

# SMLC Tutorial

An Introduction to Creating  
User-Defined Functions and  
Function Blocks

Copyright © 2006 ORMEC Systems Corp  
All rights reserved

Mick Oakley,  
Principal Applications Engineer  
ORMEC Systems Corp  
19 Linden Park  
Rochester NY 14625

(585) 385-3520  
[www.ormec.com](http://www.ormec.com)



# Creating User-Defined Functions and Function Blocks

## 1 Introduction

User-defined functions and function blocks provide an important tool that enables you to create well-structured, modular programs with reusable elements.

Here are some of the characteristics that will help you decide whether you should create a function or function block:

### 1.1 Functions:

- Must have at least one input, but may have more than one<sup>1</sup>.
- May only have one output.
- Do not retain internal variable values from one program scan to the next.
- For the same input values, must always return the same output<sup>2</sup>.
- Must always return an output during the same program scan they are called.

### 1.2 Function Blocks:

- Must have at least one input, but may have more than one.
- Must have at least one output, but may have more than one.
- May take one or more program scans to complete execution.
- May retain internal variable values from program scan to the next
- Do not necessarily return the same output values for a given set of input values.

### 1.3 Choice of Language<sup>3</sup>

#### 1.3.1 Functions

The nature of the tasks performed by functions, and the fact they should complete execution and return a value in the same scan they are called, makes ST the best choice of language for their implementation. For the same reasons, SFC is usually a poor choice.

#### 1.3.2 Function Blocks

The best language choice for implementing function blocks depends on your design goals. If you want the function block to complete execution and return its outputs in the same scan that it is executed, ST is usually the best language choices. If the operation the function block performs has multiple steps and will take more than one program scan to complete, SFC is usually the best starting point with the SFC steps being implemented in ST or LD.

---

<sup>1</sup> The one input minimum is not a firm requirement; it is only needed to ensure the function appears correctly when used in an LD program. If the function needs no inputs, it is best to include a “dummy” Boolean input for this purpose.

<sup>2</sup> This is not true for functions that return values from the operating system like SysRtcGetTime..

<sup>3</sup> Individual preferences and the nature of the task at hand may override these recommendations.

## 2 Programming Function Blocks

There are two types of function block. One causes a single operation, which may take more than one program scan to complete, but it is executed once and it is done. The tasks performed might be as simple as writing a value to a file or as complex as a complete sequence of manufacturing a part. This type of function block should be triggered by a rising edge on an Execute input. Once the function block has been triggered, its execution continues *even if the Execute input returns false*.

The second type of function block is started by a true value on an Enable input. It runs for as long as the Enable input is true and *stops as soon as the Enable input becomes false*.

You should note that the code you write when creating your function block will be run any time an instance of the function block is encountered in the program, *regardless of the state of the Execute or Enable input*.

*The behavior regarding the Execute and Enable inputs described above must be implemented by the code you write (see Examples in section 3).*

### 2.1 General rules for function block inputs<sup>4</sup>:

Every function block has either an **Enable** input or an **Execute** input.

#### 2.1.1 Enable input (BOOL):

The function blocks functionality is performed *every time* the block is called with the Enable input true. (MC\_Read... blocks all use Enable inputs). When the Enable input becomes false, the function block should stop executing.

#### 2.1.2 Execute input (BOOL):

The function blocks functionality is performed on the *rising edge* of the Execute input. With the edge triggered “Execute” input your function block may monitor the Execute input for a rising edge to accept new input values during execution.

#### 2.1.3 Input Variables

All input variables for function blocks should have default values assigned in the Variable Declaration.

In function blocks that take multiple program scans to complete, the rising edge of the Execute or Enable input should copy input variables, for which you need to ignore changes during execution, to local variables.

#### 2.1.4 Missing Input Parameters:

If any parameter of a function block input is missing (“open”) then the value from the previous invocation of this instance will be used. In the first invocation the initial (or default) value is applied. This functionality is automatically provided by CoDeSys.

#### 2.1.5 SFCReset Input

Function blocks that are programmed using SFC may need an SFCReset input.

