2023

Code Creator

VERSION 2.0

EN

Contents

1 Revision Table

Date	Version	Description of Changes	Status	Author
12-09-2024	1.0	Initial version		Femke Parthoens

2 Introduction

With the code creator version 2.0 and up the user will be able to not only automatically create modbus RTU or TCP programs, but also add the readed values automatically to a PLC program that would interprete these values. This will save a lot of manual work and will make creating programs that use these modbus programs lesss errorprone.

The user will be able to import a zip file containing the information needed to link the devices' outputs in the program. To create this zip file the user can refer to Chapter 2

Please note that a basic knowledge of Code Creator v1.2 or v1.3 and PLC Next Engineer is required for using the new features.

Please note that all code in this document should be considered pseudo code and is not guarenteed to work when directly copied from this document, but should serve as a guidline.

3 Code Creator Syntax

3.1 Creating Sheets

The Create sheet command allows the creation of multiple sheets from *.ccsheet* files. Each *.ccsheet* file can define a sheets name and index within the filename.

Example:

Configuration [1].ccsheet

Controlsystem [3].ccsheet

Measurement [2].ccsheet

These *.ccsheet* files will create three sheets in the program in the following order:

1. Configuration
2. Measurement
3. Controlsystem

3.1.1 Overriding Name and Index

You can override the sheet name and index within the .ccsheet file itself using the following syntax:

SHEET Configuration INDEX 1

Note: Any duplication of sheet names or indexes will result in an error.

3.1.2 Importing Existing Variables

To include an existing variable sheet, add a *Variables.var* file to your project template's zip file and rename the extension to .ccvar. This allows you to include pre-defined variables from other programs.

Example of a *.ccvar* File:

```
{ CustomGroupDefinition('d652d334-3638-44fb-ab3d-631d2858b682', 'Inputs') }

{ CustomGroupDefinition('f97e56a2-91ab-49a2-8577-c96f0730ab97', 'In - Out') }

{ CustomGroupDefinition('fb84caa8-36a3-47d9-8bf0-95ff0e8a7b24', 'Local Variables') }

VAR_INPUT

        iID : INT

                { CustomGroupReference('d652d334-3638-44fb-ab3d-631d2858b682') }

                {Id('b521a8b0-5121-4324-9243-3f148d9f44e3')};

        strSelect : strCC_MB_RTU_M_EEM_MA_250_R1_WriteSelect_3

                { CustomGroupReference('f97e56a2-91ab-49a2-8577-c96f0730ab97') }

                {Id('dc3419cc-f28c-4ca0-8a2f-89a295c33cd8')};
```

```
END_VAR
VAR_IN_OUT
    strProcess : strCC_MB_RTU_M_ProcessCommunication
        { CustomGroupReference('f97e56a2-91ab-49a2-8577-c96f0730ab97') }
        {Id('324b30ad-f05e-4e22-a74c-7ce1838833e3')};
    strControl : strCC_MB_RTU_M_WriteSelect_FC6
        { CustomGroupReference('f97e56a2-91ab-49a2-8577-c96f0730ab97') }
        {Id('bd1b8f6b-62a6-46e0-9c0a-891a1e5aa32c')};
END_VAR
VAR
    iCommandProcedure : INT
        { CustomGroupReference('fb84caa8-36a3-47d9-8bf0-95ff0e8a7b24') }
        {Id('f659f8f9-7ff7-460b-a7ee-5ce69dba0d00')};
    dwSelectRotate : DWORD
        { CustomGroupReference('fb84caa8-36a3-47d9-8bf0-95ff0e8a7b24') }
        {Id('340975c8-3ad6-4bb6-93e0-108198818504')};
END_VAR
```

3.2 Creating Variable Groups

Variable groups are automatically generated when you create variables. However, if needed, you can create empty variable groups manually.

Command:

```
CREATE VARIABLE GROUP My variable group
```

Variable groups are ordered alphabetically by name.

3.3 Creating Variables

The *CREATE* command is used to define new variables and associate them with variable groups.

Example:

```
CREATE Local VARIABLE iConfiguration OF TYPE INT IN VARIABLE GROUP 'default'
```

This command creates a Local variable iConfiguration of type INT in the variable group default. If the group does not exist, it will be automatically created.

