CoDeSys

Object-Oriented Controller Programming

OOP in an IEC 61131-3 Tool

CoDeSys (Controller Development System) is an IEC 61131-3 development tool for embedded and PC based devices. Thousands of end users such as machine or plant builders rely on CoDeSys to program their controller applications.

New language constructs in the IEC 61131-3 allow the CoDeSys user to program his application object-oriented in the languages of the standard. The new object-oriented functionality is optional, meaning, it is left up to the user to choose between classic or object-oriented programming (OOP) or combine both programming philosophies.

Terminology of object-oriented programming in CoDeSys

- A function block is a **class** with exactly one method. With the expansion to full class functionality, methods and interfaces with their methods can be implemented in one function block.
- **Methods** are routines which are firmly assigned to a function block or an interface. They operate with the data of the function block but can, just like IEC functions, have I/O variables or local variables.
- An **interface** possesses a set of methods and defines the required variables of the methods. The body of the method is then programmed in the class the interface is implemented into.
- **Objects** are instances of function blocks (classes).
- With the new key word **IMPLEMENTS**, a function block implements an interface. Therefore, all methods of the interface have to be realized in the function block.
- A function block can extend a class with the new key word **EXTENDS** and then adopts the data and the methods of the class.
- Which implementation of the method is actually executed when a method is called via its interface, is decided at runtime. This functionality is called **polymorphism**.

Typical application scenarios in the automation industry

- Programming different drives with an identical functionality (e.g. homing, positioning, geterror) with interfaces and methods
- Programming different machine modules with identical or similar functionality (e.g. manual mode, automatic mode, homing) with interfaces and methods
- Data access to / Operation of different fieldbuses (e.g. start, stop, send asynchronous message) with interfaces and methods

Further functions

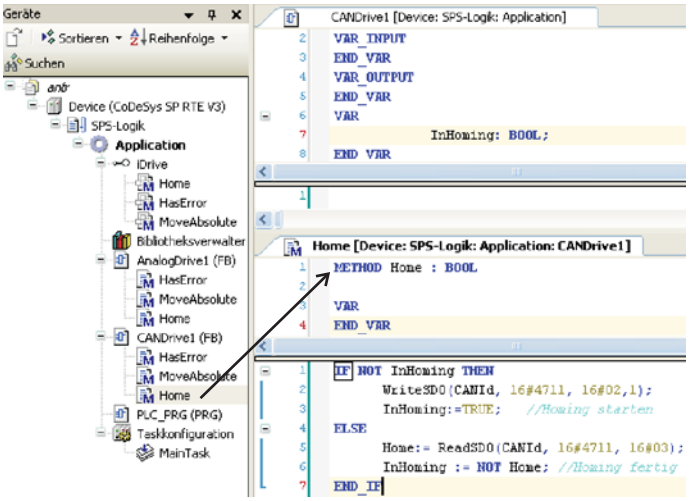
- Support of class hierarchies for the combination of classes
- Constructor and destructor methods FB_Init and FB_Exit for initializing and ending objects
- Object orientation is not restricted to a specific programming language of the IEC 61131-3. Methods and classes can be created in all IEC editors.
- Support of class properties
- Keywords such as SUPER and THIS give access to the father of a method or the method itself.
- Functions deliberately omitted:
  Dynamic memory allocation as well as special high-level language constructs (such as private, protected, internal, public, abstract or virtual)

Smart Software Solutions

Example

Different drives (CAN drives, CAN drives with extended functionality, analogue drives) are to be programmed in one controller application. All drives have the functionality Home, MoveAbsolute and HasError.

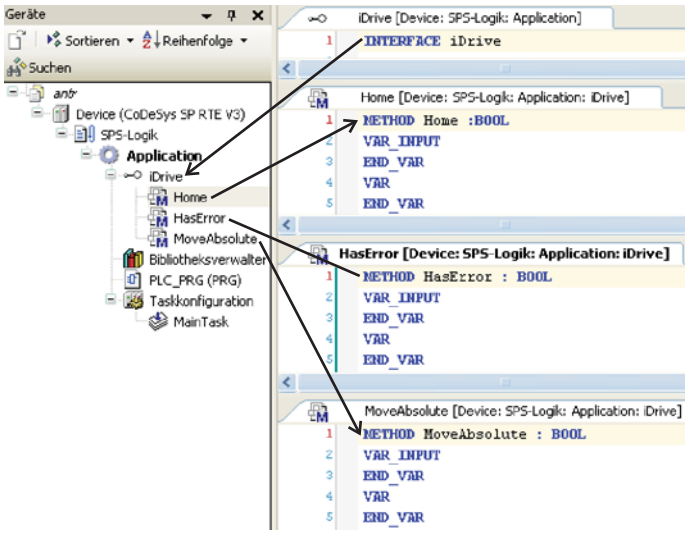
Due to this identical functionality an interface with the corresponding methods is created in the CoDeSys project. These methods only contain their call and return variables in the interface and no code.



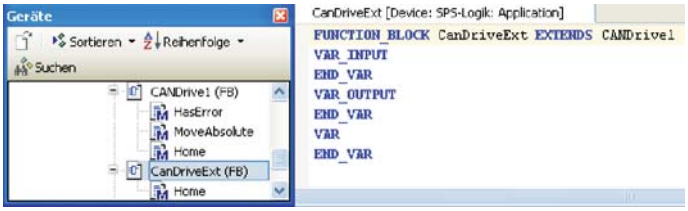
The function block CANDriveExt for the drive with extended functionality extends the class CANDrive1 and is automatically assigned all methods of the base class. The method *home* is newly created and overwrites the inherited method.

```
DriveManager [Device: SPS-Logik: Application]
VAR
      Drive1, Drive2: CANDrive;
      Drive3, Drive4: AnalogDrive;
      Drive5,
      Drive6: CanDriveExt;
      NumOfDrives: INT := 6;
      DriveArray: ARRAY[1..6] OF iDrive :=
      [Drive1, Drive5, Drive3, Drive4, Drive2, Drive6];
      i: INT;
END_VAR

//Alle Drives auf Home-Position
FOR i := 1 TO NumOfDrives DO
      DriveArray[i].Home();
END_FOR
```



The new function blocks CANDrive1 and AnalogDrive1 implement the interface iDrive and thus immediately receive the methods of this interface. The device specific call can now be programmed in these methods.



All drives are instantiated in the program DriveManager and turned into the objects Drive1, Drive2 etc. The drives are arranged in an array so that they can comfortably be moved to home position or the like in one loop. The data type of this array is the interface iDrive. When the method home is called in the loop, it is the object assignment that tells CoDeSys which method it is dealing with.

Now, if the application changes and Drive2 for example is no longer a CANDrive but an AnalogDrive or two further drives are added, only the instantiation in the declaration part of the DriveManager has to be done.

Advantages of Object-Oriented Programming

- The created program code can more easily be changed or extended and is thus much easier to maintain.
- The reusability of the code is much easier and the encapsulation clearly improved.
- The performance of the controller application is improved, except for very small applications.

- Object-oriented programming is an established standard in high-level language teaching today (This does not yet apply to the training of controller application developers.)
- Object-oriented programming is an established standard in the office world and thus a proven programming technology.