If an SFC step in your program becomes inactive before a function block has completed its execution, the function block may be waiting for some event that may never occur

---

<sup>4</sup> These rules are based on the PLCOpen standard.

## Creating User-Defined Functions and Function Blocks

such as an axis that has now been stopped by another part of the program, reaching a certain position.

The next time the code in the same instance of the function block is run, it will still be running the code in the step that was previously active. This step will be “stuck” waiting for an event that will never happen.

### 2.1.5.1 How to Use the SFCReset Input in the Program

If a function block in an SFC step has an SFCReset input you should set a BOOL variable in the entry action of the step. The variable should be attached to the SFCReset of the function block(s) in the step. The variable should be set false at the very end of the step.

On the first scan through the program SFC step the SFCReset input will be true forcing the function block back to the Init step. On the next scan the SFCReset will be false allowing the function block to respond to a subsequent rising edge on its Execute input. (See section 3.5.3)

## 2.2 General rules for function block outputs<sup>4</sup>:

### 2.2.1 Done output (BOOL):

The Done output indicates the function block has completed execution successfully and all other outputs showing the results of successful completion are valid. All such outputs should be set in the same scan as the Done output.

### 2.2.2 Error Handling Behavior:

All function blocks have two outputs (**Error & ErrorID**) that deal with errors that occur during execution:

#### 2.2.2.1 Error Output (BOOL)

Rising edge of the Error output indicates an error occurred during the execution of the Function Block.

#### 2.2.2.2 ErrorID Output (WORD)

If the Error output is true, ErrorID should indicate an error code number

### 2.2.3 Exclusivity

The Done and Error outputs are mutually exclusive (they should never be true at the same time).

When the Execute input is false, all outputs must be reset to false, zero, null string, etc. on the next program scan after completion of the function block.

For function blocks with Enable inputs all outputs must be reset to false, zero, null string, etc. on the next program scan after the Enable input returns false.

### 2.2.4 Intermediate Result Outputs

Outputs indicating the function block status or intermediate results before the function block is complete may be set at any time during execution. However they should be reset along with the other outputs.

# Creating User-Defined Functions and Function Blocks

## 2.2.5 Re-Executing

If an instance of a Function Block receives a new execute before it finished (such as a series of commands on the same instance), the Function Block will not return any feedback, like 'Done' or 'Error', for the previous action.

## 2.2.6 Resetting Outputs

Function blocks with Execute inputs should reset all their outputs the first time the Execute input is false, ***on the program scan following setting the Done (or Error) output***. This is to ensure the outputs stay on for one program scan ***even if the Execute is reset before the Done (or Error) output is set***.

# 3 Examples

## 3.1 Function in ST

### 3.1.1 Variable Declarations

```
FUNCTION MoveTime : DINT
(*
  Calculates the move time in mS for an X,Yand Z move.
*)
VAR_INPUT
  X_Distance: DINT:=0; (* mils *)
  Y_Distance: DINT:=0; (* mils *)
  Z_Distance: DINT:=0; (* mils *)
  Feedrate: WORD:=0; (* inches/sec *)
END_VAR
VAR
END_VAR
```

### 3.1.2 Code Section

```
IF FeedRate > 0 THEN
  MoveTime :=
    REAL_TO_DINT(
      SQRT(
        EXPT(X_Distance, 2)
        + EXPT(Y_Distance, 2)
        + EXPT(Z_Distance, 2)
      ) / FeedRate
    );
ELSE
  MoveTime :=0;
END_IF
```

# Creating User-Defined Functions and Function Blocks

## 3.2 Single Scan Function Block in ST

### 3.2.1 Variable Declarations

```
FUNCTION_BLOCK DistBetweenAxes_ST
(*
  Calculates the distance between two axes.
*)
VAR_INPUT
  Enable: BOOL:=FALSE;      (* True enables the function block *)
  Tag: STRING:='DistBetweenAxes'; (* Text description for error log *)
  Axis1: AXIS_REF;          (* Reference to axis 1 *)
  Axis2: AXIS_REF;          (* Reference to axis 2 *)
END_VAR
VAR_OUTPUT
  Done: BOOL;               (* True indicates the outputs are valid *)
  Error:BOOL;               (* True indicates an error occurred *)
  ErrorID: WORD;            (* Error identification number *)
  Distance: DINT;           (* The distance between Axis1 and Axis2)
END_VAR
VAR
  ReadAxis1Position: MC_ReadActualPosition;
  ReadAxis2Position: MC_ReadActualPosition;
END_VAR
```