3.3.1 Variable Types

- Local
- Input
- Output
- InOut
- External

3.4 Adding Solution Dependencies (Libraries)

To include a library, use the *INCLUDE SOLUTION DEPENDENCY* command, providing the library name and path information.

Example:

INCLUDE SOLUTION DEPENDENCY Modbus_RTU_10

HintPath=$(LibrariesPath)\Modbus_RTU_10.pcwlx;Signature=yEseYXYzIZDmiH0mg6TCVXs/w0Q=;

To find this information:

1. Save your project as .pcweax.
2. Unzip the file.
3. Open the PROJECT/PROJECT.Proj file.
4. Locate the SolutionReference items:

<SolutionReference Include="Modbus_RTU_10">

<Id>6baa8565-d7e7-4cd6-bcab-e42da7a3f554</Id>

<HintPath>$(LibrariesPath)\Modbus_RTU_10.pcwlx</HintPath>

<Signature>yEseYXYzIZDmiH0mg6TCVXs/w0Q=</Signature>

<LibraryId>5fb48421-e798-46e4-825f-902b9d1e7c78</LibraryId>

<ParentIndex>2</ParentIndex>

</SolutionReference>

3.4.1 Adding Version Conditions

You can specify conditions based on the PLCnext Engineer version to control library inclusion.

Example:

INCLUDE SOLUTION DEPENDENCY WHEN PLCNEXT_ENGINEER_VERSION<=2023.9 Modbus_RTU_10

HintPath=$(LibrariesPath)\Modbus_RTU_10.pcwlx;Signature=yEseYXYzIZDmiH0mg6TCVXs/w0Q=;

The library will only be added if the PLCnext Engineer version is less than or equal to 2023.9. The following operators can be used for conditions:

- <
- <=
- =
- !=
- >
- >=

4 Practical example

In this example a program will be generated that uses the MINT library to generate an Energy Management Program that will control 2 Chargingpoints and one Collector. The Chargingpoints will be coupled to Module one and two of a Master and single slave combination of CHARX modules. The collector will be an EMPRO measurement device.

4.1 Needed Files

4.1.1 Configuration sheet

To use the MINT program we need to make a configuration sheet, the first file will be named *Configuration [1].ccsheet* with contents:

```
IF iConfiguration < 500 THEN iConfiguration:= iConfiguration + 1; END_IF;

CASE iConfiguration OF

1: MINT_Core_ControlSystem.Configuration_Collector(

        'Grid',

        'VPK',

        eMINT_Configuration_PhasePermutation#RST_L1L2L3,

        eMINT_Configuration_CollectorFallbackInvalidMonitoring#Disable,

        eMINT_Configuration_CollectorDynamicReserve#Off,

        0,

        32);

100: MINT_Core_ControlSystem.Configuration_AC_Chargepoint(

        'VPK',

        'VPK_AC1',

        eMINT_Configuration_PhasePermutation#RST_L1L2L3,

        eMINT_Configuration_CollectorDynamicReserve#Off,

        eMINT_Configuration_AC_ChargepointMaxCurrent#Max_32A);
```

```
101: MINT_Core_ControlSystem.Configuration_AC_Chargepoint(
        'VPK',
        'VPK_AC2',
        eMINT_Configuration_PhasePermutation#RST_L1L2L3,
        eMINT_Configuration_CollectorDynamicReserve#Off,
        eMINT_Configuration_AC_ChargepointMaxCurrent#Max_32A);
200:    MINT_Core_ControlSystem.Configuration_AC_Chargepoint_WorkingMode(
        eMINT_Configuration_AC_ChargepointWorkingMode#AdviceBased);
201:    MINT_Core_ControlSystem.Configuration_AC_Chargepoint_EnergyReport(
        eMINT_Configuration_AC_ChargepointEnergyReport#EnableArray);
202:    MINT_Core_ControlSystem.Configuration_Collector_EnergyReport(
        eMINT_Configuration_CollectorEnergyReport#EnableArray);
203:    MINT_Core_ControlSystem.Configuration_PV_EnergyReport(
        eMINT_Configuration_PVEnergyReport#EnableArray);
204:    MINT_Core_ControlSystem.Configuration_DefaultUserProfile(
        eMINT_Configuration_DefaultUserProfile#MaxPower);
300:    MINT_Core_ControlSystem.Configuration_Ready();
400:    MINT_Core_ControlSystem.Control_EnableDisable(eMINT_Control_ChargingPark#Enable);
END_CASE;
```

