20 Programmable logic controller

A programmable logic controller (PLC) is a digital device adapted for control of processes which require high reliability, ease of programming and realtime responses. Such functionality has long since replaced hardwired relays, timers and sequencers which would be required to complete various tasks. Programmable logic controllers usually had to conform to IEC 611313 standard which defines four programming languages: function block diagram (FBD), ladder diagram (LD), structured text (ST) and sequential function chart (SFC). This standard does not support distributed control systems therefore IEC 61499 standard was published in 2005. The standard is considered an extension of IEC 611313 standard. WCC Lite supports PLC functionality while conforming to specifications of IEC 61499 standard.

- 20.1 IEC 61499
- 20.2 4Diac framework
- 20.3 Example project
- 20.4 Configuring data endpoints
- 20.5 Debugging an IEC 61499 application
- 20.6 Generating and uploading FORTE logic file
- 20.7 Distributed control application

20.1 IEC 61499

IEC 61499-1 defines the architecture for distributed systems. In IEC 61499 the cyclic execution model of IEC 61131 is replaced by an event driven execution model. The event driven execution model allows for an explicit specification of the execution order of function blocks. If necessary, periodically executed applications can be implemented by using the E_CYCLE function block for the generation of periodic events.

IEC 61499 enables an application-centric design, in which one or more applications, defined by networks of interconnected function blocks, are created for the whole system and subsequently distributed to the available devices. All devices within a system are described within a device model. The topology of the system is reflected by the system model. The distribution of an application is described within the mapping model. Therefore, applications of a system are distributable but maintained together.

Like IEC 61131-3 function blocks, IEC 61499 function block types specify both an interface and an implementation. In contrast to IEC 61131-3, an IEC 61499 interface contains event inputs and outputs in addition to data inputs and outputs. Events can be associated with data inputs and outputs by WITH constraints. IEC 61499 defines several function block types, all of which can contain a behavior description in terms of service sequences:

Service interface function block – SIFB: The source code is hidden and its functionality is only described by service sequences;
• Basic function block - BFB: Its functionality is described in terms of an Execution Control Chart (ECC), which is similar to a state diagram (UML). Every state can have several actions. Each action references one or zero algorithms and one or zero events. Algorithms can be implemented as defined in compliant standards.
• Composite function block - CFB: Its functionality is defined by a function block network.
• Adapter interfaces: An adapter interface is not a real function block. It combines several events and data connections within one connection and provides an interface concept to separate specification and implementation.
• Sub application: Its functionality is also defined as a function block network. In contrast to CFBs, sub applications can be distributed.

To maintain the applications on a device IEC 61499 provides a management model. The device manager maintains the lifecycle of any resource and manages the communication with the software tools (e.g., configuration tool, agent) via management commands.

20.2 4Diac framework

The PLC functionality in the WCC Lite is implemented using Eclipse 4diac framework, consisting of the 4diac IDE and the 4diac FORTE runtime. The system corresponds to IEC 61499, an extension of IEC 61131-3. For more in-depth instructions and function block reference please see the 4diac manual - this document is merely a quick start guide that emphasizes the specifics of tailoring the applications to run on the WCC Lite.

The 4diac IDE application is used to model logic sequences. An output file, *.fboot, is then generated and either loaded into the runtime for debugging purposes (functionality available from within the IDE), or uploaded into the controller for normal use via web interface.