### 3.2.2 Code Section

Notes:

The outputs can be set before the end of the function block since we know it will be completed in one program scan.

It would not be good practice to set the Done output before calculating the Distance output if it were possible the calculation could generate an error.

```
IF Enable THEN
  IF NOT Error THEN
    (* If there have been no errors, read axis 1 position *)
    ReadAxis1Position(Enable:= TRUE,
      Tag:= CONCAT(Tag, ' - ReadAxis1Position'),
      Axis:= Axis1,
      Error=> Error,
      ErrorID=> ErrorID);
  END_IF
  IF NOT Error THEN
    (* If there have been no errors, read axis 2 position *)
    ReadAxis2Position(Enable:= TRUE,
      Tag:= CONCAT(Tag, ' - ReadAxis2Position'),
      Axis:= Axis2,
      Error=> Error,
      ErrorID=> ErrorID);
  END_IF
  Done := NOT Error AND ReadAxis1Position.Done AND ReadAxis2Position.Done;
  IF Done THEN
    (* If there have been no errors and both reads are done,
      calculate the distance between the axes*)
```

## Creating User-Defined Functions and Function Blocks

```
Distance := ReadAxis2Position.Position - ReadAxis1Position.Position;
ELSE
  (* Both reads are not done, so set the distance to zero *)
  Distance := 0;
END_IF
ELSE
  (* Enable is false so reset the outputs *)
  Done := FALSE;
  Error := FALSE;
  ErrorID := 0;
  Distance := 0;
END_IF
```

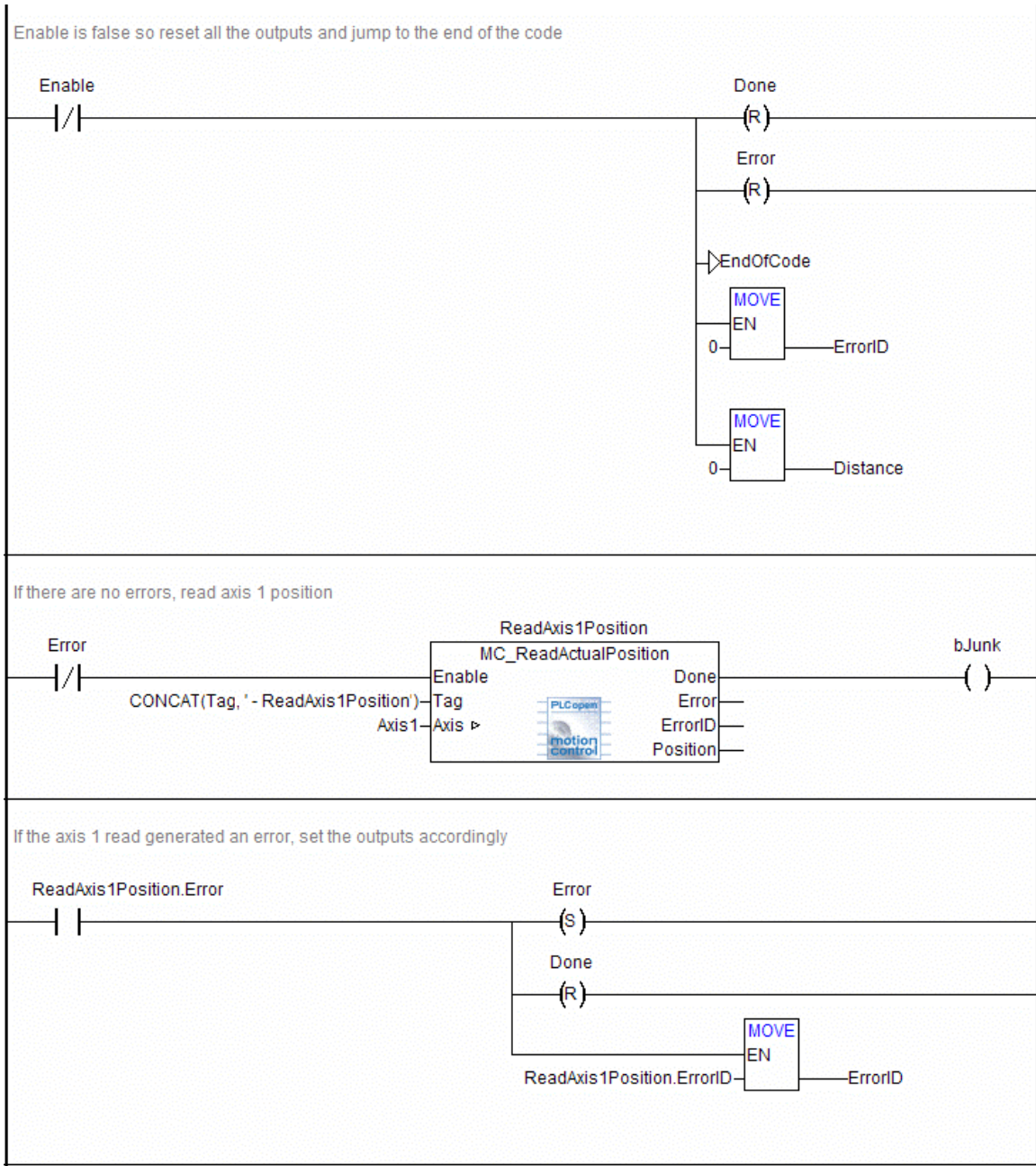
### 3.3 Single Scan Function Block in LD

