LEFT-ALIGNABLE INTERBUS MASTER

INTERBUS startup with the AXC F XT IB left-alignable Axioline F extension module in PLCnext Engineer

Application note
108949_en_00

1 Description

Up to 255 INTERBUS remote bus devices can be integrated into an Axioline F station using the AXC F XT IB left-alignable Axioline F extension module (INTERBUS master). The INTERBUS master can be aligned to the left of the following AXC F .../GTC F ... PLCnext controllers:

– AXC F 2152 with firmware version ≥ 2019.3
– GTC F 2172 with firmware version ≥ 2019.0 LTS

With the aid of an example, this document describes how to integrate the INTERBUS master and the connected INTERBUS remote bus devices into an Axioline F station and how to start them up in PLCnext Engineer.

Make sure that the left-alignable INTERBUS master is only aligned next to a PLCnext controller with a firmware version that is permitted.

Make sure you always use the latest documentation.
It can be downloaded via the product page at phoenixcontact.net/products.

Table of contents

2 Adding necessary libraries

Depending on the controller used, in order to use the INTERBUS master, you need to add certain libraries to your PLCnext Engineer project:

Controller	Necessary libraries
AXC F 2152	– Interbus physical – PLCnext Controller **Please note:** If you have created the project with the “Empty AXC F 2152 v00 / 2019.3.0 project” project template, the “PLCnext Controller” library is already added by default.
GTC F 2172	– GTC – PLCnext Controller

To add a library in the “COMPONENTS” area, proceed as follows:

- In the “COMPONENTS” area, open the “Libraries (x)” section.
- Right-click on “Libraries (x)”.
- From the context menu, select “Add Library...”.

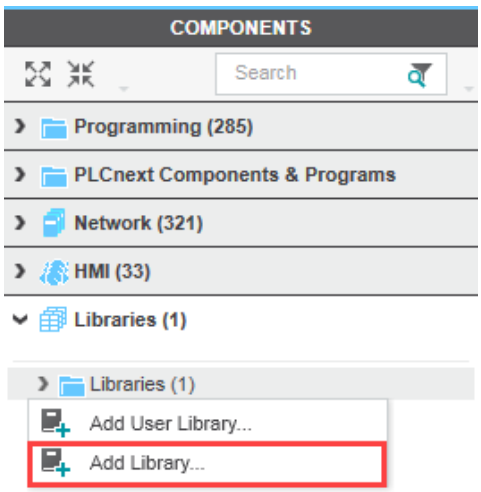

Figure 1 Context menu, “Add Library...”

- In the file explorer that opens, select the library to be added.
- Click on the “Open” button.

The selected library is now displayed in the “Libraries (x)” section in the “COMPONENTS” area.

3 Adding the INTERBUS master to the bus configuration in PLCnext Engineer

3.1 For projects with a controller of the type AXC F 2152

- Double-click on the controller node in the “PLANT” area.

The controller editor group opens.

- Select the “Settings” editor.
- Select the “Hardware Extensions” view.
- From the “Interbus” drop-down list, select the “AXC F XT IB (2403018)” INTERBUS master.

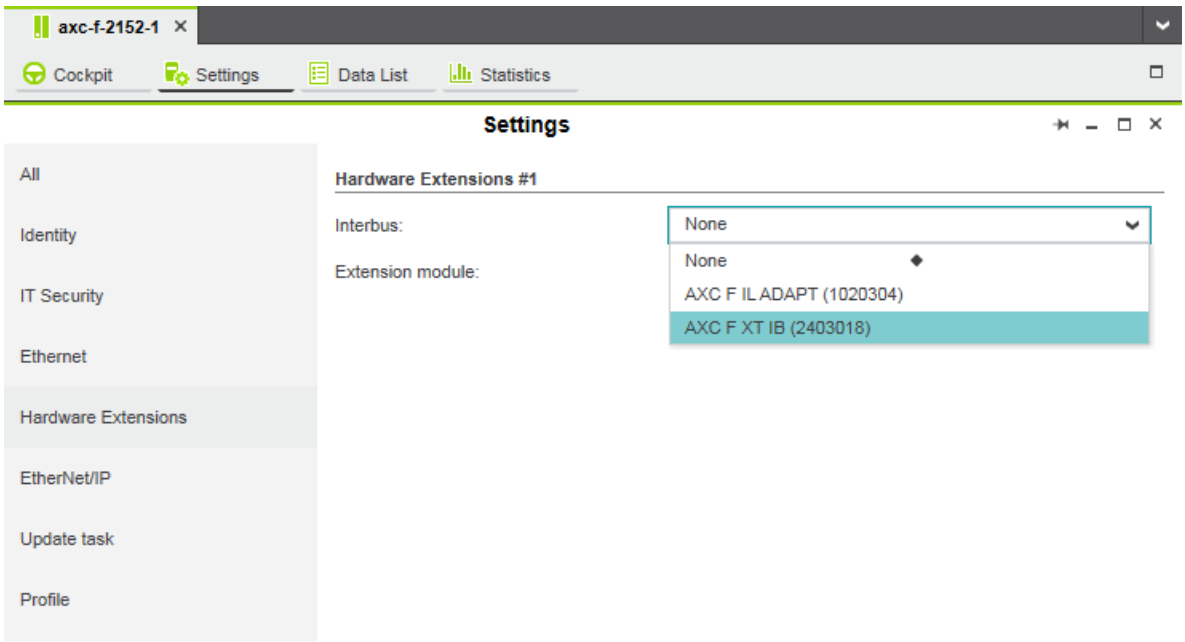

Figure 2 “Hardware Extensions” view, “AXC F XT IB” setting

The “AXC F IL ADAPT / AXC F XT IB (x)” node is now displayed in the “PLANT” area.

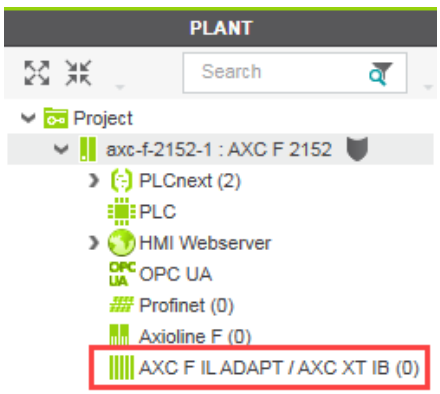

Figure 3 “AXC F IL ADAPT / AXC F XT IB (x)” node in the “PLANT” area

- Double-click on the “AXC F IL ADAPT / AXC F XT IB (x)” node in the “PLANT” area.

The “AXC F IL ADAPT / AXC F XT IB” controller editor group opens.

- Select the “Device List” editor.
- Select “Select type here” in the first row of the “Device List” editor.

The role picker opens.

- Select the “IB 256” module in the role picker.

The “IB 256” module is added and mapped under the “AXC F IL ADAPT / AXC F XT IB (x)” node in the “PLANT” area.

