Creating a device package for code Creator (v2.0 and up)

Femke Parthoens

12 September 2024

Contents

1 Introduction to Code Creator **4**

2 What is a device package **4**

3 What is in a device package **4**

3.1 Source folder 4

3.1.1 DataConversionFunctionBlocks 5

3.1.2 DataTypeSheets 5

3.1.3 SelectedFunctionBlocks 6

3.2 Desscription.XML 7

3.2.1 metadata - line 1 through 20 7

3.2.2 Read transactions 8

3.2.3 Write transactions 9

3.2.4 Rest of the XMl 9

4 Practical Guidelines - Do's and Dont's **10**

4.1 STEP 1: Select an existing package as a base 10

4.2 STEP 2: Update the XML file 10

4.3 STEP3: Upload the new Device package and generate a PLC project 10

4.4 STEP4: Open the project in PLCNextEngineer 11

4.4.1 DataTypeSheet 12

4.4.2 Functions and function blocks 12

4.5 STEP 5: Update the Device package 14

4.6 STEP 6: Upload the Device package and generate the PLC project once more 15

List of Figures

1 Contents of the Source folder . . . 4
2 DataConversionFunctionBlocks . . . 5
3 DataTypeSheets . . . 5
4 SelectedFunctionBlocks . . . 6
5 XML Lines 1 through 20 . . . 7
6 Read Transcation . . . 8
7 Write Transaction . . . 9
8 End of the document . . . 9
9 Example project . . . 11
10 ConversionMethod_FC3 . . . 12
11 Calculation tab . . . 12
12 GetResult_To_Real_1 . . . 12
13 Define data type in method . . . 13
14 Define data type in datatypesheet . . . 13
15 SelectFunctionBlock . . . 13
16 PLCnext Engineer project opened as an archive . . . 14
17 CHARX project functionblocks . . . 14

List of Tables

1 Revision History . . . 3

List of abbreviations

Eq. Equation
Fig. Figure
Rev. Revision

Revision Table

Table 1: Revision History

Date	Version	Description of Changes	Revised by
17-07-2023	0.0	Documentation for version 1.2	Thomas Oorts
12-09-2024	1.0	Initial draft	Femke Parthoens

1 Introduction to Code Creator

With the Code Creator, IEC 611131 code for PLCnext Engineer can be generated for setting up a modbus RTU or Modbus TCP connection. Currently, the FC3, FC6, and FC16 features are supported.

In the Code Creator, devices are made available with the help of device packages. In the code creator the desired registers only need to be checked and the Code Creator itself will generate the necessary code to read this data from the device and convert it into the a standard data type, unit and format order.
The device packages contain all the necessary information about the device and the transformation methods that are needed to generate the necessary code with the Code Creator.

2 What is a device package

A Device package is essentially the input that code creator needs to compile the code for a modbus program to a certain device. This document should serve as a guidline and a convention document. As an example we will take the code creator package of a CHARX module set of two.

3 What is in a device package

A device package is a zipfile. The name of this zip file should follow the format:
[Vendor]_[Product]_[Rev.]_[RTU/TCP/RTU_TCP]
In this file there should be three files present:

1. An XML File that contains all the information for the registers and the function codes.
2. A pdf that serves as a manual for the device
3. A Source folder the functionblocks and data types

3.1 Source folder

The Source folder contains all the Functionblblocks and datatypesheets needed to automatically generate a PLC Project in Code Creator. In the Source folder are typically three subfolders:

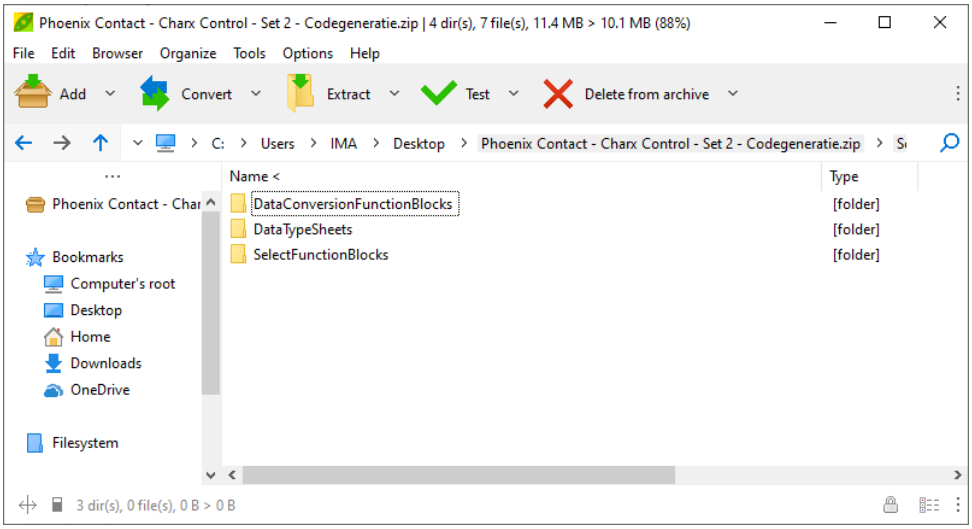


Figure 1: Contents of the Source folder

- DataConversionFunctionBlocks
- DataTypeSheets
- SelectfunctionBlocks

3.1.1 DataConversionFunctionBlocks

This folder should contain the dataconversionfunctionblock. For each Read function code needed, there should be one present.

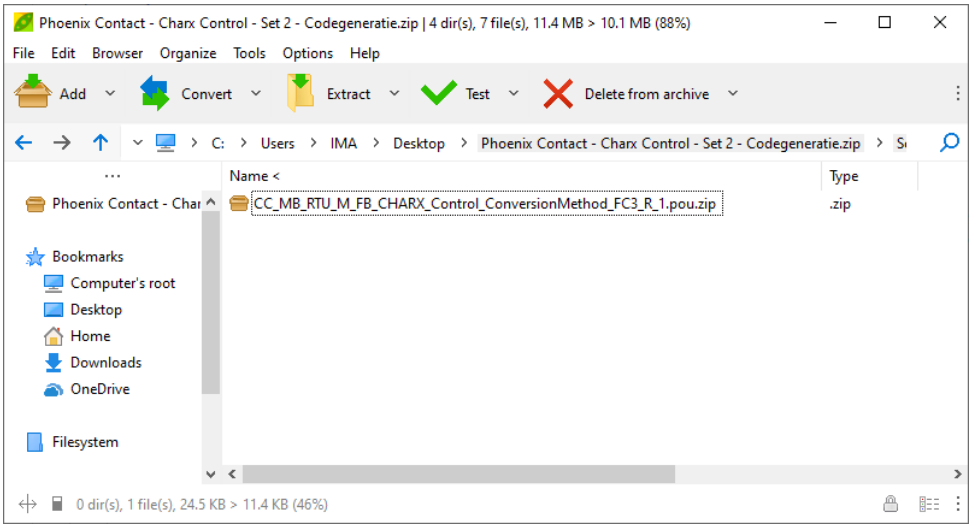


Figure 2: DataConversionFunctionBlocks

3.1.2 DataTypeSheets

This folder should contain all the datasheets needed to automatically generate a PLC project.

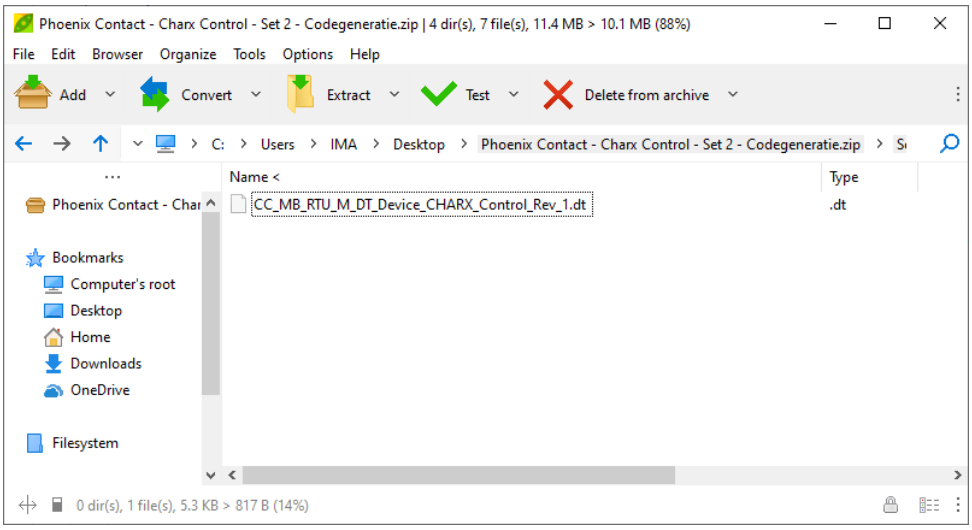


Figure 3: DataTypeSheets

3.1.3 SelectedFunctionBlocks

This folder contains all the functionblocks needed for the write transactions to the device.