#### 3.3.1 Variable Declarations

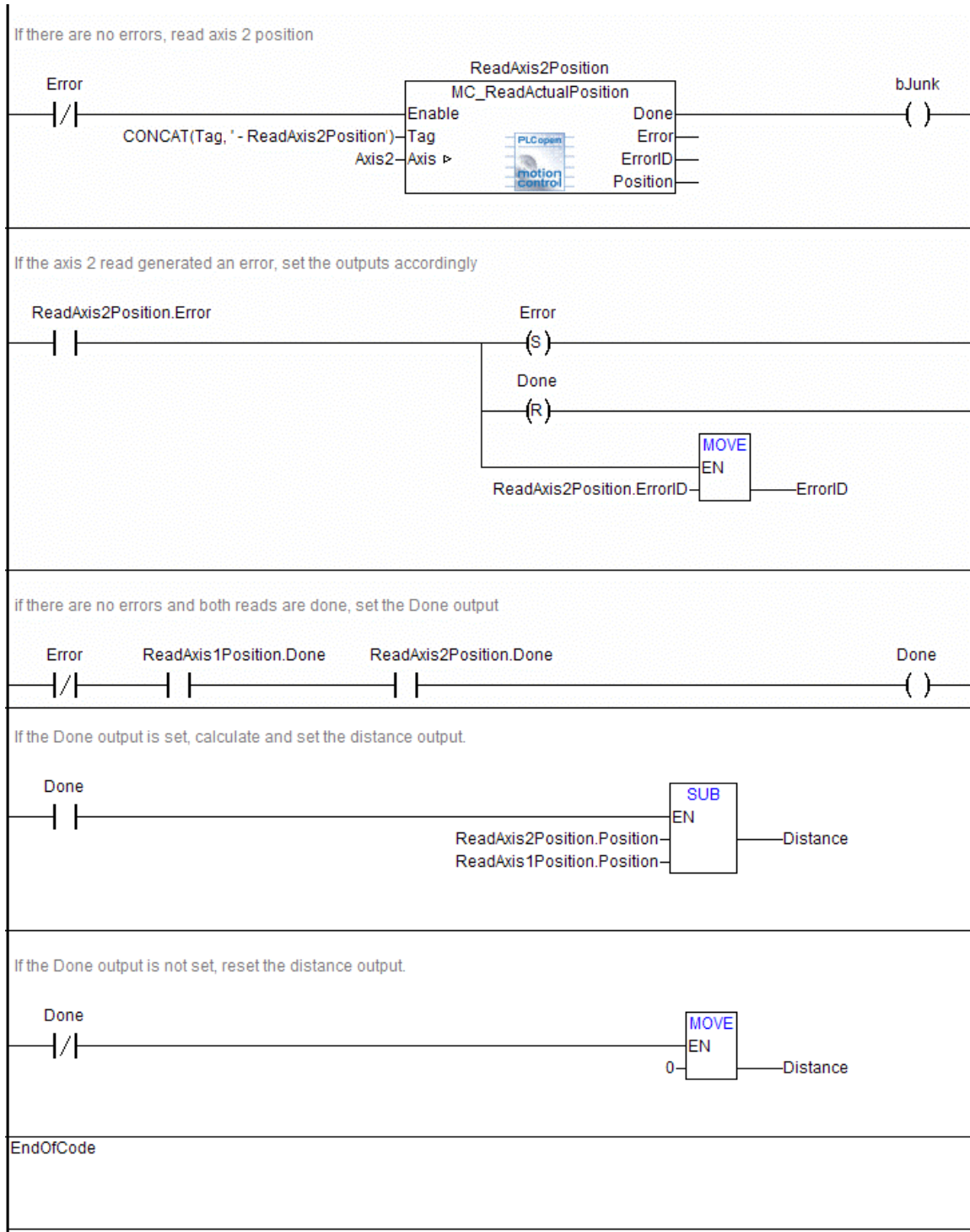
```
FUNCTION_BLOCK DistBetweenAxes_LD
(*
  Calculates the distance between two axes.
*)
VAR_INPUT
  Enable: BOOL:=FALSE;      (* True enables the function block *)
  Tag: STRING:='DistBetweenAxes'; (* Text description for error log *)
  Axis1: AXIS_REF;          (* Reference to axis 1 *)
  Axis2: AXIS_REF;          (* Reference to axis 2 *)
END_VAR
VAR_OUTPUT
  Done: BOOL;      (* True indicates the outputs are valid *)
  Error:BOOL;      (* True indicates an error occurred *)
  ErrorID: WORD;   (* Error identification number *)
  Distance: DINT;  (* The distance between Axis1 and Axis2)
END_VAR
VAR
  bJunk; BOOL;
  ReadAxis1Position: MC_ReadActualPosition;
  ReadAxis2Position: MC_ReadActualPosition;
END_VAR
```

# Creating User-Defined Functions and Function Blocks

## 3.3.2 Code Section



# Creating User-Defined Functions and Function Blocks



# Creating User-Defined Functions and Function Blocks

## 3.4 Multi-Scan Enable Function Block in SFC

### 3.4.1 Variable Declarations

FUNCTION\_BLOCK Sequence

VAR\_INPUT

Enable: **BOOL**; (\* Rising edge executes the function block \*)  
SFCReset: **BOOL**; (\* True resets and holds the function block in the Init step \*)  
OtherInput\_1: **DINT**;  
OtherInput\_2: **DINT**;  
OtherInput\_3: **DINT**;  
OtherInput\_4: **DINT**;

END\_VAR

VAR\_OUTPUT

Done: **BOOL**; (\* The function block outputs are valid \*)  
Error: **BOOL**; (\* The function block generated an error \*)  
ErrorID: **WORD**; (\* Error identification number \*)  
Enabled: **BOOL**; (\* The function block is enabled \*)  
StepNumber: **INT**; (\* A status or intermediate output \*)  
OtherOutput: **DINT**; (\* A result output \*)