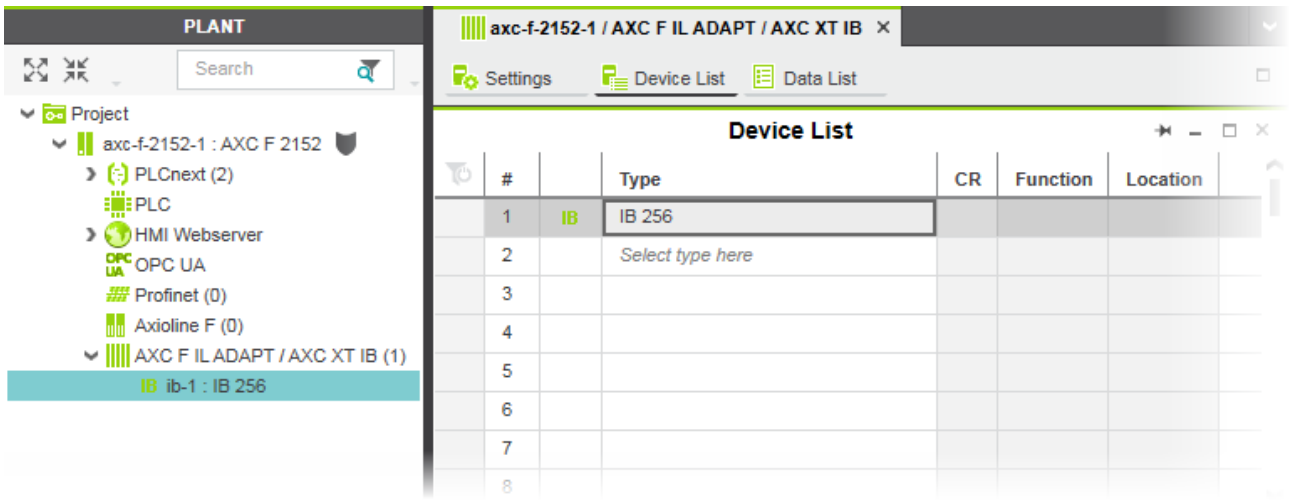

Figure 4 “IB 256” module in the “PLANT” area and in the “Device List” editor

3.2 For projects with a controller of the type GTC F 2172

- Double-click on the “Interbus (x)” node in the “PLANT” area.

The “/ Interbus” controller editor group opens.

- Select the “Device List” editor.
- Select “Select type here” in the first row of the “Device List” editor.

The role picker opens.

- Select the “IB 256” module in the role picker.

The “IB 256” module is added and mapped under the “Interbus (x)” node in the “PLANT” area.

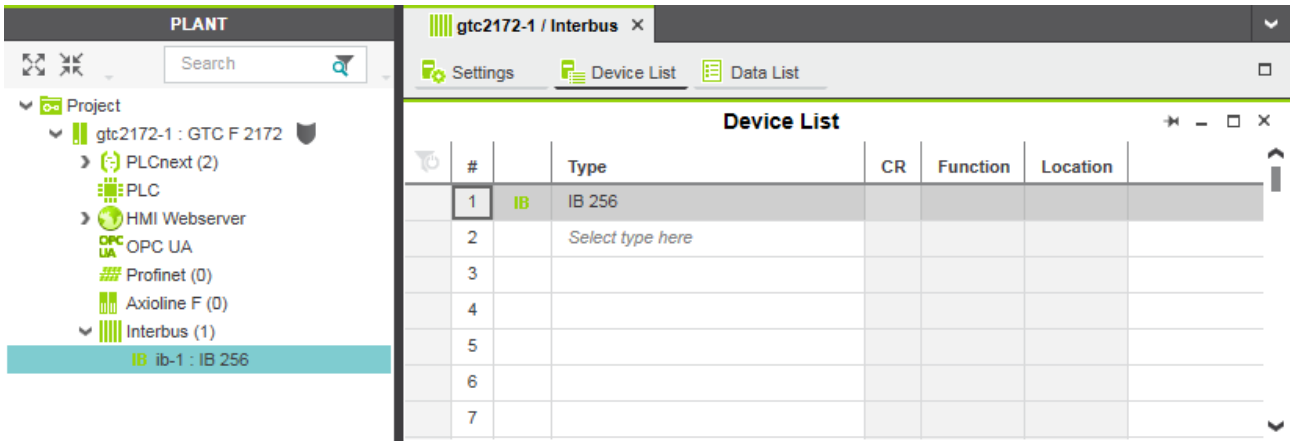

Figure 5 “IB 256” module in the “PLANT” area and in the “Device List” editor

4 Function blocks

PLCnext Engineer includes the following function blocks for INTERBUS startup and PCP communication:

- IB_CONTROL_NEXT
- PCP_CONNECT
- PCP_READ
- PCP_WRITE

For information on the function blocks for PCP communication, please refer to the online help for PLCnext Engineer.

The IB_CONTROL_NEXT function block is required for INTERBUS startup.

IB_CONTROL_NEXT function block

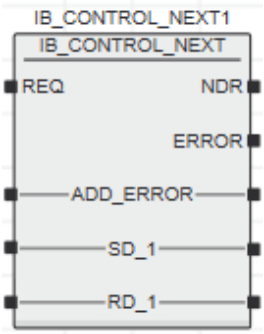

Figure 6 IB_CONTROL_NEXT function block

The IB_CONTROL_NEXT function block sends G4 firmware services and receives the corresponding requested data.

Input

Name	Data type	Description
REQ	BOOL	The function block is started via a rising edge at this input. The function block is prepared for a restart via a falling edge at this input.

Inputs/outputs

Name	Data type	Description
ADD_ERROR	ARRAY [2] OF WORD or ARRAY [4] OF BYTE	An error has occurred. ADD_ERROR contains the corresponding error code.
SD_1	ARRAY OF BYTE or ARRAY OF WORD	Buffer for G4 firmware services to be sent (transmit buffer)
RD_1	ARRAY OF BYTE or ARRAY OF WORD	Buffer for received data (receive buffer)

Outputs

Name	Data type	Description
NDR	BOOL	A rising edge indicates that new data has arrived in receive buffer RD_1.
ERROR	BOOL	TRUE: An error has occurred. The ADD_ERROR input/output provides error details FALSE: No error has occurred

The following sections describe how to start up INTERBUS using the IB_CONTROL_NEXT function block.

5 Creating data types

In order to handle INTERBUS process data, you need to create ARRAY data type fields. To do this, proceed as follows:

- In the “COMPONENTS” area, open the “Programming (x), Local (x), Data Types (x)” section.
- Right-click to open the context menu and select “Add Data Type Worksheet”.