> ⚠ During debugging, the output logic is executed directly in the runtime. Any logic loaded during debugging will be discarded after a reboot of the controller. Logic applications for regular use should be uploaded via the web interface.
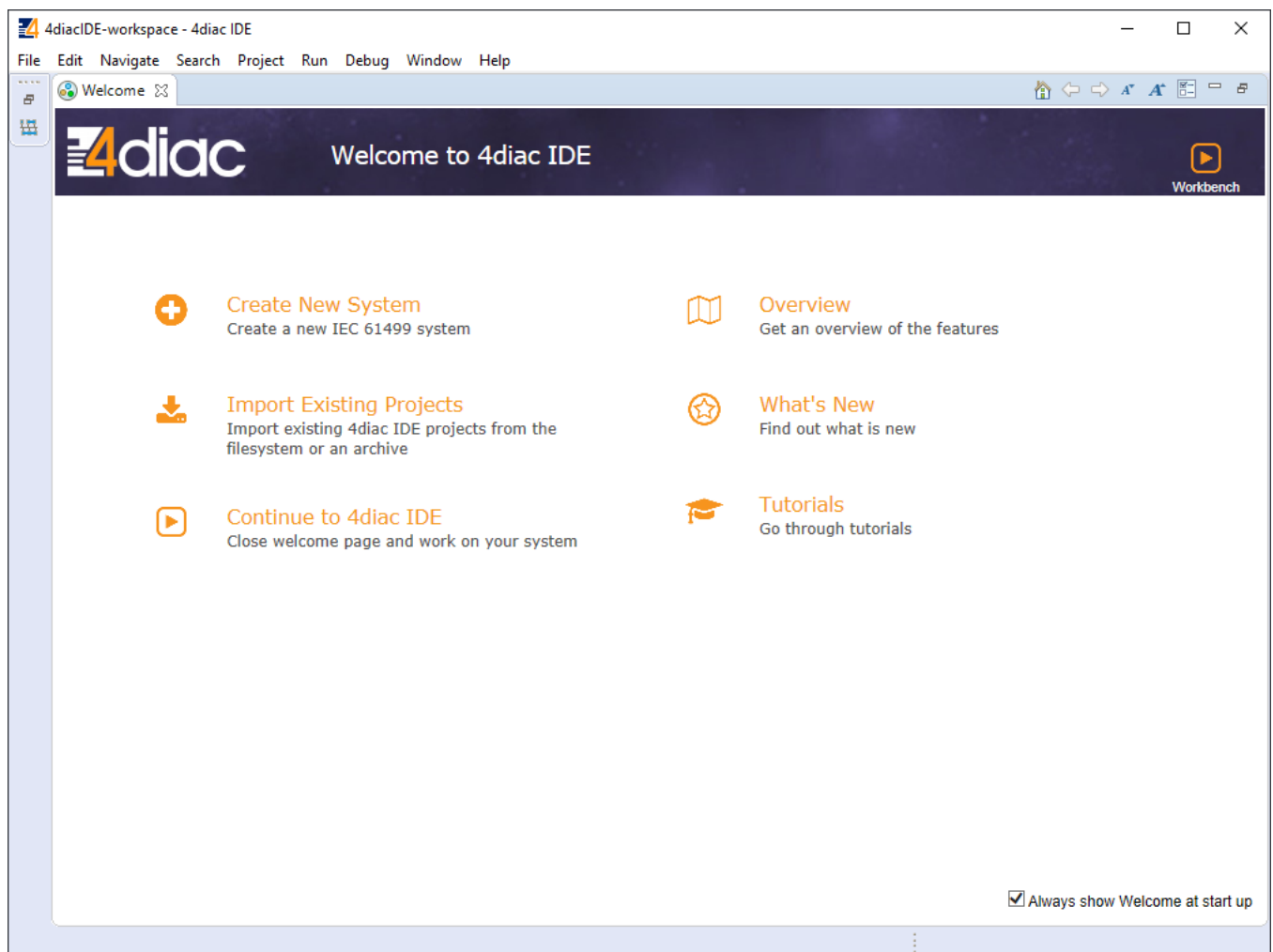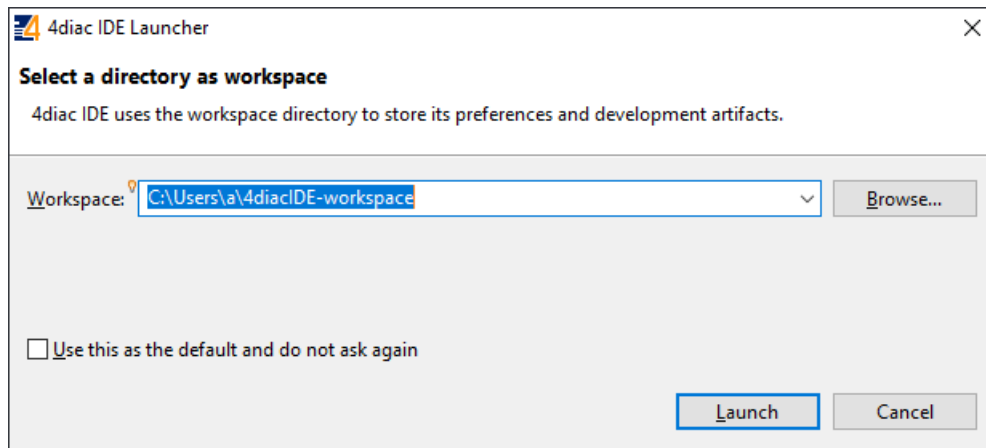
> ⓘ It is possible to run multiple tasks at once. These tasks can either be implemented in the same screen or split into separate tasks. Please note, however, that all elements should have unique names, even between different tasks. As of 4diac IDE 1.11.3 this is not enforced between separate apps, however, 4Diac runtime application rejects such file purely because of naming issues.