END\_VAR

VAR

bStepDone: **BOOL**; (\* SFC step is complete, mutually exclusive with bError \*)  
bError: **BOOL**; (\* SFC step had an error, mutually exclusive with bStepDone \*)  
wErrorID: **WORD**; (\* Internal value of the error code \*)  
diOtherOutput: **DINT**; (\* Internal copy of output variable \*)  
diOtherInput\_1: **DINT**; (\* Internal copy of input variable \*)  
diOtherInput\_2: **DINT**; (\* Internal copy of input variable \*)  
diOtherInput\_3: **DINT**; (\* Internal copy of input variable \*)  
diOtherInput\_4: **DINT**; (\* Internal copy of input variable \*)

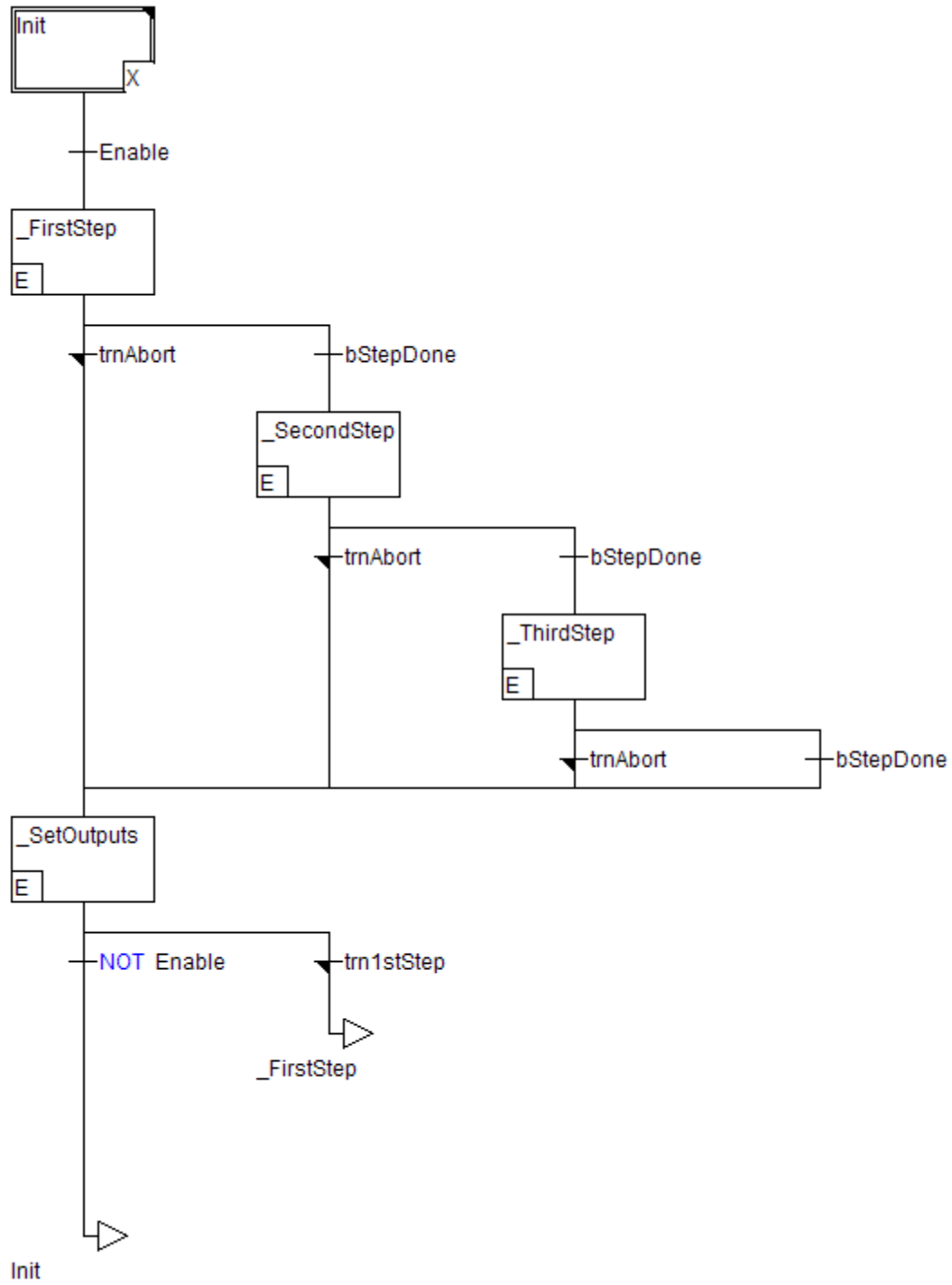
END\_VAR

### 3.4.2 Code Section

Notes: The actual design of the SFC between the Init and \_SetOutputs steps will vary depending on the specific requirements for the function block. However, the guidelines listed below should apply to most designs.

- The Init action should reset all FB outputs
- The Init Exit action should reset the internal variables and copy the FB inputs to local variables in the FB. It may also set the Enabled output, if used.
- The entry action for each step should reset bStepDone.
- Each step may set outputs to provide intermediate results and/or the status of the FB.
- The code within each step should be conditional on Enable being true.
- The conditional statements for the trnAbort transitions should be: bError **OR NOT** Enable. Where bError is set in the step if an error is detected.
- The conditional statement for the trnFirstStep transition should be Enable **AND NOT** Error