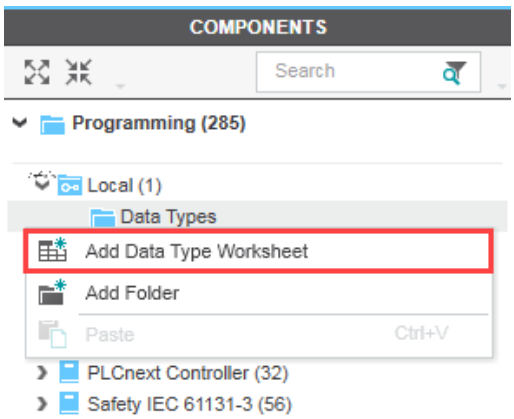

Figure 7 “Add Data Type Worksheet” in the context menu

The data type worksheet is displayed in the “Local (x), Data Types (x)” section.

- Enter a unique and meaningful designation for the data type worksheet.
- Double-click to open the data type worksheet.
- Create the necessary data types.

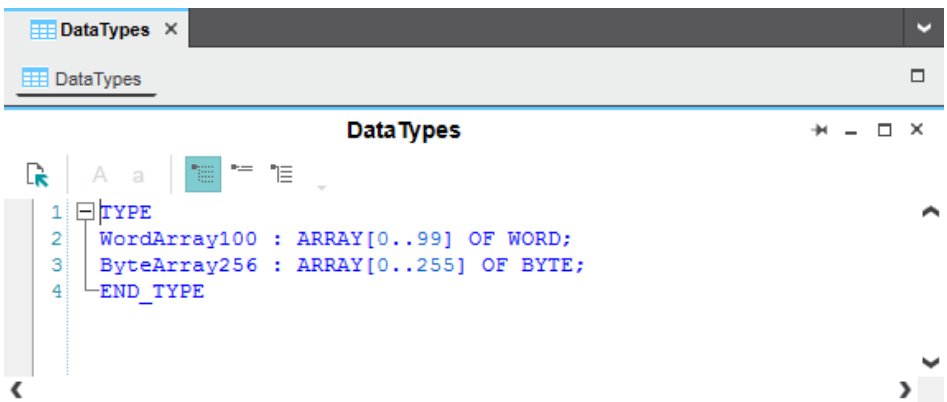

Figure 8 Data type worksheet with two ARRAY data type fields created (in the example: word array and byte array)

6 Creating and opening the POU, creating variables

To use the IB_CONTROL_NEXT function block, you need to create a corresponding program. The program is created within a POU.

6.1 Creating a POU

To create a new POU, proceed as follows:

- In the “COMPONENTS” area, open the “Programming (x), Local (x), Programs (x)” section.
- Right-click to open the context menu and select “Add Program”.

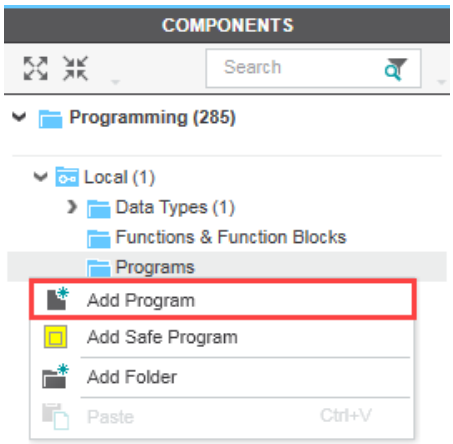

Figure 9 “Add Program” in the context menu

The “NewProgram” POU is displayed under the “Programming (x), Local (x), Programs (x)” section in the “COMPONENTS” area.

Recommended:

- Rename the newly inserted “NewProgram” POU.
- To do this, right-click on “NewProgram”.
- In the context menu, select “Rename”.
- Enter a unique and meaningful name for the POU.
- Press the ENTER key to apply the new name.

Please note the following for projects with AXC F 2152 controllers:
If you have created the project with the “Empty AXC F 2152 v00 / 2019.3.0 project” project template, the “Main” POU is already created by default.

6.2 Opening the POU

To open a POU, proceed as follows:

- In the “COMPONENTS” area, open the “Programming (x), Local (x), Programs (x)” section.
- Double-click on the desired POU – in the example in Figure 10: “Main”.

The editor group for the selected POU opens. When you open a POU for the first time, you are prompted to select the programming language for the first worksheet of the POU.

Programming is performed using the structured text (ST) programming language.

- Click on “Add ST Code Worksheet”.

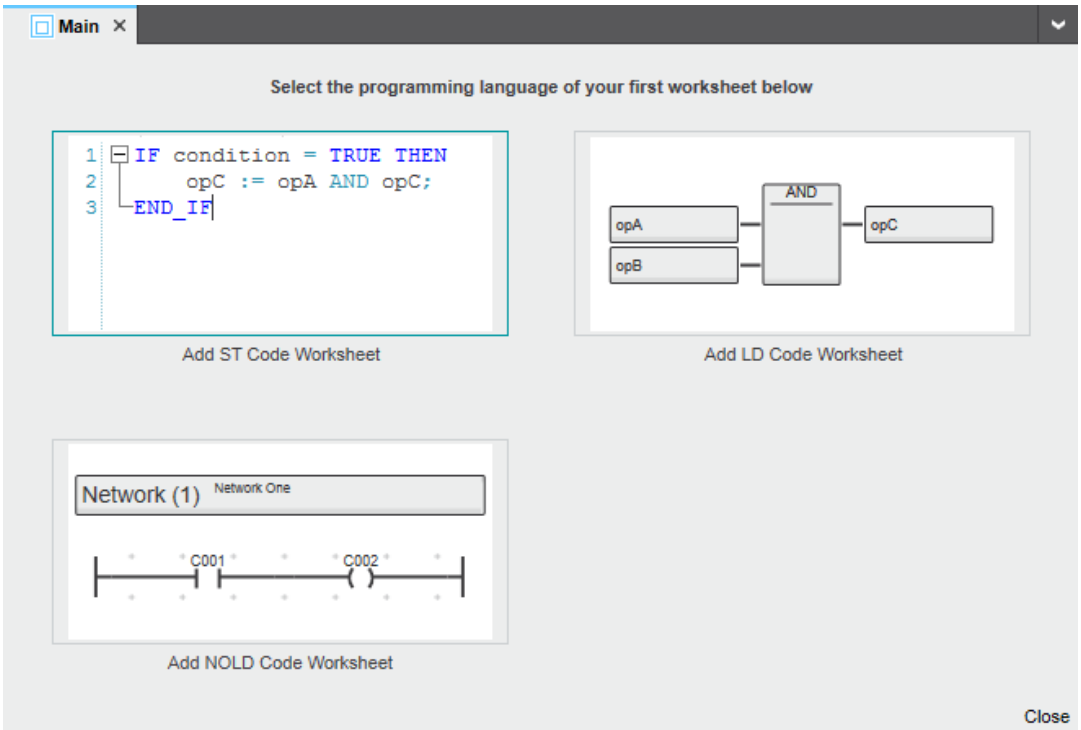

Figure 10 Selecting the programming language for the first worksheet

Once you have selected the programming language, the POU editor group opens. You can now create the necessary variables.

