Pointers

Pointers store the addresses of variables, programs, function blocks, methods, and functions while an application program is running. A pointer points to one of the objects mentioned or to a variable with any data type.

Syntax of a pointer declaration:

```
<identifier>: POINTER TO <data type | function block | program | method | function>;
```

When dereferencing a pointer, the value of the address to which the pointer points is determined. In order to dereference a pointer, append the content operator to the pointer identifier (see `pt^` in the example below).

Using the address operator `ADR`, you can assign the address of a variable to a pointer.

Example

```
VAR
    pt:POINTER TO INT;  (* declaration of pointer pt *)
    var_int1:INT := 5;  (* declaration of variables var_int1 and var_int2 *)
    var_int2:INT;
END_VAR

pt := ADR(var_int1); (* pointer pt is assigned to address of varint1 *)
var_int2:= pt^;      (* value 5 of var_int1 is assigned to variable var_int2 by dereferencing of pointer pt *)
```

Hint

If a pointer to a device input is used, then the access (for example `pTest := ADR(input);`) is considered to be a write access. This leads to a compiler warning when code is generated : " `...invalid assignment target` ".

If you require a construct of this kind, you must first copy the input value (`input`) to a variable with write access.

In online mode, you can jump from a pointer to the declaration location of the referenced variable by clicking the Go To Reference command.

See also

- Operator 'Content Operator'
- Operator 'ADR'
- Command 'Go To Reference'

Function pointers to external functions

CODESYS supports function pointers that replace the INDEXOF operator. You can pass these pointers to external libraries. However, CODESYS does not provide any means for calling a function pointer from within an application in the development system. The function of the runtime system for the registration of callback functions (system library function) expects the function pointer. Depending on which callback was registered, the runtime system implicitly calls the function concerned (in the case of STOP for example). So that such a system call is possible (runtime system), you must set the corresponding object property in the Build tab.

You can use the `ADR` operator for functions, programs, function blocks, and methods. CODESYS outputs the address of a pointer to the function, not the address of the function, because the values of functions can change after an online change. This address is valid as long as the function exists on the target system.

See also

- Dialog ‘Properties’ - ‘Build’
- Operator ‘ADR’

Index access to pointers

In CODESYS, index access “[]” to type `POINTER`, `STRING`, and `WSTRING` variables is permitted.

- `pint[i]` returns the basic data type
- Index access to pointers is done arithmetically: If you use index access for a POINTER TO variable, then CODESYS computes the offset by `pint[i] = (pint + i * SIZEOF(base type))^`. The index access also causes an implicit dereferencing of the pointer. The resulting data type is the basic data type of the pointer. Note that `pint[7] != (pint + 7)^`.
- When you use the index access with a variable of the type `STRING`, you get the character at the offset of the index expression. The result is a `BYTE` type. str[i] returns the i-th character of the string as `SINT` (ASCII).
- When you use the index access with a variable of the type `WSTRING`, you get the character at the offset of the index expression. The result is a `WORD` type. wstr[i] returns the i-th character of the string as `INT` (Unicode).

Note

The result of the difference between two pointers is of type `DWORD`, even on 64-bit platforms, when the pointers are 64-bit pointers.

Note

Note that it is possible to use references which control a value directly, in contrast to pointers.

Note

Note that it is possible to monitor the memory access of pointers at runtime by the implicit monitoring function `CheckPointer`

See also

- POU ‘CheckPointer’

Operator ‘Content Operator’

This operator is an extension of the IEC 61131-3 standard.

You can use this operator to dereference pointers by appending the operator as `^` to the pointer identifier.

Caution

When using pointers to addresses, please note that applying an online change can shift address contents.

Example

ST:

```
pt : POINTER TO INT;
var_int1 : INT;
var_int2 : INT;
pt := ADR(var_int1);
var_int2 := pt^;
```

Operator ‘ADR’

This operator is an extension of the IEC 61131-3 standard.

`ADR` yields the address of its argument in a `DWORD`. You can pass this address to the manufacturer functions or assign them to a pointer in the project.

Hint

As opposed to CoDeSys V2.3, you can use the ADR operator with function names, program names, function block names, and method names. Therefore, `ADR` replaces the `INDEXOF` operator. When using function pointers, please note that you can pass a function pointer to external libraries, but it is not possible to call a function pointer from within CODESYS. To enable a system call (runtime system), you must set the respective object property (Build tab) for the function object.

Caution

When you apply an online change, address contents can shift, causing `POINTER` variables to reference invalid memory ranges. To avoid problems, make sure that CODESYS updates pointer values in every cycle.

Caution

Do not return `Pointer-TO` variables of functions and methods to the caller or assign them to global variables.

Examples

ST:

```
dwVar := ADR(bVAR);
```

Command ‘Go To Reference’

Symbol: 

Function: The command opens the declaration location of the variable that is referenced by the pointer currently in focus in online mode.

Call:

- Context menu in the declaration part or implementation code
- Menu bar: Edit ▸ Browse

Requirement: Online mode. A POU is open in the editor and the cursor is at a pointer. The referenced variable is stored in static memory.

Note

If the pointer does not point exactly to the beginning of the variable, then a corresponding message is displayed when you switch to the variable declaration.

Dialog ‘Properties’ - ‘Build’

Symbol: 

Function: This dialog includes options for compiling the object.

Call: Menu bar: View ▸ Properties ; context menu of the object in the device tree.

Name	Description
Exclude from build	☑: This object and inherently its child objects are not considered for the next compile process. The object entry is displayed in green fonts in the Devices or POUs view.
External implementation (Late link in the runtime system)	☑: CODESYS does not generate any code for this object when compiling the project. The object is linked as soon as the project is running on the target system, provided it is available there (for example, in a library). The object name is postfixed with `(EXT)` in Devices or POUs view.
Enable system call	☑: A system call (runtime system) for functions is possible. Background: As opposed to CoDeSys V2.3, the ADR operator in V3 can be used with function names, program names, function block names, and method names. It replaces the INSTANCE_OF operator. BUT: It is not possible to call function pointers from within CODESYS.
Link Always	☑: The object is marked by the compiler and therefore always included in the compile information. This means that it is always compiled and downloaded to the PLC.
Compiler defines	Here you can specify defines or conditions for compiling the object (conditional compile). You can also type in the expression `expr`, which is used in these kinds of pragmas. Multiple entries are possible as a comma-separated list (see `{define}` statements). Example: `hello, test:='1'`
Additional compiler definitions from the device description	
Defined in device	List of compiler definitions that originate from the device description. These compiler definitions are used in the build if they are not listed in the Ignored definitions field.
Ignored definitions	List of compiler definitions from the device description that are not used in the build.
>	Copies the selected compiler definition from the Defined in device field to the Ignored definitions field.
<	Moves the selected compiler definition from the Ignored definitions field to the Defined in device field. The compiler definition is used in the build.