- The \_SetOutputs entry action should set the Done, Error and ErrorID outputs appropriately providing the Enable input is true.

# Creating User-Defined Functions and Function Blocks



# Creating User-Defined Functions and Function Blocks

## 3.5 Multi-Scan Execute Function Block in SFC

### 3.5.1 Variable Declarations

**FUNCTION\_BLOCK** Sequence

**VAR\_INPUT**

Execute: **BOOL**; (\* Rising edge executes the function block \*)  
SFCReset: **BOOL**; (\* True resets and holds the function block in the Init step \*)  
OtherInput\_1: **DINT**;  
OtherInput\_2: **DINT**;  
OtherInput\_3: **DINT**;  
OtherInput\_4: **DINT**;

**END\_VAR**

**VAR\_OUTPUT**

Done: **BOOL**; (\* The function block has completed without error \*)  
Error: **BOOL**; (\* The function block generated an error \*)  
ErrorID: **WORD**; (\* Error identification number \*)  
Executing: **BOOL**; (\* The function block is executing \*)  
StepNumber: **INT**; (\* A status or intermediate output \*)  
OtherOutput: **DINT**; (\* A result output \*)

**END\_VAR**

**VAR**

bStepDone: **BOOL**; (\* SFC step is complete, mutually exclusive with bError \*)  
bError: **BOOL**; (\* SFC step had an error, mutually exclusive with bStepDone \*)  
wErrorID: **WORD**; (\* Internal value of the error code \*)  
diOtherOutput: **DINT**; (\* Internal copy of output variable \*)  
diOtherInput\_1: **DINT**; (\* Internal copy of input variable \*)  
diOtherInput\_2: **DINT**; (\* Internal copy of input variable \*)  
diOtherInput\_3: **DINT**; (\* Internal copy of input variable \*)  
diOtherInput\_4: **DINT**; (\* Internal copy of input variable \*)