6.3 Creating variables

- Select the “Variables” editor.
- Create all the necessary variables.

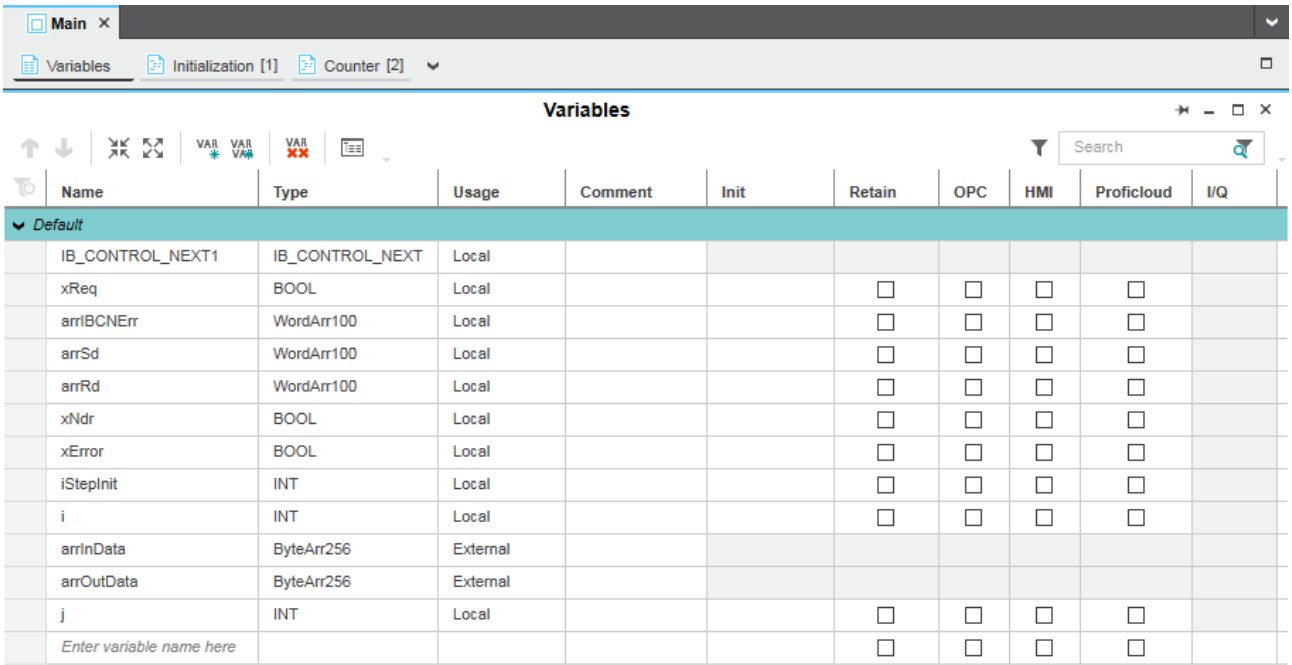

Figure 11 Creating variables for a POU

The variables shown in Figure 11 are used for the example program in this application note. The table below shows how the variables are used in the example program:

Variable	Data type	Use in the example program
IB_CONTROL_NEXT1	IB_CONTROL_NEXT	Instantiation of the IB_CONTROL_NEXT function block
xReq	BOOL	At the REQ input of the IB_CONTROL_NEXT function block
arrIBCNErr	WordArray100	At the ADD_ERROR input/output of the IB_CONTROL_NEXT function block
arrSd	WordArray100	At the SD_1 input/output of the IB_CONTROL_NEXT function block
arrRd	WordArray100	At the RD_1 input/output of the IB_CONTROL_NEXT function block
xNdr	BOOL	At the NDR output of the IB_CONTROL_NEXT function block
xError	BOOL	At the ERROR output of the IB_CONTROL_NEXT function block
iStepInit	INT	Specification of cases in the CASE instruction
i	INT	Iterator for FOR loop
arrInData[1]	ByteArray256	For receiving INTERBUS process data
arrOutData[1]	ByteArray256	For sending INTERBUS process data
j	INT	Iterator for IF condition

[1] This variable must be created as an external variable or an IN/OUT port in PLCnext Engineer.

Once you have created all of the necessary variables, create the program for the selected POU.

7 Creating a program

7.1 Adding code worksheets and renaming editors

Recommended:
To ensure the program is as clear as possible, create distinct code worksheets for the individual program components. Each worksheet is inserted in the POU editor group as an additional “Code” editor.

- Change the designation of each “Code” editor in order to clearly distinguish between the individual worksheets.

- Select the “Code” editor.

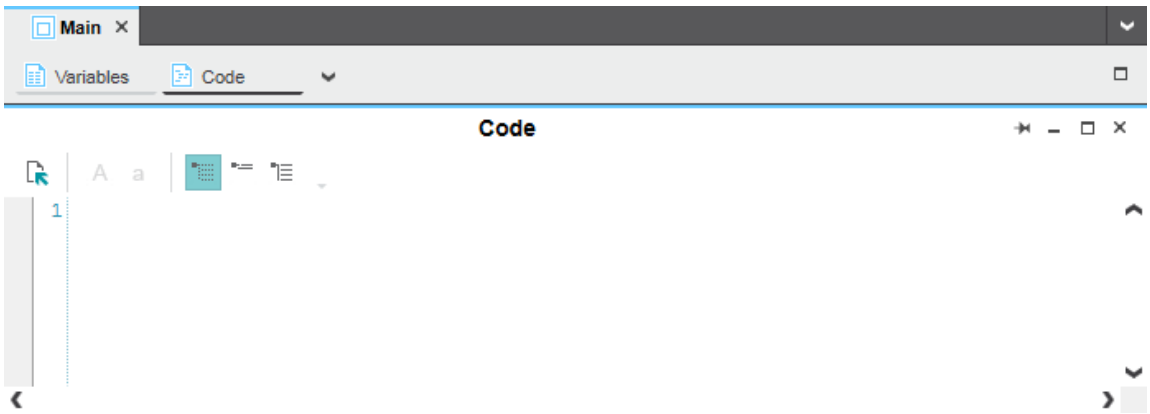

Figure 12 “Main” editor group, “Code” editor

- Change the designation of the “Code” editor.
  To do this, right-click to open the context menu and select “Rename”.