4.1.2 Measurements sheet

one file will be added for the collector measurements and one for the AC Chargepoints measurements *Collector [2].ccsheet* with contents:

```
//***Code Creator***
//ADD DEVICE VPK
//*** Code Creator ***
// *** Collector *** VPK ********************************************************************
Collector_1.Input_Measurements(
        REQUIRED VPK - Power TYPE REAL UNIT W (TotalActivePower),
        REQUIRED VPK - Power L1 TYPE REAL UNIT W (ActivePower1),
        REQUIRED VPK - Power L2 TYPE REAL UNIT W (ActivePower2),
        REQUIRED VPK - Power L3 TYPE REAL UNIT W (ActivePower3),
        REQUIRED VPK - I L1 TYPE REAL UNIT A (Current1),
        REQUIRED VPK - I L2 TYPE REAL UNIT A (Current2),
```

REQUIRED VPK - I L3 TYPE REAL UNIT A (Current3));

Collector_01(sName:= 'VPK',

xCommunicationValid:= IN_VPK.strDiag.iSecondsSinceLastSuccessfulTransaction < 30,
arrDataInterface:= arrCollectorDataInterface); AC [3].ccsheet

with contents:

*//***Code Creator***

//ADD DEVICE VPK_AC1

*//*** Code Creator ***

*// *** AC Chargepoint *** VPK_AC1 **

AC_Chargepoint_1.Input_Measurements(

REQUIRED VPK_AC1 - Power TYPE REAL UNIT W (TotalActivePower),

REQUIRED VPK_AC1 - I L1 TYPE REAL UNIT A (Current1),

REQUIRED VPK_AC1 - I L2 TYPE REAL UNIT A (Current2),

REQUIRED VPK_AC1 - I L3 TYPE REAL UNIT A (Current3),

REQUIRED VPK_AC1 - Vehicle status TYPE STRING (strVehicleStatus));

AC_Chargepoint_1.Input_Voltage(

OPTIONAL VPK_AC1 - U L1_N TYPE REAL UNIT V (Voltage_1N),

OPTIONAL VPK_AC1 - U L2_N TYPE REAL UNIT V (Voltage_2N),

OPTIONAL VPK_AC1 - U L3_N TYPE REAL UNIT V (Voltage_3N));

AC_Chargepoint_1(

sName:= 'VPK_AC1',

xCommunicationValid:= IN_VPK_AC1.strDiag.iSecondsSinceLastSuccessfulTransaction < 30,

iMaxChargingCurrent => TO_BE_FILLED_OUT,

arrDataInterface:= arrACChargepointDataInterface);

*//***Code Creator***

//ADD DEVICE VPK_AC2

*//*** Code Creator ***

*// *** AC Chargepoint *** VPK_AC2 **

AC_Chargepoint_2.Input_Measurements(

REQUIRED VPK_AC2 - Power TYPE REAL UNIT W (TotalActivePower),

REQUIRED VPK_AC2 - I L1 TYPE REAL UNIT A (Current1),

REQUIRED VPK_AC2 - I L2 TYPE REAL UNIT A (Current2),

REQUIRED VPK_AC2 - I L3 TYPE REAL UNIT A (Current3),

REQUIRED VPK_AC2 - Vehicle status TYPE STRING (strVehicleStatus));

AC_Chargepoint_2.Input_Voltage(

OPTIONAL VPK_AC2 - U L1_N TYPE REAL UNIT V (Voltage_1N),

OPTIONAL VPK_AC2 - U L2_N TYPE REAL UNIT V (Voltage_2N),

OPTIONAL VPK_AC2 - U L3_N TYPE REAL UNIT V (Voltage_3N));

AC_Chargepoint_2(

sName:= 'VPK_AC2',

xCommunicationValid:= IN_VPK_AC2.strDiag.iSecondsSinceLastSuccessfulTransaction < 30,

iMaxChargingCurrent => TO_BE_FILLED_OUT,

arrDataInterface:= arrACChargepointDataInterface);

4.1.3 Variable sheet

To add the variables to the program the following sheet will have to be made variables.ccvar with contents:

{ CustomGroupDefinition('5f7b5719-2d70-44e2-8bf9-74328f9c9cd2', 'Inputs') }

{ CustomGroupDefinition('1932cd6c-80f7-47b4-96ce-a20f0122d4fa', 'In - Out') }

{ CustomGroupDefinition('82ffa061-68fd-44e2-991d-70206423e1a0', 'Local Variables') }

{ CustomGroupDefinition('794109b9-8b8f-4e47-8247-664dec6969dc', 'Control') }

{ CustomGroupDefinition('955b2942-b0f8-4183-a7f2-b983dedb4f6c', 'Assets') }

{ CustomGroupDefinition('9cc54a1c-6e8c-41b9-a0cc-4b65a8ee7707', 'MQTT') }

VAR_GLOBAL

strMINT_IncomingCommunicationDatahub : strMINT_Input_DatahubCloudCommunication

{ InputPort, CustomGroupReference('9cc54a1c-6e8c-41b9-a0cc-4b65a8ee7707') }

{Id('b22fda05-91db-40e3-9e23-7673e767fcfe')};

strMINT_OutgoingCommunicationDatahub : strMINT_Output_DatahubCloudCommunication

{ OutputPort, CustomGroupReference('9cc54a1c-6e8c-41b9-a0cc-4b65a8ee7707') }

{Id('7ee903a0-e878-4dc1-a000-a4c5cbb0d478')};

strMQTT_Datahub_Settings : strMQTT_Input_DatahubCloudCommunication_Settings

{ OutputPort, CustomGroupReference('9cc54a1c-6e8c-41b9-a0cc-4b65a8ee7707') }

{Id('398ba85d-cf3c-4b08-94c4-e97fe5561115')};

END_VAR

VAR

MINT_Core_ControlSystem : MINT_FB_ControlSystem

{ CustomGroupReference('794109b9-8b8f-4e47-8247-664dec6969dc') }

{Id('e938feaa-4e14-40c0-a86e-0dd0211f9506')};

AC_Chargepoint_1 : MINT_FB_AC_Chargepoint_DataInterface

{ CustomGroupReference('955b2942-b0f8-4183-a7f2-b983dedb4f6c') }

{Id('73817490-525d-4ca4-bc93-47f6a0611d3b')};

AC_Chargepoint_2 : MINT_FB_AC_Chargepoint_DataInterface

{ CustomGroupReference('955b2942-b0f8-4183-a7f2-b983dedb4f6c') }

{Id('ac65b1e1-61f8-4c55-9a11-7f7434319259')};

Collector_1 : MINT_FB_Collector_DataInterface

{ CustomGroupReference('955b2942-b0f8-4183-a7f2-b983dedb4f6c') }

{Id('3ec25d05-f855-4ae9-a1bc-ea164d6cf1a9')}; xControlSystemActive : BOOL

{ CustomGroupReference('794109b9-8b8f-4e47-8247-664dec6969dc') }

{Id('17f172f5-7d56-4163-8a22-312418190fe6')}; xRunningInDemoMode : BOOL

{ CustomGroupReference('794109b9-8b8f-4e47-8247-664dec6969dc') }

{Id('a6b2e20c-0989-4730-a741-0f8376d72b8a')}; arrControlSystemDiagnostic : arrMINT_Diag_1_25_String

{ CustomGroupReference('794109b9-8b8f-4e47-8247-664dec6969dc') }

{Id('6bb8e311-a093-4a6c-a978-3a6464fdc370')}; arrCollectorDataInterface : arrMINT_Data_1_500_Byte

{ CustomGroupReference('794109b9-8b8f-4e47-8247-664dec6969dc') }

{Id('8359cfd9-6313-4c3a-84ad-f1096fbb3f95')}; arrACChargepointDataInterface : arrMINT_Data_1_500_Byte

{ CustomGroupReference('794109b9-8b8f-4e47-8247-664dec6969dc') }

{Id('9c8ccd2e-648a-4444-9443-9d8d14a90c9d')};
arrDCChargepointDataInterface : arrMINT_Data_1_500_Byte

{ CustomGroupReference('794109b9-8b8f-4e47-8247-664dec6969dc') }

{Id('e5313328-3c67-4f2b-abb0-42564e52a40b')};
arrPVSystemDataInterface : arrMINT_Data_1_500_Byte