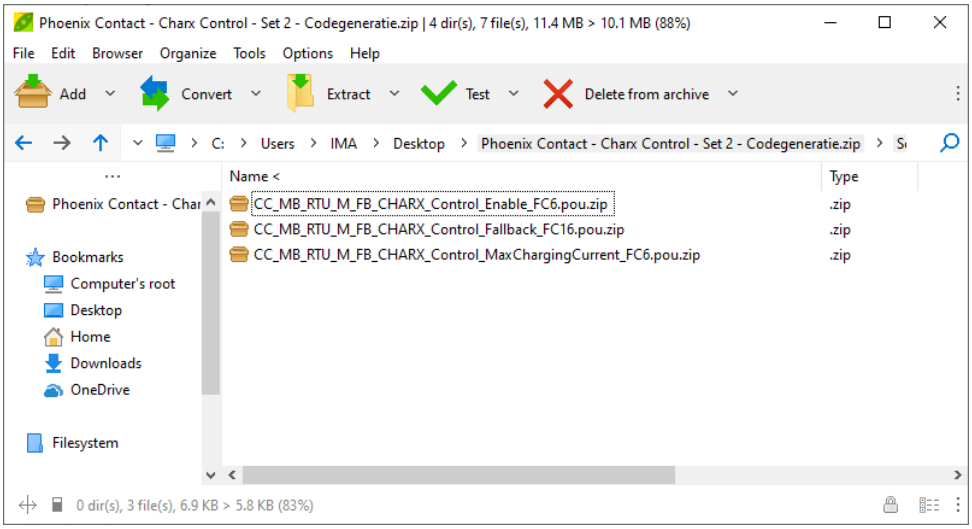


Figure 4: SelectedFunctionBlocks

3.2 Desscription.XML

3.2.1 metadata - line 1 through 20

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <Device xmlns="http://phoenixcontact.com/code-creator/modbus" type="ModbusTCP">
3      <Id>009ae121-724a-4012-9eb4-aa0e3e3c1424</Id>
4      <Name>Charx_Control_Socket</Name>
5      <DisplayName>PhoenixContact_CharxControlSet2_Rev.14032023_TCP</DisplayName>
6      <Manufacturer>Phoenix Contact</Manufacturer>
7      <Article>1139022</Article>
8      <FileRevision>1</FileRevision>
9      <RequiredComponent>MB-TCP Master</RequiredComponent>
10     <Type>Charx_Control_Socket</Type> <!--   Direct input for the transaction and dataconversion function block   -->
11     <DataConversionFunctionBlocks>
12         <DataConversionFunctionBlock>
13             <FunctionCode>3</FunctionCode>
14             <SourcePath>Source/DataConversionFunctionBlocks/CC_MB_RTU_M_FB_CHARX_Control_ConversionMethod_FC3_R_1.pou.zip</SourcePath>
15         </DataConversionFunctionBlock>
16     </DataConversionFunctionBlocks>
17     <DataTypeSheets>
18         <DataTypeSheet>Source/DataTypeSheets/CC_MB_RTU_M_DT_Device_CHARX_Control_Rev_1.dt</DataTypeSheet> <!--   Datatype sheet with th
19     </DataTypeSheets>
20     <ManualLocation>um_en_charx_sec_109999_en_02.pdf</ManualLocation>
```

Figure 5: XML Lines 1 through 20

- Line 1: Encoding, is always version 1.0, and UTF-8
- Line 2: Namespace declaration(fixed) and devicetype (ModbusRTU or ModbusTCP)
- Line 3: Id should be a Version 4 UUID
- Line 4: Name of the device
- Line 5: Displayname: This is the name that will be suggested when dragging and dropping the device in code Creator
- Line 6: Manufacturer
- Line 7: Article number reference provided by the manufacturer
- Line 8: Revision number, for revision tracking
- Line 9: Required Component: describes what program has to predeceed the device in Code Creator
- Line 10: Prefix for the transaction and dataconversion function blocksnames generated by Code Creator
- Line 11 through 16: The Path defined for the dataconversionfunctionblocks for read transaction, per type of function code, other function codes have to be added in the same way.
- Line 17 through 19: The path defined for the datatypesheet.
- Line 20: The Path to the manual

3.2.2 Read transactions