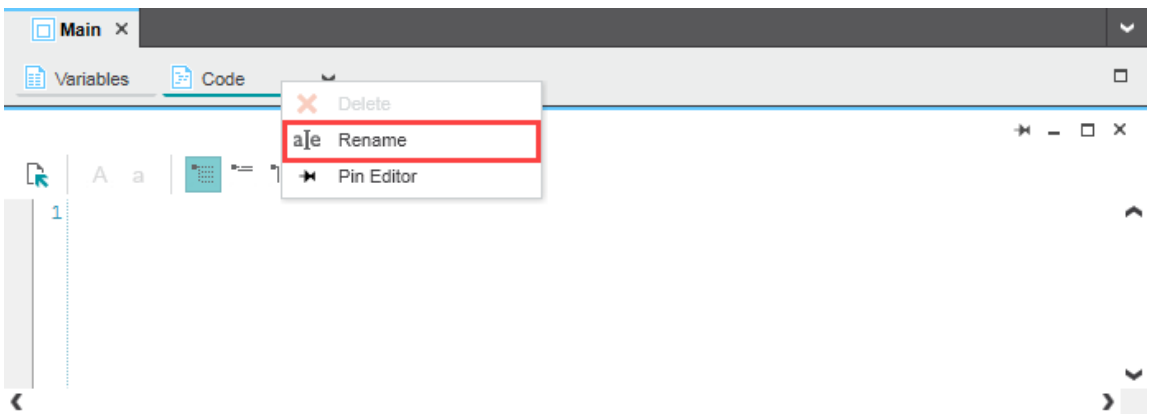

Figure 13 “Code” editor, “Rename” context menu

- Enter a unique and meaningful designation for the editor (in the example in Figure 14: “Initialization”).
- Press the ENTER key to apply the new designation.

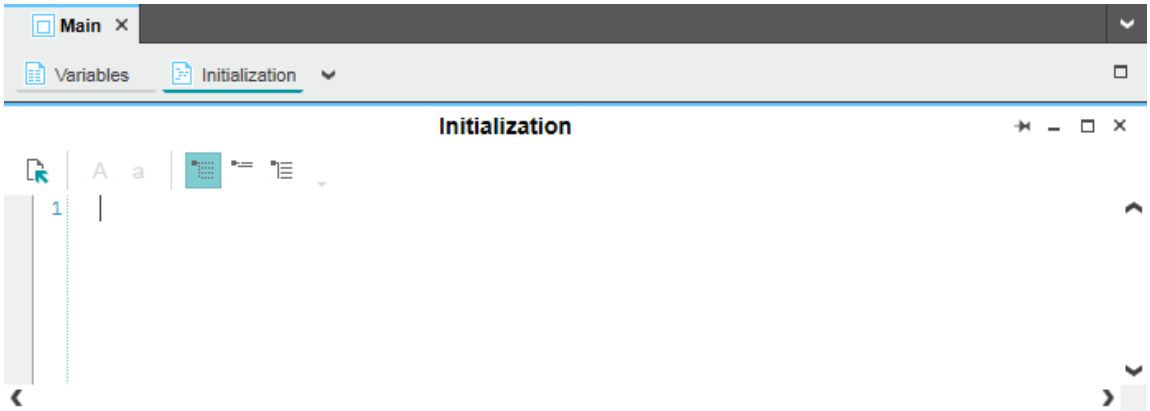

Figure 14 New designation of the “Code” editor

- Insert all the necessary worksheets in the POU.
- To do this, click on the arrow on the right next to the designation of the editor.
- From the drop-down list that opens, select the desired code worksheet.
- Change the designation of each newly inserted “Code” editor.

7.2 Programming INTERBUS startup

Once you have inserted all the necessary worksheets and created all the necessary data types, program INTERBUS startup in structured text (ST) programming language using the IB_CONTROL_NEXT function block.

Use Generation 4 (G4) INTERBUS firmware services for programming.

For information on Generation 4 INTERBUS firmware services, please refer to the IBS SYS FW G4 UM E user manual. The user manual can be downloaded at phoenixcontact.net/products.

- To start up INTERBUS, follow the step-by-step instructions provided in Sections 7.2.1 ... 7.2.4.

Figure 15 shows the individual programming steps.

- Adhere to the sequence of the sections when programming.

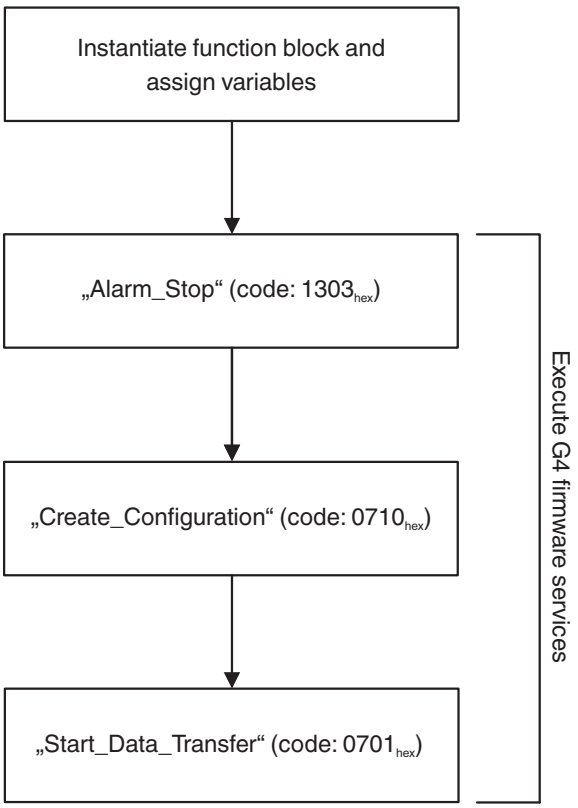

Figure 15 Programming steps for INTERBUS startup

Please note:

The example codes in Sections 7.2.1 ... 7.2.4 are **extracts** from the example program used in this application note. You cannot create an executable program using the example codes.
The complete example program including CASE instructions can be found in Section 10.

7.2.1 Instantiating the IB_CONTROL_NEXT function block and assigning variables

- Instantiate the IB_CONTROL_NEXT function block.
- Assign the corresponding variables to the inputs and outputs of the function block.

Example code: Instantiation of the IB_CONTROL_NEXT function block and variable assignment to the inputs/outputs of the function block

```
IB_CONTROL_NEXT1(   REQ         := xReq,
                    ADD_ERROR   := arrIBCNErr,
                    SD_1        := arrSd,
                    RD_1        := arrRd);
xNdr    := IB_CONTROL_NEXT1.NDR;
xError  := IB_CONTROL_NEXT1.ERROR;
```

7.2.2 Executing the “Alarm_Stop” G4 firmware service (code: 1303_{hex})

Once you have instantiated the IB_CONTROL_NEXT function block and assigned the variables to the inputs/outputs, you need to reset INTERBUS.

- To do this, execute the “Alarm_Stop” G4 firmware service (code: 1303_{hex}).

After having executed “Alarm_Stop”, INTERBUS is in the READY state.