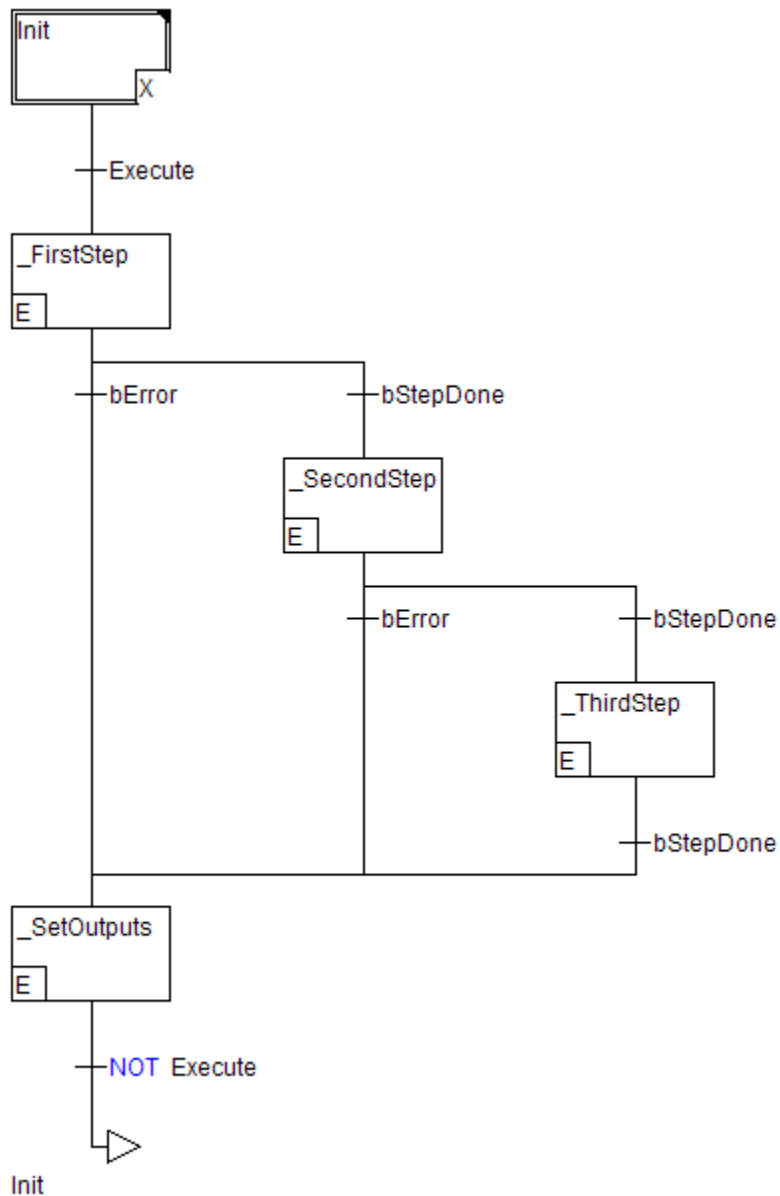
**END\_VAR**

# Creating User-Defined Functions and Function Blocks

## 3.5.2 Code Section

Notes: The actual design of the SFC between the Init and \_SetOutputs steps will vary depending on the specific requirements for the function block. However, the guidelines listed below should apply to most designs.

- The Init action should reset all FB outputs
- The Init Exit action should reset the internal variables and copy the FB inputs to local variables in the FB. It may also set the Executing output, if used.
- The entry action for each step should reset bStepDone.
- Each step may set outputs to provide intermediate results and/or the status of the FB.
- The \_SetOutputs entry action should reset the Executing output, if used, and set the Done, Error and ErrorID outputs appropriately. It should also set any outputs representing the *final* result of execution.

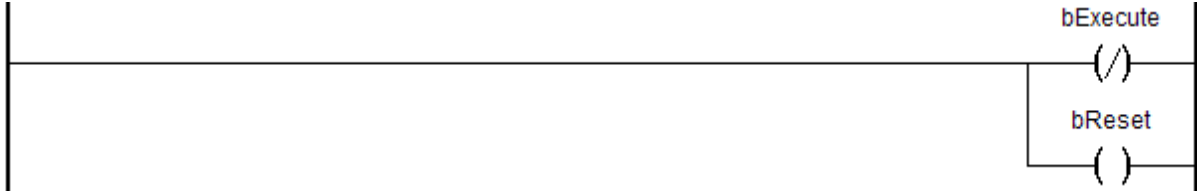


# Creating User-Defined Functions and Function Blocks

## 3.5.3 Use of SFCReset Input

### 3.5.3.1 Calling SFC Step Entry Action

The entry action for the program step should set the bExecute variable false and the bReset variable true.



### 3.5.3.2 Calling SFC Step Action

On the first scan through the Action, The Execute input will be false and the SFCReset input true. This will force the SFC within the function block to its Init step.

On the second scan, the Execute input will be true and the SFCReset input false. This will cause the function block to execute.