```
21  <DataGroups>
22      <DataGroup name="Module 1" structName="Module_1" module="1">
23          <Description>Module 1: Measurement and Control</Description>
24          <Transactions>
25              <Transaction readFunctionCode="3" dataGroupSuffix="_Measurement">
26                  <StartAddress>1232</StartAddress>
27                  <NumberOfRegisters>68</NumberOfRegisters>
28                  <DataPoints>
29                      <ParameterDataPoint isEnabledByDefault="True">
30                          <Description>Voltage Phase V(L1-N)</Description>
31                          <VariableName>strVoltagePhase_L1_N</VariableName>
32                          <DataType>strCC_MB_RTU_M_CHARX_Control_R1_DataResult_1</DataType>
33                          <DataTypeEndpoint identifier='Voltage_1N' unit='V' dataType='REAL'>.rScaledValue</DataTypeEndpoint>
34                          <DataConversion>
35                              <SetTaskMethodName>ConvertDatapoint</SetTaskMethodName>
36                              <GetResultMethodName>GetResult_To_Real_1</GetResultMethodName>
37                              <ConversionParameters>
38                                  <ConversionParameter index="1" description="Conversion type">1</ConversionParameter>
39                                  <ConversionParameter index="2" description="address offset">1</ConversionParameter>
40                                  <ConversionParameter index="3" description="factor">0.001</ConversionParameter>
41                                  <ConversionParameter index="4" description="unit">'V'</ConversionParameter>
42                              </ConversionParameters>
43                          </DataConversion>
44                      </ParameterDataPoint>
```

Figure 6: Read Transcation

- Line 21: Open <DataGroups>
- Line 22: Open the first <DataGroup>, and add a name. For this device, multiple devices will be reached via one IP address. The structnume and module number can be added, so that code creator can link the devices seperatly. One DataGroup per module (per device) will be added.
- Line 23: Description of the datagroup
- Line 24: Open the Transactions
- Line 25: Add the first transaction, define which pre-defined (line 11 through 16) read transaction is needed. and the name of the struct.
- Line 26 and 27: Startaddress of the modbus transaction, and number of registers to be readed in this single transaction.
- Line 28: Open <DataPoints>: in this section all the datapoints that can be found in this modbus call will be described and defined.
- Line 29: Should this datapoint be pre-selected in Code creator?
- Line 30: Display name off in code creator for the data point.
- Line 31: Name off the struct where the variable will end up in the generated PLC project.
- Line 32: Data type of the variable in the generated PLC solution
- Line 33: Metadata for the automatic linking off variables in Code Creator, the name of the expected variable, the unit that the datapoint will have in the generated program, and the pointer in the struct final struct.
- Line 34: Open the dataconversion for this datapoint.
- Line 35: Set the task name
- Line 36: The name of the method needed for the conversion.

- Line 37 through 44: Add the metadata for the conversion. Address offset means the how manyeth address, starting from the start address off the modbus transaction, is the start address for this datapoints. How many registers are counted for this datapoint is stored in the address offset of the next datapoint in the list. (If two are needed, the next parameterdatapoint starts at three).
  Close Conversionparameters, Dataconversion, And ParameterDataPoint.
- Line 38: Open the Next parameterdatapoint and continue
- When all the datapoints for the transaction were added, close with </DataPoints> and close the Transaction with </Transaction>

3.2.3 Write transactions

A lot of the info is similar toe the read transactions, mostly the differences will be highlighted in this section

```
438             <Transaction writeFunctionCode="6" dataGroupSuffix="_MaxChargingCurrent">
439                 <StartAddress>1301</StartAddress>
440                 <NumberOfRegisters>1</NumberOfRegisters>
441                 <DataPoints>
442                     <CommandDataPoint isEnabledByDefault="True">
443                         <Description>Maximum ChargingCurrent</Description>
444                         <VariableName>strMaxChargingCurrent</VariableName>
445                         <SelectFunctionBlockPath>Source/SelectFunctionBlocks/CC_MB_RTU_M_FB_CHARX_Control_MaxChargingCurrent_FC6.pou.zip</SelectFunctionBlockPath>
446                         <InPortControlStructDataType>strCC_MB_RTU_M_CHARX_Control_R1_MaxChargingCurrent</InPortControlStructDataType>
447                     </CommandDataPoint>
448                 </DataPoints>
449             </Transaction>
```

Figure 7: Write Transaction

- Open a new transaction, describe the needed write function code and the name to be generated in the PLC projet
- Add start- and number of registers for the transaction. Add the Description, and the name.
- CommandDataPoint, similar to ParameterDataPoint, but for a write transaction, instead of a read transaction
- Line 445: <SelectFunctionBlockPath> this is the function block needed to control the variable.
- Line 446: <InPortControlStructDataType> The struct created for the parameter.

3.2.4 Rest of the XMl

After adding all the read and write sections for one Module, the Transactions should be closed with </Transactions> and the DataGroup with </DataGroup>. If there is another module or datagroup add it in the same way as described above.
Then close the datagroups with </DataGroups> and end with </Device>.