Example code: Executing the “Alarm_Stop” G4 firmware service

```
arrSd[0] := WORD#16#1303; (* Write the “Alarm_Stop” service to the transmit buffer *)
xReq    := TRUE; (* Start function block (input REQ = TRUE) *)
IF xNdr THEN (* Service has been executed *)
   xReq := FALSE; (* Stop function block *)
   IF  arrRd[0] = WORD#16#9303 (* Alarm_Stop_Confirmation (code 9303hex) *)
   AND arrRd[1] = WORD#16#0001 (* Parameter_Count (code 0001hex) *)
   AND arrRd[2] = WORD#16#0000 THEN (* Result (code 0000hex) *)
       FOR i := 0 TO 99 DO
           arrRd[i] := WORD#16#0000; (* Delete receive buffer *)
           arrSd[i] := WORD#16#0000; (* Delete transmit buffer *)
       END_FOR
   END_IF;
END_IF;
```

7.2.3 Executing the “Create_Configuration” G4 firmware service (code: 0710_{hex})

To start INTERBUS, you first need to create a valid configuration frame.

- To do this, execute the “Create_Configuration” G4 firmware service (code: 0710_{hex}).

After having executed “Create_Configuration”, INTERBUS is in the ACTIVE state.

Example code: Executing the “Create_Configuration” G4 firmware service

```
arrSd[0] := WORD#16#0710; (* Write the “Create_Configuration” service to the transmit buffer *)
arrSd[1] := WORD#16#0001; (* Parameter_Count (code 0001hex) *)
arrSd[2] := WORD#16#0001; (* Frame_Reference of the configuration frame to be created *)
xReq := TRUE; (* Start function block (input REQ = TRUE) *)
IF xNdr THEN (* Service has been executed *)
   xReq := FALSE; (* Stop function block *)
   IF arrRd[0]  = WORD#16#8710 (* Create_Configuration_Confirmation (code 8710hex) *)
   AND arrRd[1] = WORD#16#0001 (* Parameter_Count (code 0001hex) *)
   AND arrRd[2] = WORD#16#0000 THEN (* Result (code 0000hex) *)
       FOR i := 0 TO 99 DO
           arrRd[i] := WORD#16#0000; (* Delete receive buffer *)
           arrSd[i] := WORD#16#0000; (* Delete transmit buffer *)
       END_FOR
   END_IF;
END_IF;
```

7.2.4 Executing the “Start_Data_Transfer” G4 firmware service (code: 0701_{hex})

Once you have created a valid configuration frame, you can activate cyclic data transfer on INTERBUS and start INTERBUS.

- To do this, execute the “Start_Data_Transfer” G4 firmware service (code: 0701_{hex}).

Example code: Executing the “Start_Data_Transfer” G4 firmware service

```
arrSd[0] := WORD#16#0701; (* Write the “Start_Data_Transfer” service to the transmit buffer *)
xReq := TRUE; (* Start function block (input REQ = TRUE) *)
IF xNdr THEN (* Service has been executed *)
   xReq := FALSE; (* Stop function block *)
   IF arrRd[0]  = WORD#16#8701 (* Start_Data_Transfer_Confirmation (code 8701hex) *)
   AND arrRd[1] = WORD#16#0001 (* Parameter_Count (code 0001hex) *)
   AND arrRd[2] = WORD#16#0000 THEN (* Result (code 0000hex) *)
       FOR i := 0 TO 99 DO
           arrRd[i] := WORD#16#0000; (* Delete receive buffer *)
           arrSd[i] := WORD#16#0000; (* Delete transmit buffer *)
       END_FOR
   END_IF;
END_IF;
```

7.3 Sending and receiving INTERBUS process data

Once you have started INTERBUS, you need to create a program to send and receive the INTERBUS process data.

To send and receive process data, you first need to link a byte array or word array to the input process data and the output process data of the “IB 256” module.
In the example in Figure 16, the “arrInData” and “arrOutData” byte arrays from Section 6.3 are used.

To link the arrays to the input and output process data of the “IB 256” module, proceed as follows:

- Double-click on the “ib-1 : IB 256” node in the “PLANT” area.

The “IB 256” module editor group opens.

- Select the “Data List” editor.

You can see the process data items of the “IB 256” module in the “Data List” editor.

- To assign a variable to a process data item, click on “Select Variable (PLC) here” in the “Variable (PLC)” column.

The role picker opens.

- Assign the corresponding byte or word array to the process data items of the “IB 256” module.
  To do this, select the relevant variable in the role picker.

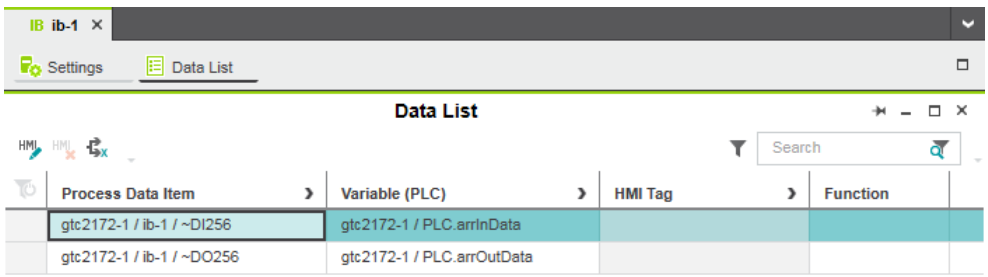

Figure 16 Assignment of the “arrInData” and “arrOutData” variables to the input and output process data items of the “IB 256” module

Please note:

If you have created the byte or word arrays for sending and receiving INTERBUS process data as IN or OUT port variables, the process data is **not** assigned in the “Data List” editor.
The process data is assigned in the “Port List” editor for IN or OUT port variables.
To open the “Port List” editor, proceed as follows:

- Double-click on the “PLCnext” node in the “PLANT” area.

The “/ PLCnext” editor group opens.

- Select the “Port List” editor.
- Assign the process data.

There are two ways to assign process data:

1 You can assign an IN port to an OUT port:
  - Click on “Select IN Port here” in the “IN Port” column.
  - Select the IN port that you want to assign to the relevant OUT port in the role picker.

2 You can assign an OUT port to an IN port:
  - Click on “Select OUT Port here” in the “OUT Port” column.
  - Select the OUT port that you want to assign to the relevant IN port in the role picker.

Once you have linked the process data items, you need to determine the location of the input and/or output process data of each connected INTERBUS remote bus device in the corresponding byte or word array.

The location of the input and/or output process data of each connected INTERBUS remote bus device in the byte or word array depends on:

– The position of the relevant INTERBUS remote bus device in the bus configuration
– The input and/or output process data width of the relevant INTERBUS remote bus device

To determine the location of the input and/or output process data in the byte or word arrays, proceed as follows:

- Steadily work through the bus configuration from left to right for all INTERBUS remote bus devices.
- The process data width of each INTERBUS remote bus device can be found in the device-specific data sheet.

The data sheets can be downloaded via the product page at phoenixcontact.net/products.

- Determine the location of the input and/or output process data based on the position of each INTERBUS remote bus device in the bus configuration and the relevant process data width.

The input and/or output process data of the INTERBUS remote bus devices is assigned to the byte or word arrays in ascending order based on the position of the INTERBUS remote bus device in the bus configuration (from left to right) (see example on Page 20 and onwards).