{ CustomGroupReference('794109b9-8b8f-4e47-8247-664dec6969dc') }

{Id('bdbdda1d-e9ff-4a9d-b851-e24f2f3f2ab4')}; iConfiguration : INT

{ CustomGroupReference('794109b9-8b8f-4e47-8247-664dec6969dc') }

{Id('8b8a63ad-6585-4003-976a-8a6ad894c00e')};

END_VAR

4.1.4 zip file

The resulting zip file will look like this:

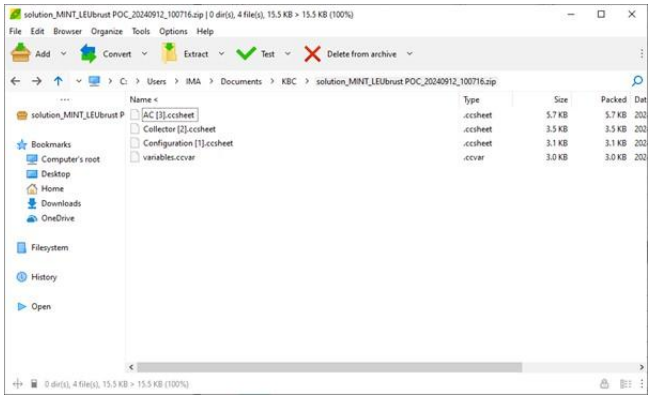

Figure 1: The resulting zip file.

4.2 Code Creator

Now we will drag the zipfile in the Code Creator environment and add it alongside all of our devices:

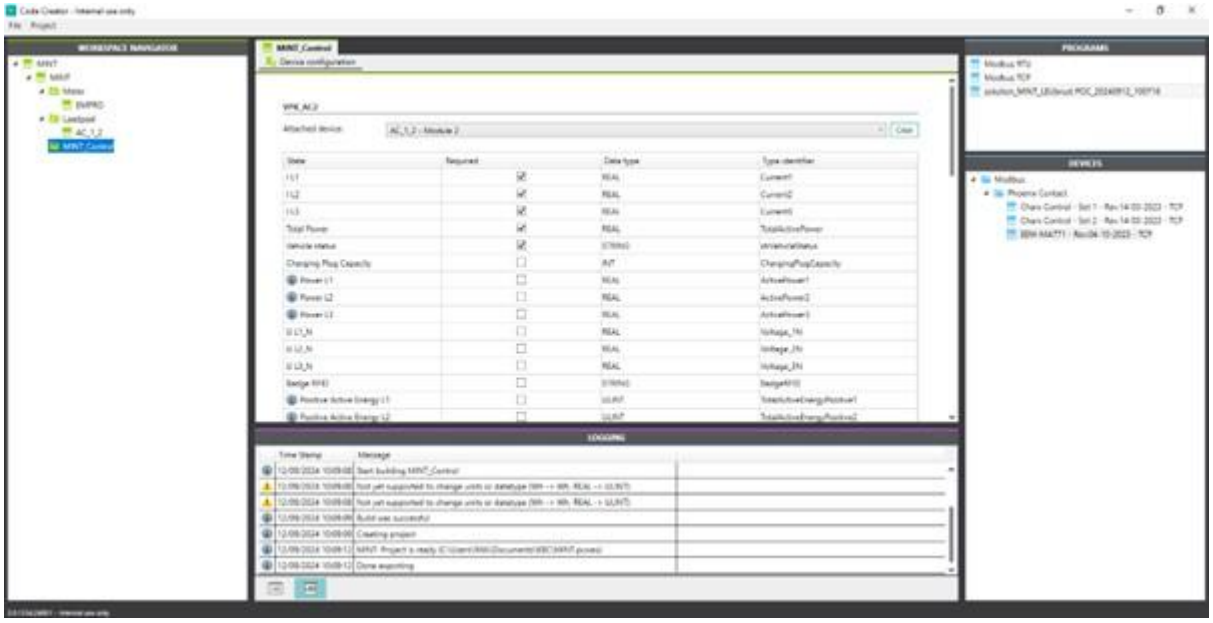

Figure 2: Code Creator.