```
904                 </Transactions>
905             </DataGroup>
906         </DataGroups>
907     </Device>
```

Figure 8: End of the document

4 Practical Guidelines - Do's and Dont's

To generate a device package, one can start and make the needed documents from scratch. But it is easier if you make a new device package based on an existing one. To do this, it is best to base the new package on the package of a device that is similar to the device you want to create a package for.

4.1 STEP 1: Select an existing package as a base

In a lot of cases, you'll need similar datatypes, conversionfunctionblocks and maybe even Select-FunctionBlocks. Copy it, and rename the zipfile for the new package.

4.2 STEP 2: Update the XML file

Update the XML, fill out the correct metadata and add the registers for your new device. Use the same functionblocks and datatypes that are already present in the folder.
Replace the manual in the folder, and update the path in the XML file.

4.3 STEP3: Upload the new Device package and generate a PLC project

Select all the read and write options, to check the communication.

4.4 STEP4: Open the project in PLCNextEngineer

Resolve all the errors if there are any. Add or adjust the datatypes and dataconversion function blocks if needed. (Re)write the Functionblocks for the write transactions.
Most of these functions, function blocks and data type sheets are the same for each project that is generated and therefore device independent. These should therefore not be put in a device package.

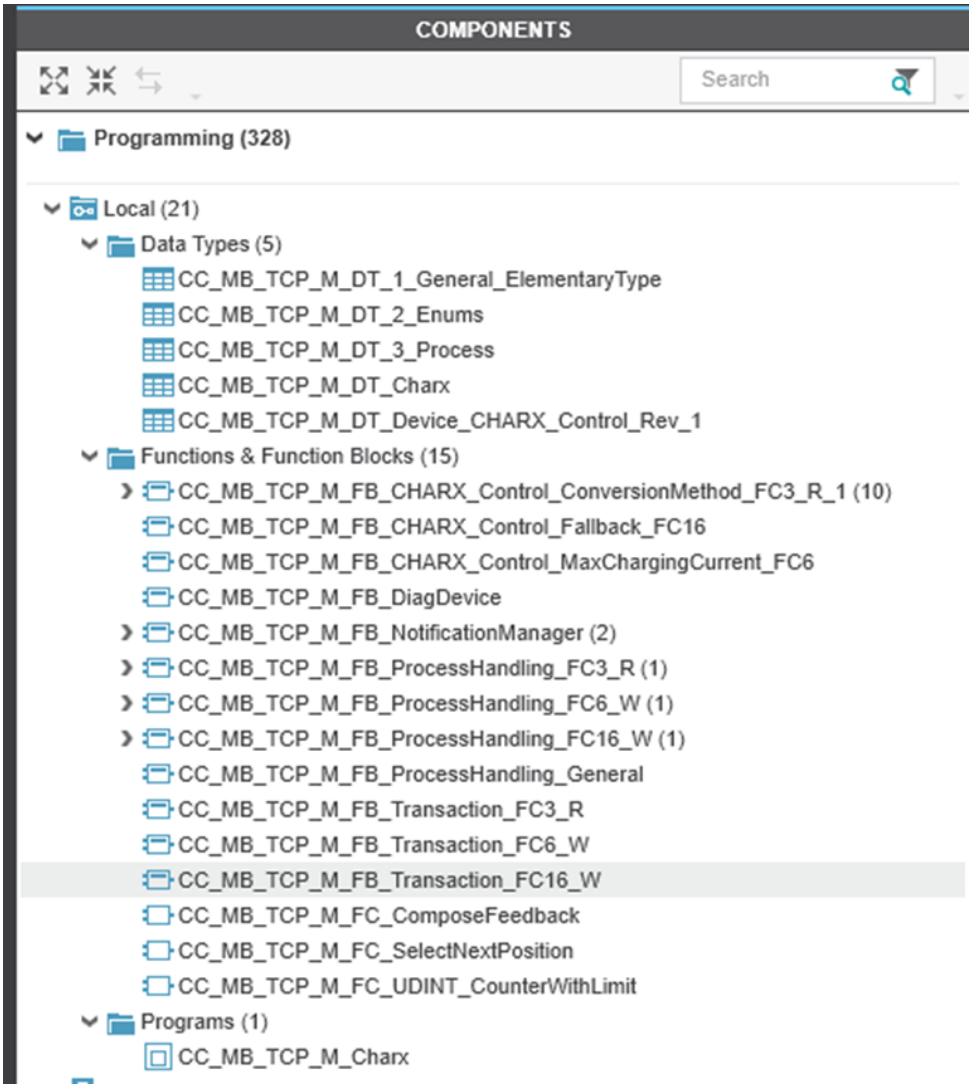


Figure 9: Example project

For example: in the above example to create a new device package, the objects to be modified are:

- Datatype sheets
  - CC_MB_TCP_M_DT_Device_CHARX_Control_Rev_1
- Features and function blocks:
  - CC_MB_TCP_M_FB_CHARX_Control_ConversionMethod_FC3_R_1
- And the underlying methods
  - CC_MB_TCP_M_FB_CHARX_Control_MaxChargingCurrent_FC6
  - CC_MB_TCP_M_FB_CHARX_Control_Fallback_FC16
- The FC6 and FC 16 function blocks can be adjusted depending on whether you also need to write to the device.

4.4.1 DataTypeSheet

In the datatype sheet, the data types are defined to store and display the read-in values after conversion.

4.4.2 Functions and function blocks

ConversionMethod_FC3 In the conversionMethod, the read-in modbus registers are transformed and scaled to the required value. For this, a calculation must be done in the second tab ”Calculation” depending on the conversion type for each type. These types are device dependent and need to be defined for each device. For each calculation type, a method must also be made to pass the data to the calculation.
Please note that there shouldn’t be any initialized variables in the Functions that should be automatically generated.

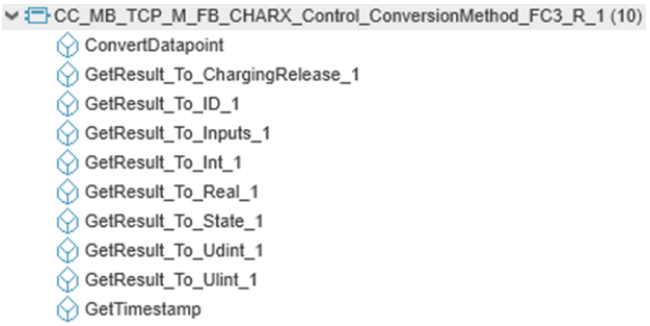


Figure 10: ConversionMethod_FC3

In the example below, only a transformation from a value to a real is defined. For this, the calculation type below is used and the data is transformed into a REAL data type via the GetResult_To_Real_1 method.

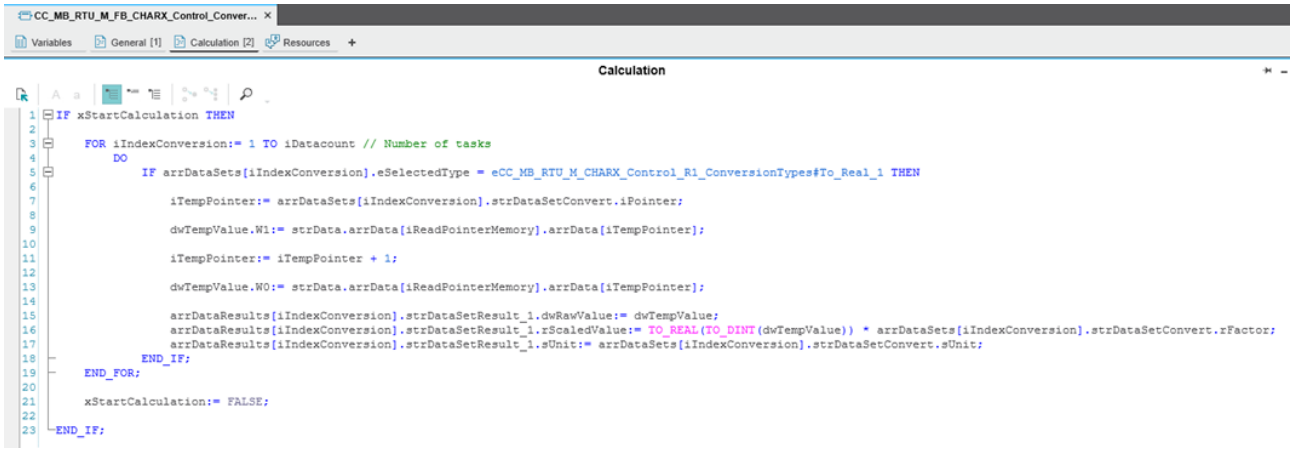


Figure 11: Calculation tab

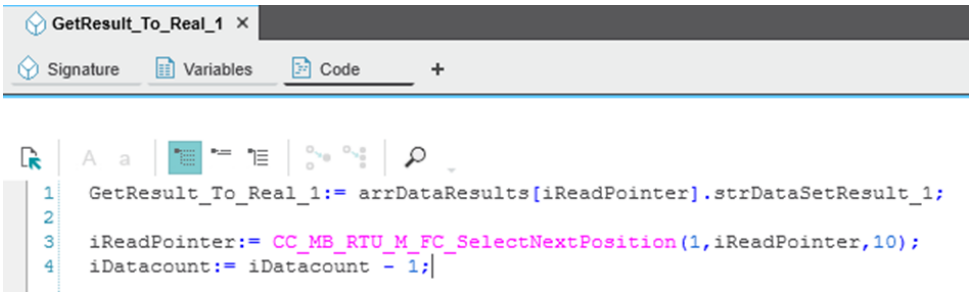


Figure 12: GetResult_To_Real_1

This is then returned via the data type below that is defined in the data type sheet.

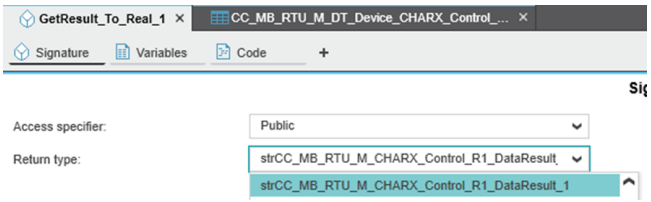


Figure 13: Define data type in method

