Table of contents

Methodenaufruf

Methodenaufruf

Um einen Methodenaufruf zu implementieren, werden den Schnittstellenvariablen die tatsächlichen Parameter (Argumente) übergeben. Dabei kann alternativ auf die Parameternamen verzichtet werden.

Je nach deklariertem Zugriffsmodifizierer kann eine Methode nur innerhalb des eigenen Namensraums (INTERNAL) , nur innerhalb des eigenen Programmierbausteins und seinen Ableitungen (PROTECTED) oder nur innerhalb des eigenen Programmierbausteins (PRIVATE) aufgerufen werden. Bei PUBLIC kann die Methode überall aufgerufen werden.

Innerhalb der Implementierung kann eine Methode sich selbst rekursiv aufrufen: entweder direkt mit Hilfe des THIS-Pointers, oder mit Hilfe einer lokalen Variablen für den zugeordneten Funktionsbaustein.

Methodenaufruf als virtueller Funktionsaufruf

Durch Vererbung kann es zu virtuellen Funktionsaufrufen kommen.

Virtuelle Funktionsaufrufe ermöglichen, dass derselbe Aufruf in einem Programm-Quellcode während der Laufzeit verschiedene Methoden aufruft.

In folgenden Fällen wird der Methodenaufruf dynamisch gebunden:

- Sie rufen eine Methode über einen Pointer auf einen Funktionsbaustein auf, beispielsweise pfub^.method.
  Der Pointer kann in dieser Situation auf Instanzen vom Typ des Funktionsbausteins und auf Instanzen von allen abgeleiteten Funktionsbausteinen zeigen.
- Sie rufen die Methode einer Schnittstellen-Variablen auf, beispielsweise interface1.method.
  Die Schnittstelle kann auf alle Instanzen von Funktionsbausteinen verweisen, die diese Schnittstelle implementieren.
- Eine Methode ruft eine andere Methode desselben Funktionsbausteins auf. Die Methode kann in dem Fall auch die Methode eines abgeleiteten Funktionsbausteins mit gleichem Namen aufrufen.
- Der Aufruf einer Methode erfolgt über eine Referenz auf einen Funktionsbaustein. Die Referenz kann in dieser Situation auf Instanzen vom Typ des Funktionsbausteins und auf Instanzen von allen abgeleiteten Funktionsbausteinen zeigen.
- Sie weisen VAR_IN_OUT-Variablen eines Basis-Funktionsbaustein-Typen einer Instanz eines abgeleiteten FB-Typen zu.
  Die Variable kann in dieser Situation auf Instanzen vom Typ des Funktionsbausteins und auf Instanzen von allen abgeleiteten Funktionsbausteinen zeigen.

Beispiel

Methoden überladen

Die Funktionsbausteine fub1 und fub2 erweitern den Funktionsbaustein fubbase und implementieren die Schnittstelle interface1. Es gibt die Methoden method1 und method2.

```
PROGRAM PLC_PRG
VAR_INPUT
  b : BOOL;
END_VAR

VAR  pInst : POINTER TO fubbase;
  instBase : fubbase;
  inst1 : fub1;
  inst2 : fub2;
  instRef : REFERENCE to fubbase;
END_VAR

IF b THEN
  instRef REF= inst1;          (* reference to fub1 *)
  pInst := ADR(instBase);
ELSE
  instRef REF= inst2;          (* reference to fub2  *)
  pInst := ADR(inst1);
END_IF
pInst^.method1();           (* If b is TRUE, fubbase.method1 will be called, otherwise fub1.method1 is called *)
instRef.method1();       (* If b ist TRUE, fub1.method1 will be called, otherwise fub2.method1 is called*)
```

Unter der Annahme, dass fubbase des oberen Beispiels zwei Methoden method1 und method2 enthält, überschreibt fub1 method2, aber nicht method1. Der Aufruf von method1 erfolgt folgendermaßen:

pInst^.method1();

Wenn b TRUE ist, ruft PLC Engineering fubbase.method1 auf, ansonsten fub1.method1.

Zusätzliche Ausgänge

Gemäß der Norm IEC 61131-3 können Methoden so wie normale Funktionen zusätzliche Ausgänge deklariert haben. Beim Methodenaufruf weisen Sie den zusätzlichen Ausgängen Variablen zu.

Genaue Informationen hierzu finden Sie beim Thema "Funktion".

Syntax beim Aufruf

```
<function block name>.<method name>(<first input name> := <value> (, <further input assignments>)+ , <first outp
ut name> => <first output variable name> (,<further output assignments>)+ );
```

Beispiel

Deklaration

```
METHOD PUBLIC DoIt : BOOL
VAR_INPUT
    iInput_1 : DWORD;
    iInput_2 : DWORD;
END_VAR
VAR_OUTPUT
    iOutput_1 : INT;
    sOutput_2 : STRING;
ENDVAR
```

Autruf

```
fbInstance.DoIt(iInput_1 := 1, iInput_2 := 2, iOutput_1 => iLocal_1, sOUtput_2 => sLocal_2);
```

Bei Aufruf der Methode werden die Werte der Methodenausgänge auf die lokal deklarierte Ausgabevariablen geschrieben.

Aufruf einer Methode, auch wenn die Applikation im STOP-Status ist

In der Gerätebeschreibung kann definiert sein, dass eine bestimmte Funktionsbaustein-Instanz (eines Bibliotheksbausteins) eine bestimmte Methode immer taskzyklisch aufruft. Wenn die Methode die Eingabeparameter des folgenden Beispiels enthält, arbeitet PLC Engineering die Methode auch dann ab, wenn die aktive Applikation gerade im STOP-Status ist:

Beispiel

```
VAR_INPUT
    pTaskInfo : POINTER TO DWORD;
    pApplicationInfo: POINTER TO _IMPLICIT_APPLICATION_INFO;
END_VAR

(*Now the status of the application can be queried via pApplicationInfo and the instructions can be implemented: *)
IF pApplicationInfo^.state = RUNNING THEN <instructions> END_IF;
```

Methode rekursiv aufrufen

Verwenden Sie Rekursionen vorwiegend zur Bearbeitung von rekursiven Datentypen wie beispielsweise verketteten Listen. Allgemein ist es ratsam bei der Verwendung von Rekursion vorsichtig zu sein. Bei einer unerwartet tiefen Rekursion kann es zu einem Stacküberlauf und damit zu einem Maschinenstillstand kommen."

Innerhalb ihrer Implementierung kann eine Methode sich selbst aufrufen:

- direkt mit Hilfe des THIS-Pointers
- indirekt mit Hilfe einer lokalen Funktionsbaustein-Instanz des Basisfunktionsbausteins

Üblicherweise wird bei einem solchen rekursiven Aufruf eine Compilerwarnung ausgegeben. Wenn die Methode mit dem Pragma {attribute 'estimated-stack-usage' := '<estimated stack size in bytes>'} versehen ist, wird die Compilerwarnung unterdrückt. Im Kapitel "Attribut 'estimated-stack-usage' " finden Sie ein Implementierungsbeispiel.

Siehe auch

- ↘ „Attribut 'estimated-stack-usage'“
- ↘ „THIS“
- ↘ „SUPER“
- ↘ „Objekt 'Methode'“
- ↘ „Objekt 'Eigenschaft'“