BYTE	
abyValue[0]	BYTE	16#34
abyValue[1]	BYTE	16#12
abyValue[2]	BYTE	16#00
abyValue[3]	BYTE	16#00
sStateText	STRING(20)	'First'
astCleaningPrg[1]	ST_STEP	
eMedium	ET_MEDIUM	Water
tTimeDuration	TIME	T#500ms
uValvePosition	UT_VALVEPOSITION	
sStateText	STRING(20)	'Second'
astCleaningPrg[2]	ST_STEP	
astCleaningPrg[3]	ST_STEP	

Alias

An alias type specifies another data type

- Example
  - In the whole project you work with a type message
    - message is a STRING(40)
    - now we have new requirements
      - message must be a STRING(80)

```
TYPE
        MESSAGE : STRING(80);
END_TYPE

VAR
        atMessage : MESSAGE := ‘My message as alias‘;
END_VAR
```

Remanent Variables

Protect data against reset, download or reboot

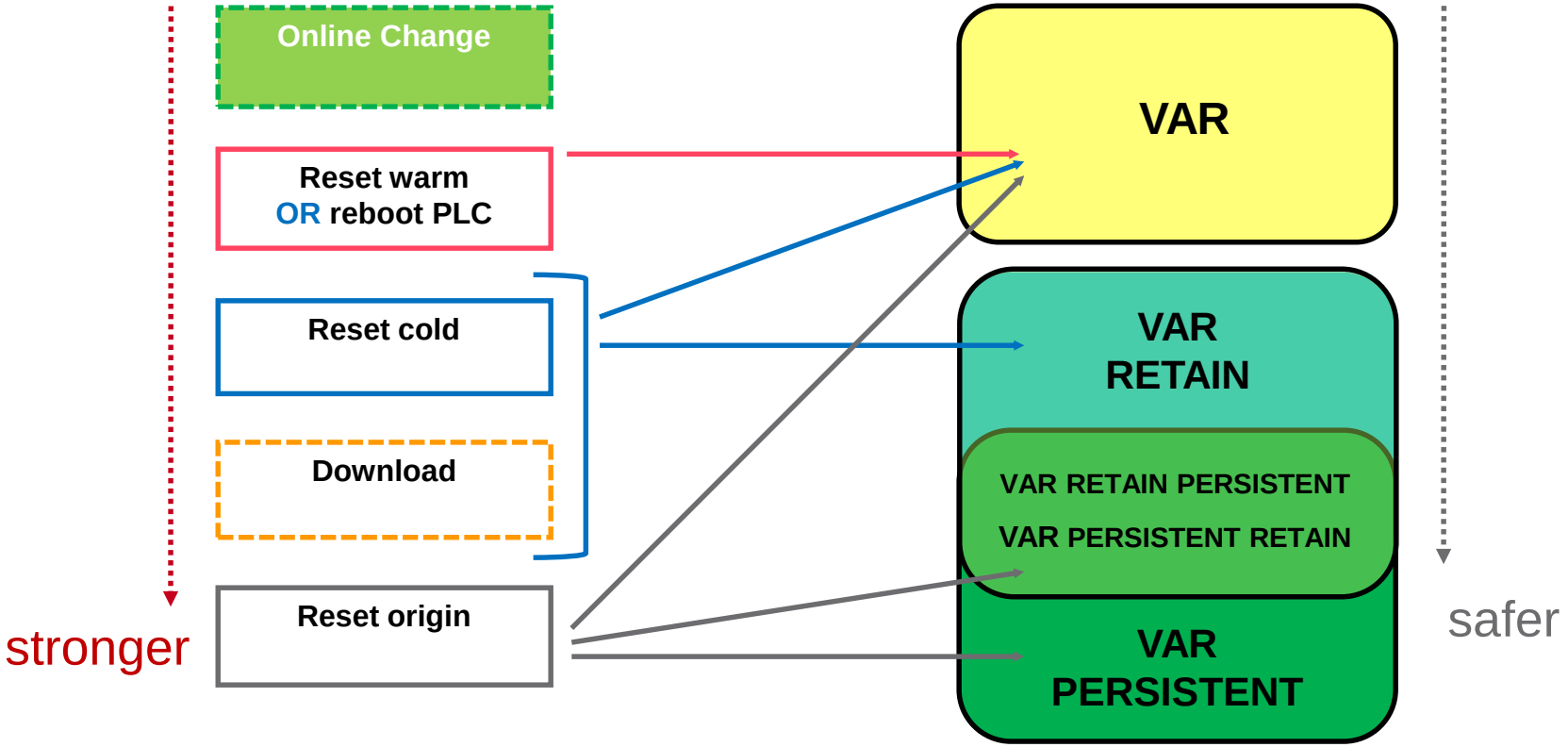
- Overview

after online command	VAR	VAR RETAIN	VAR PERSISTENT VAR RETAIN PERSISTENT VAR PERSISTENT RETAIN
Reset warm <application>	-	x	x
Reset cold <application>	-	-	x
Reset origin <application>	-	-	-
Download <application>	-	-	x
Online Change <application>	x	x	x
Reboot PLC	-	x	x

Remanent Variables

Protect data against reset, download or reboot

- Overview



Let’s check

- Why data types?
- What does operation according to the data type mean?
- How to use a data type conversion?
- What’s the difference between an array and a structure?
- What’s an enumeration?
- How does the union work?
- How to protect variables against reboot?
- What can we do in case of a download?