```
strCC_MB_RTU_M_CHARX_Control_R1_DataResult_1:
    STRUCT
        dwRawValue:                 DWORD;
        rScaledValue:               REAL;
        sUnit:                      sCC_MB_RTU_M_5;
    END_STRUCT
```

Figure 14: Define data type in datatypesheet

SelectFunctionBlocks To write to a device, a function block must be written for each FC6 and FC16 call. The structure to forward data is not fixed and can be built up at will. The example below writes a maximum current value every minute or if it changes.

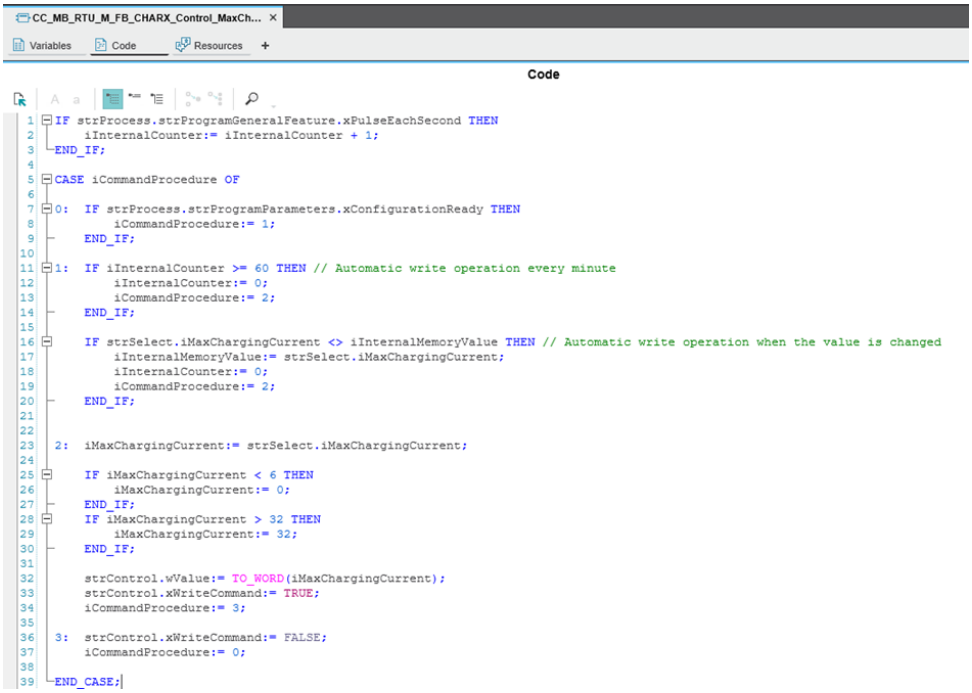


Figure 15: SelectFunctionBlock

4.5 STEP 5: Update the Device package

Update the source file, and edit the XML file to reference the datatypes and functionblocks you just updated/created. The source files that have to be put in the above folders should be extracted from a PLCnext project. You can open a PLCnext Engineer project as an archive with the following result:

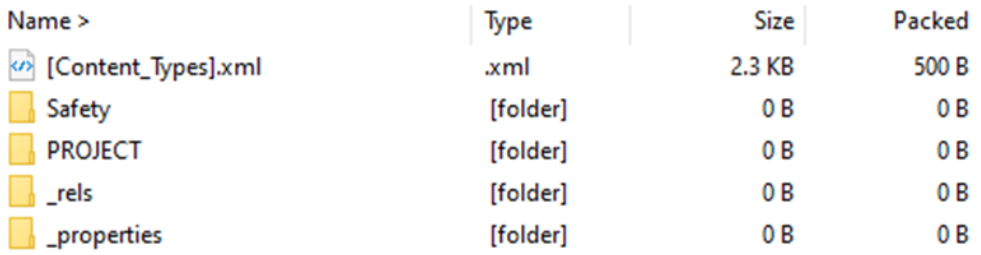


Figure 16: PLCnext Engineer project opened as an archive