Figure 17 shows how the input and output process data is assigned to the “arrInData” byte array and the “arrOutData” byte array for an example bus configuration.

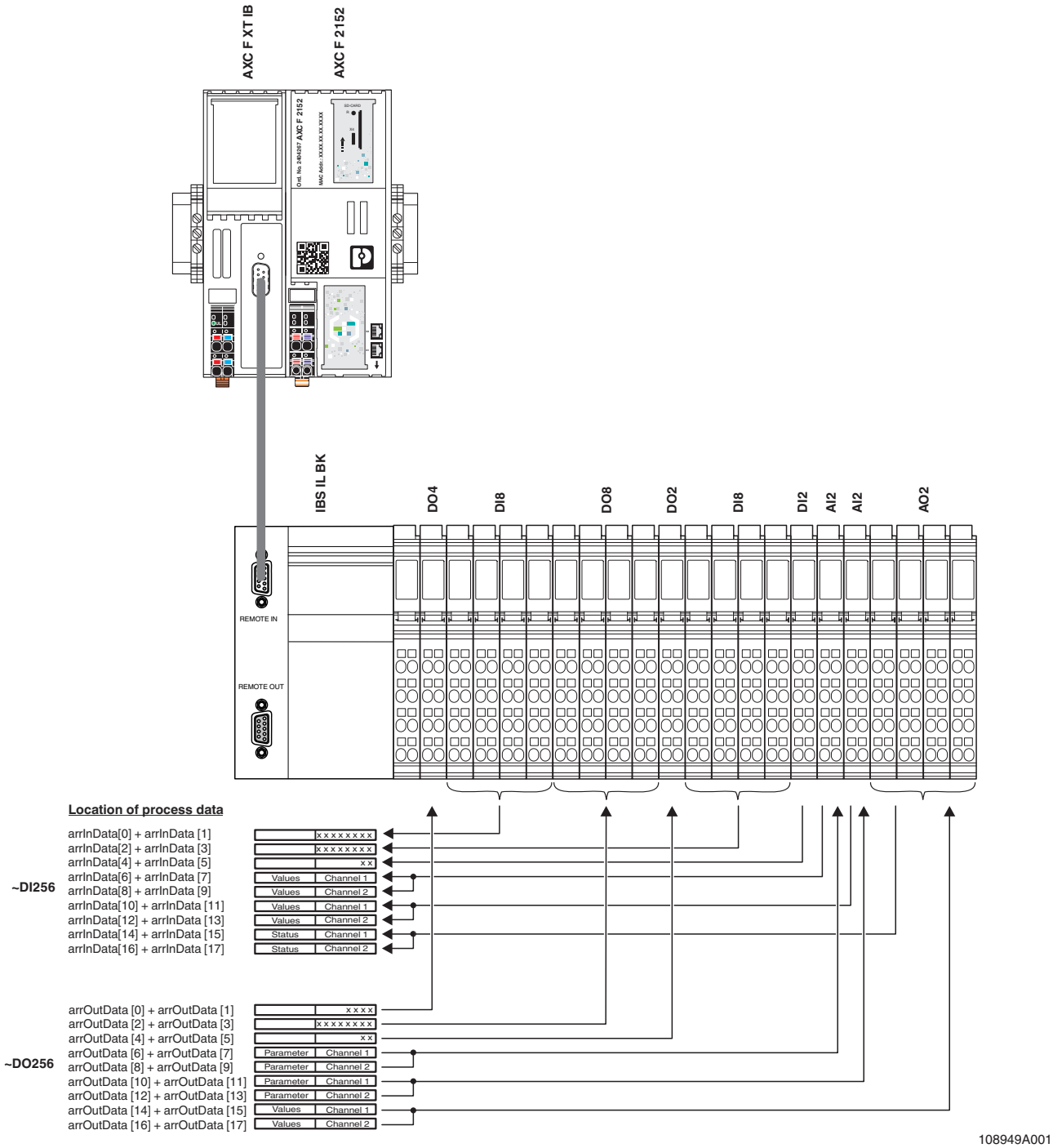

Figure 17 Example: Assignment of the input and output process data to the “arrInData” and “arrOutData” byte arrays

Explanation of the example bus configuration in Figure 17

Table 1 Input and output process data width of the INTERBUS remote bus devices in the example

INTERBUS remote bus device in the bus configuration (from left to right)	Input process data width according to device-specific data sheet	Output process data width according to device-specific data sheet
IB IL 24 DO 4-PAC (Order No. 2861276)	–	4 bits
IB IL 24 DI 8-PAC (Order No. 2861247)	8 bits	–
IB IL 24 DO 8-PAC (Order No. 2861289)	–	8 bits
IB IL 24 DO 2-PAC (Order No. 2861470)	–	2 bits
IB IL 24 DI 8-PAC (Order No. 2861247)	8 bits	–
IB IL 24 DI 2-PAC (Order No. 2861221)	2 bits	–
IB IL AI 2/4-20-PAC (Order No. 2862217)	32 bits	32 bits
IB IL AI 2/4-20-PAC (Order No. 2862217)	32 bits	32 bits
IB IL AO 2/SF-PAC (Order No. 2863083)	32 bits	32 bits

The input and/or output process data of the INTERBUS remote bus devices is assigned to the elements of the “arrInData” byte array and the elements of the “arrOutData” byte array in ascending order based on the position of the INTERBUS remote bus device in the bus configuration (from left to right):

Table 2 Assignment of the input/output process data to the “arrInData” and “arrOutData” byte arrays

INTERBUS remote bus device in the bus configuration (from left to right)	Location of the input process data in the “arrInData” byte array	Location of the output process data in the “arrOutData” byte array
IB IL 24 DO 4-PAC (Order No. 2861276)	–	arrOutData[0] + arrOutData[1] (bits: 00000000 + 0000XXXX)
IB IL 24 DI 8-PAC (Order No. 2861247)	arrInData[0] + arrInData[1] (bits: 00000000 + XXXXXXXX)	–
IB IL 24 DO 8-PAC (Order No. 2861289)	–	arrOutData[2] + arrOutData[3] (bits: 00000000 + XXXXXXXX)
IB IL 24 DO 2-PAC (Order No. 2861470)	–	arrOutData[4] + arrOutData[5] (bits: 00000000 + 000000XX)
IB IL 24 DI 8-PAC (Order No. 2861247)	arrInData[2] + arrInData[3] (bits: 00000000 + XXXXXXXX)	–
IB IL 24 DI 2-PAC (Order No. 2861221)	arrInData[4] + arrInData[5] (bits: 00000000 + 000000XX)	–
IB IL AI 2/4-20-PAC (Order No. 2862217)	arrInData[6] + arrInData[7] (bits: XXXXXXXX + XXXXXXXX) arrInData[8] + arrInData[9] (bits: XXXXXXXX + XXXXXXXX)	arrOutData[6] + arrOutData[7] (bits: XXXXXXXX + XXXXXXXX) arrOutData[8] + arrOutData[9] (bits: XXXXXXXX + XXXXXXXX)