And link the devices that we added before:

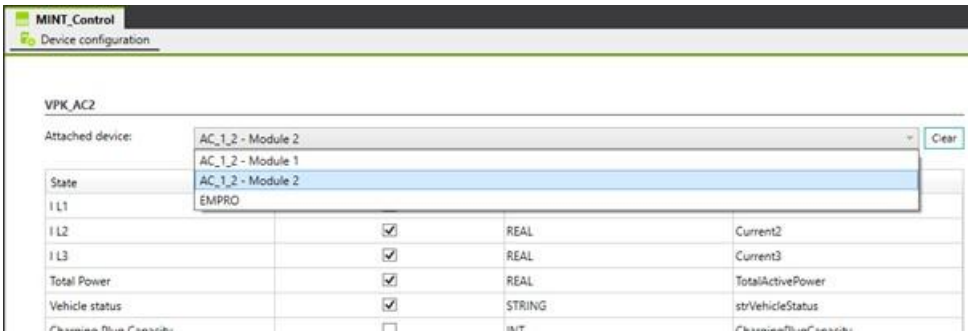

Figure 3: Linking the devices.

Now the program can be exported

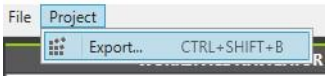

Figure 4: Export the program.

4.3 PLC Next Engineer

Open a blanco project in PLC Next Engineer, or your own template and import the project that was just created:

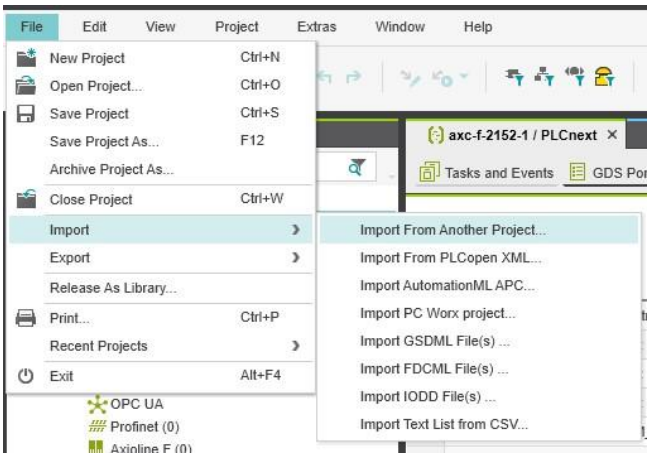

Figure 5: Import the project.

Check for any errors and resolve them if needed. Manually link the OUT PORTS of your program. The result should look similar to this code:

```
//***Code Creator***
//ADD DEVICE VPK_AC2
//*** Code Creator ***
// *** AC Chargepoint *** VPK_AC2 ********************************************************************************

AC_Chargepoint_2.Input_Measurements(                IN_VPK_AC2.strRead.strModule_2_Measurement.strData.strTotalActivePower.rScaledValue,
                                                    IN_VPK_AC2.strRead.strModule_2_Measurement.strData.strCurrentPhase_L1.rScaledValue,
                                                    IN_VPK_AC2.strRead.strModule_2_Measurement.strData.strCurrentPhase_L2.rScaledValue,
                                                    IN_VPK_AC2.strRead.strModule_2_Measurement.strData.strCurrentPhase_L3.rScaledValue,
                                                    IN_VPK_AC2.strRead.strModule_2_Measurement.strData.strVehicleStatus.sValue);

AC_Chargepoint_2(sName:= 'VPK_AC2',
xCommunicationValid:= IN_VPK_AC2.strDiag.iSecondsSinceLastSuccessfulTransaction < 30,
iMaxChargingCurrent => AC_1_2_Control.strWrite.strModule_2_MaxChargingCurrent.strControl.iMaxChargingCurrent,
arrDataInterface:= arrACChargepointDataInterface);
```

Figure 6: Snippet of resulting Code.

Now initialize all the programs you need for your project, including the modbus programs created by Code Creator.