Under PROJECT > Logical%20Elements you can then select the different data types and function blocks to place in the source files. The files in the folder of the required function block must be zipped in order to be used in the source package.

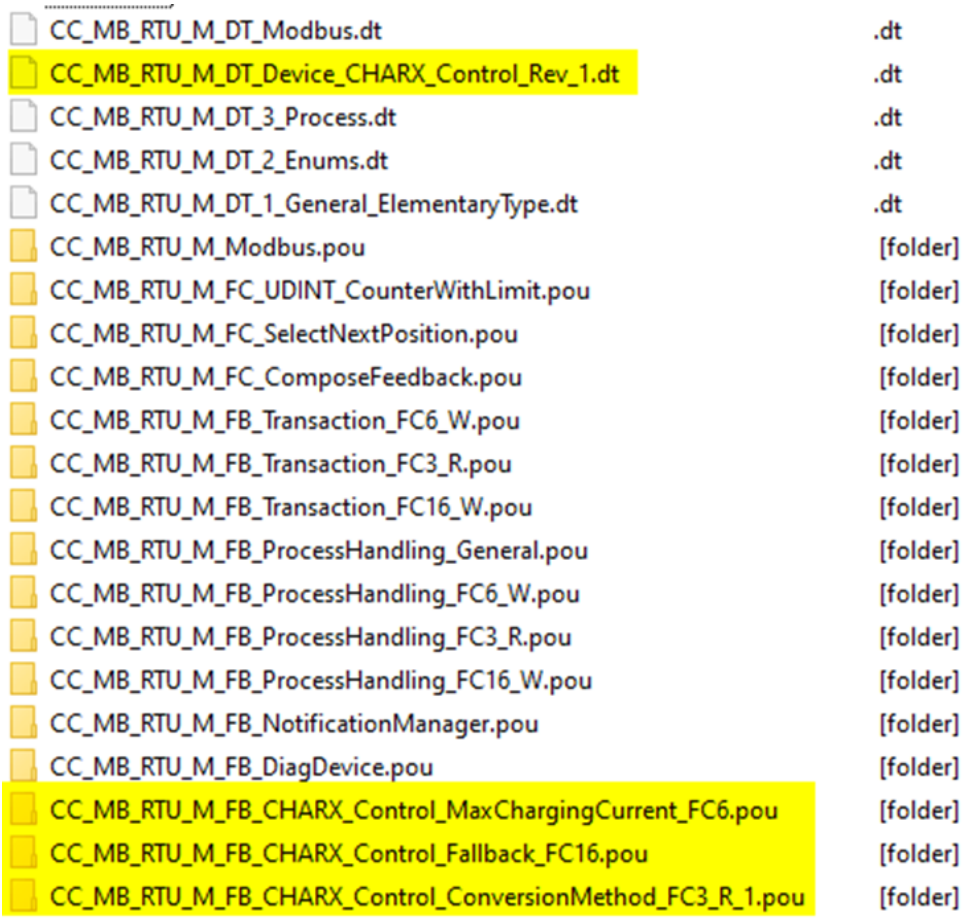


Figure 17: CHARX project functionblocks

The above example of a Charx controller shows the files to be copied into the source package. This consists of a data type sheet, 1 conversion function block and 2 selectfunctionblocks to send 2 registers via FC6 to the charging station. For FC 6, 1 selectfunctionblock is required per register to be forwarded. For FC16, multiple registers can be combined in 1 selectfunctionblock.

4.6 STEP 6: Upload the Device package and generate the PLC project once more

Test the new device package with the updated dataypes and functionblocks. Verify that there are no errors in the generated program, and test the communication.
Iterate through steps 3-6 until you are satisfied with the result of your device package.