Table 2 Assignment of the input/output process data to the “arrInData” and “arrOutData” byte arrays

INTERBUS remote bus device in the bus configuration (from left to right)	Location of the input process data in the “arrInData” byte array	Location of the output process data in the “arrOutData” byte array
IB IL AI 2/4-20-PAC (Order No. 2862217)	arrInData[10] + arrInData[11] (bits: XXXXXXXX + XXXXXXXX) arrInData[12] + arrInData[13] (bits: XXXXXXXX + XXXXXXXX)	arrOutData[10] + arrOutData[11] (bits: XXXXXXXX + XXXXXXXX) arrOutData[12] + arrOutData[13] (bits: XXXXXXXX + XXXXXXXX)
IB IL AO 2/SF-PAC (Order No. 2863083)	arrInData[14] + arrInData[15] (bits: XXXXXXXX + XXXXXXXX) arrInData[16] + arrInData[17] (bits: XXXXXXXX + XXXXXXXX)	arrOutData[14] + arrOutData[15] (bits: XXXXXXXX + XXXXXXXX) arrOutData[16] + arrOutData[17] (bits: XXXXXXXX + XXXXXXXX)

Table 3 shows the assignment of the input/output process data for the example bus configuration when word arrays are used.

Table 3 Assignment of the input/output process data to word arrays

INTERBUS remote bus device in the bus configuration (from left to right)	Location of the process data in the word array at the RD_1 input/output	Location of the process data in the word array at the SD_1 input/output
IB IL 24 DO 4-PAC (Order No. 2861276)	–	WORD 0
IB IL 24 DI 8-PAC (Order No. 2861247)	WORD 0	–
IB IL 24 DO 8-PAC (Order No. 2861289)	–	WORD 1
IB IL 24 DO 2-PAC (Order No. 2861470)	–	WORD 2
IB IL 24 DI 8-PAC (Order No. 2861247)	WORD 1	–
IB IL 24 DI 2-PAC (Order No. 2861221)	WORD 2	–
IB IL AI 2/4-20-PAC (Order No. 2862217)	WORD 3 WORD 4	WORD 3 WORD 4
IB IL AI 2/4-20-PAC (Order No. 2862217)	WORD 5 WORD 6	WORD 5 WORD 6
IB IL AO 2/SF-PAC (Order No. 2863083)	WORD 7 WORD 8	WORD 7 WORD 8

Once you have determined the location of the input and/or output process data of each connected INTERBUS remote bus device in the byte or word array, create the program:

- Open the corresponding worksheet in the relevant POU.
- Create the program.

Example program: Writing output process data

In the example program, the output process data of the first INTERBUS remote bus device is written from the example bus configuration in Figure 17.

First INTERBUS remote bus device in the example bus configuration: IB IL 24 DO 4-PAC (Order No. 2861276)

Output process data length: 4 bits

Location of the process data: arrOutData[0] + arrOutData[1] (bits: 00000000 + 0000XXXX)

The output process data is in bits 1 ... 4 in the second element of the “arrOutData” byte array.

Example code for writing bits 1 ... 4 of the IB IL 24 DO 4-PAC INTERBUS remote bus device:

```
IF j < 15 THEN
    arrOutData[1] := TO_BYTE(j);
    j := j + 1;
ELSE
    j := 0;
END_IF
```

The 4 bits to be written are incremented by variable j. This means that the relevant bit is set in the second element of the “arrOutData” byte array. The corresponding LEDs on the INTERBUS remote bus device light up one after the other creating a running light.
The “IB 256” module expects a word or byte array as the transfer value. In the example, a byte array is used. Since variable j is an INT-type variable, the integer value needs to be converted accordingly. The TO_BYTE function is used for this.

Reading input process data

A word or byte array is also used to read input process data. The “arrInData” byte array is used for the example in this application note.

8 Instantiating a program

Once you have created the program, you need to instantiate it.

- Instantiate the program as described in the user manual for the controller you are using.

9 Transferring a project to the controller

Once you have instantiated the program, you need to transfer the project to the controller. PLCnext Engineer must be connected to the controller in order to do this.

- Connect PLCnext Engineer to the controller as described in the user manual for the controller you are using.
- Transfer the project to the controller as described in the user manual for the controller you are using.

10 Complete example program

Once you have created the variables shown in the example in Section 6.3 on page 12 for your PLCnext Engineer project, you can use the following example program for INTERBUS startup.

- If you would like to use the example program, copy the example program to the desired POU in your PLCnext Engineer project.

```
IB_CONTROL_NEXT1(   REQ         := xReq,
                    ADD_ERROR   := arrIBCNErr,
                    SD_1        := arrSd,
                    RD_1        := arrRd);

xNdr    := IB_CONTROL_NEXT1.NDR;
xError  := IB_CONTROL_NEXT1.ERROR;

CASE iStepInit OF
(* Alarm_Stop (CODE 1303hex) *)
        0: arrSd[0] := WORD#16#1303;
           xReq    := TRUE;
           IF xNdr THEN
               xReq := FALSE;
               IF  arrRd[0] = WORD#16#9303
               AND arrRd[1] = WORD#16#0001
               AND arrRd[2] = WORD#16#0000 THEN
                   iStepInit := 10;
                   FOR i := 0 TO 99 DO
                       arrRd[i] := WORD#16#0000;
                       arrSd[i] := WORD#16#0000;
                   END_FOR
               ELSE
                   iStepInit := 1000;
               END_IF;
           END_IF;

(* Create_Configuration (CODE 0710hex) *)
        10: arrSd[0] := WORD#16#0710;
            arrSd[1] := WORD#16#0001;
            arrSd[2] := WORD#16#0001;
            xReq := TRUE;
            IF xNdr THEN
                xReq := FALSE;
                IF arrRd[0]  = WORD#16#8710
                AND arrRd[1] = WORD#16#0001
                AND arrRd[2] = WORD#16#0000 THEN
                    iStepInit := 20;
                    FOR i := 0 TO 99 DO
                        arrRd[i] := WORD#16#0000;
                        arrSd[i] := WORD#16#0000;
                    END_FOR
                ELSE
                    iStepInit := 1010;
                END_IF;
            END_IF;

(* Start_Data_Transfer (CODE 0701hex) *)
        20: arrSd[0] := WORD#16#0701;
            xReq := TRUE;
            IF xNdr THEN
                xReq := FALSE;
                IF arrRd[0]  = WORD#16#8701
```

```
                AND arrRd[1] = WORD#16#0001
                AND arrRd[2] = WORD#16#0000 THEN
                    iStepInit := 30;
                    FOR i := 0 TO 99 DO
                        arrRd[i] := WORD#16#0000;
                        arrSd[i] := WORD#16#0000;
                    END_FOR
                ELSE
                    iStepInit := 1020;
                END_IF;
            END_IF;
        END_CASE;
```