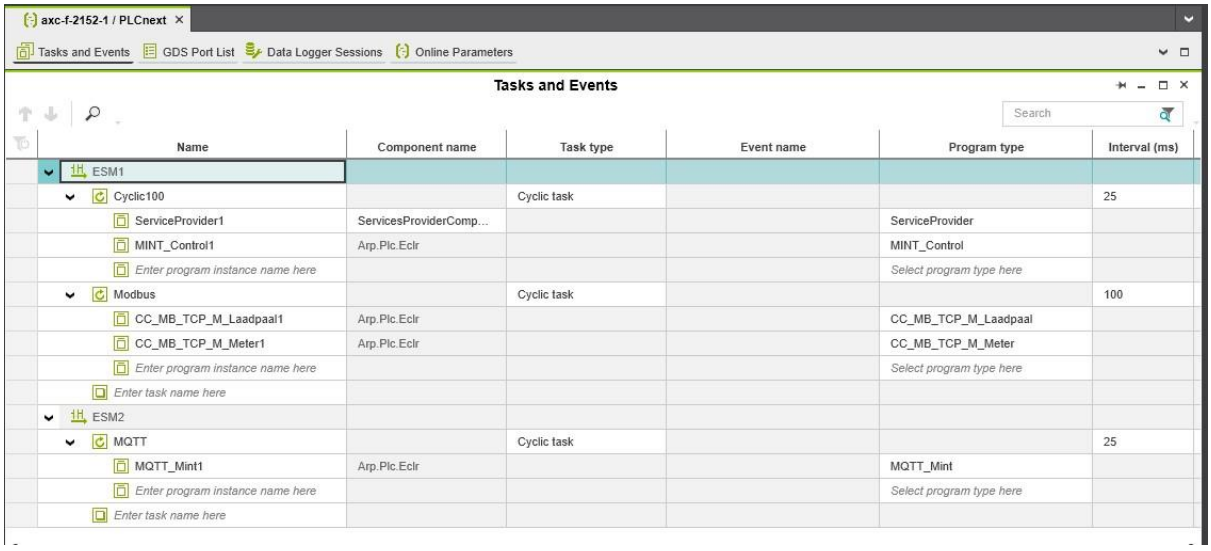

Figure 7: Initialize programs.

And link all your PORTS:

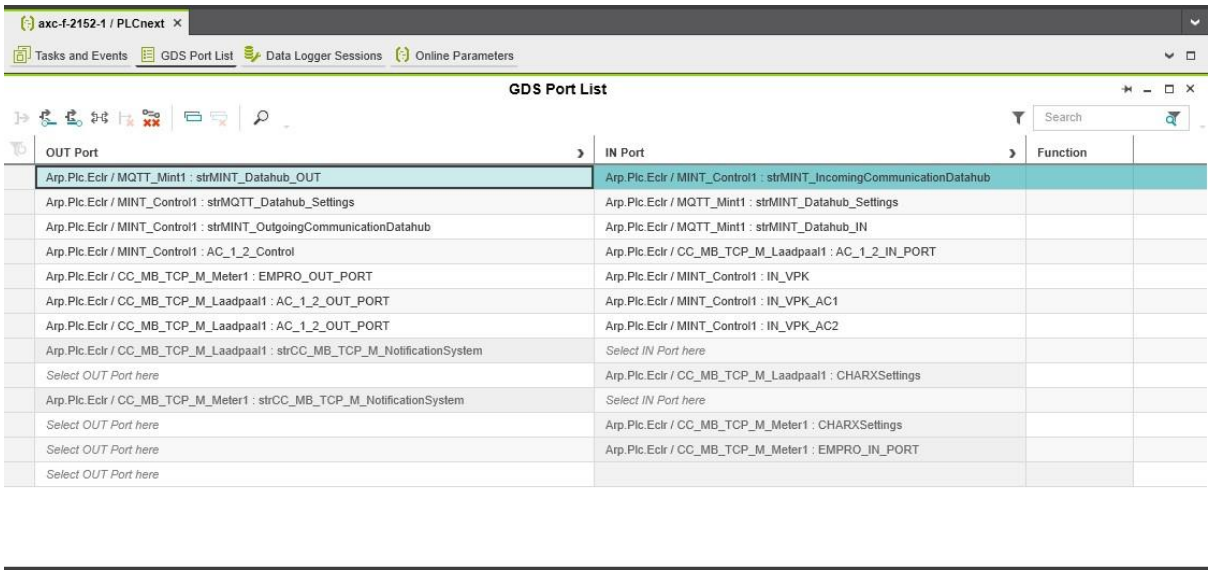

Figure 8: Link all PORTS.

The solution is ready to write to the PLC.