The 4diac FORTE runtime is able to execute the aforementioned fboot files containing the logic. The FORTE runtime can be run on both the WCC Lite and a PC for debugging purposes. The runtime is integrated to interact with the REDIS database.

20.3 Example project

The best way to understand basics of 4Diac and WCC Lite collaboration is through an example project. This user manual intends to show the pieces needed to run PLC applications on WCC Lite. It is not intended to be definitive guide on how to use 4Diac IDE or how to interpret IEC 61499 standard.

During (at least) the first start of the IDE user will be asked to select a directory for the workspace as in Figure. Workspace is used to save files needed for projects.
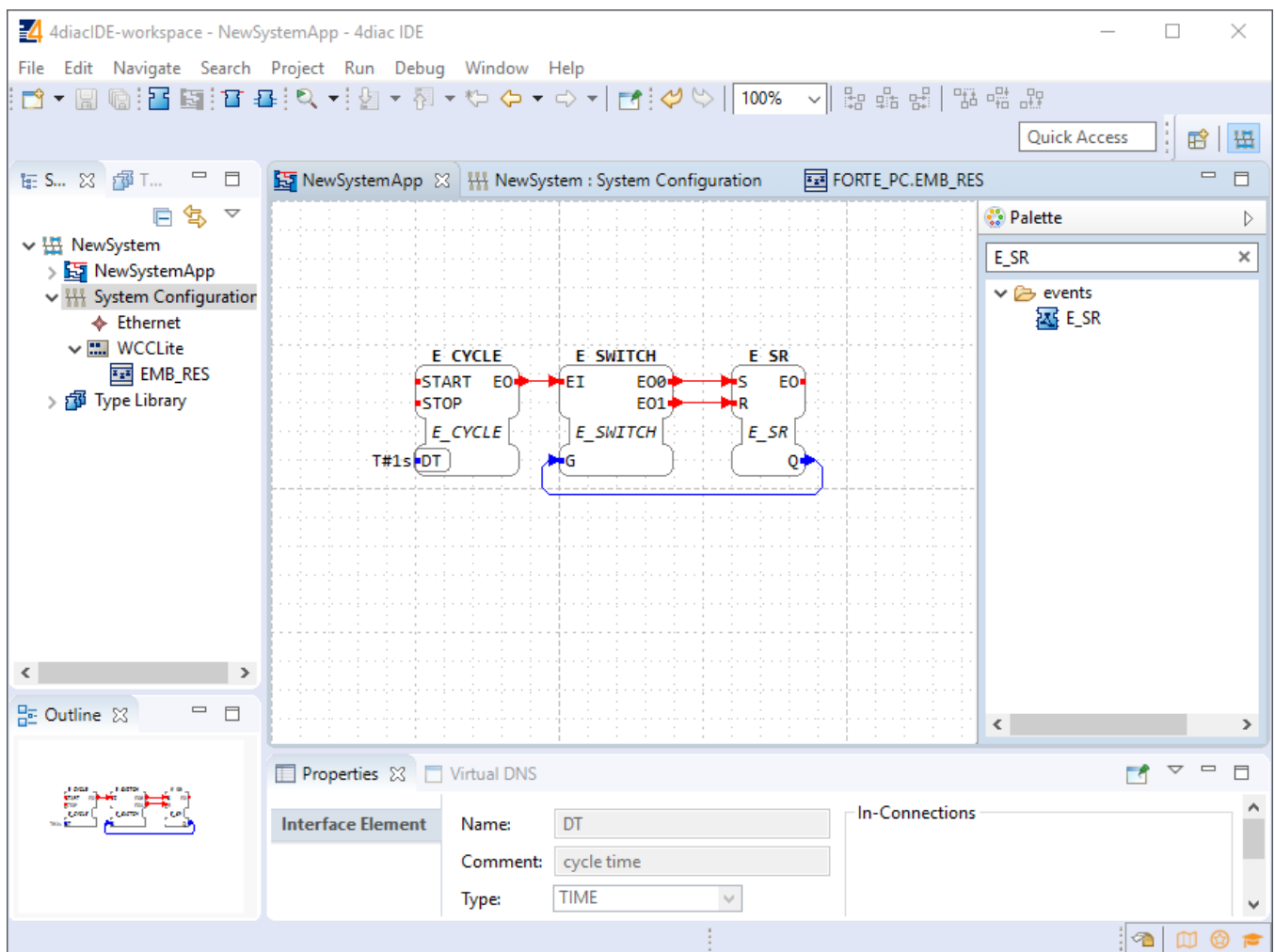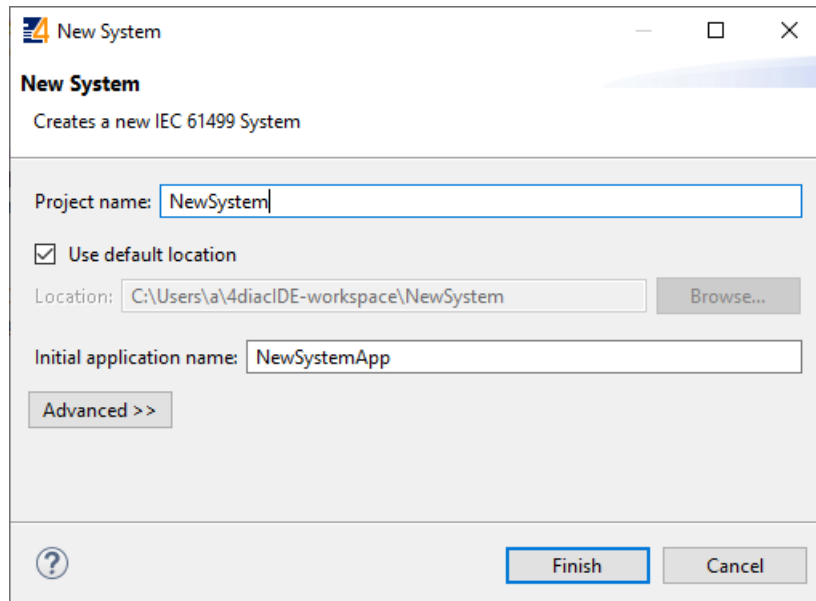

After that a user should be met by the welcome window. If such window is not shown, one can create project by selecting File -->New --> Project and filling in the required fields.

Here is a library of function blocks, which are working with the newest 4Diac version: Elseta Library.zip. Upload the library to your project folder.

To create a simple application, simply drag and drop objects from the palette to the canvas and wire them accordingly. Event trigger and data pathways cannot be connected to one another. Displayed below is an example of a simple

blinker application.

> ℹ️ Having less wiring by connecting several signals to same subnet as PCB designer (such as Altium Designer) as of 4Diac IDE 1.11.3 is not supported. However, if some parts are used frequently, it is highly advised to have less wiring by simply compiling several elements into a sub application. For this, you would have to select elements to be grouped, press right key and select New Sub application. You can later change names of such elements and its pins.
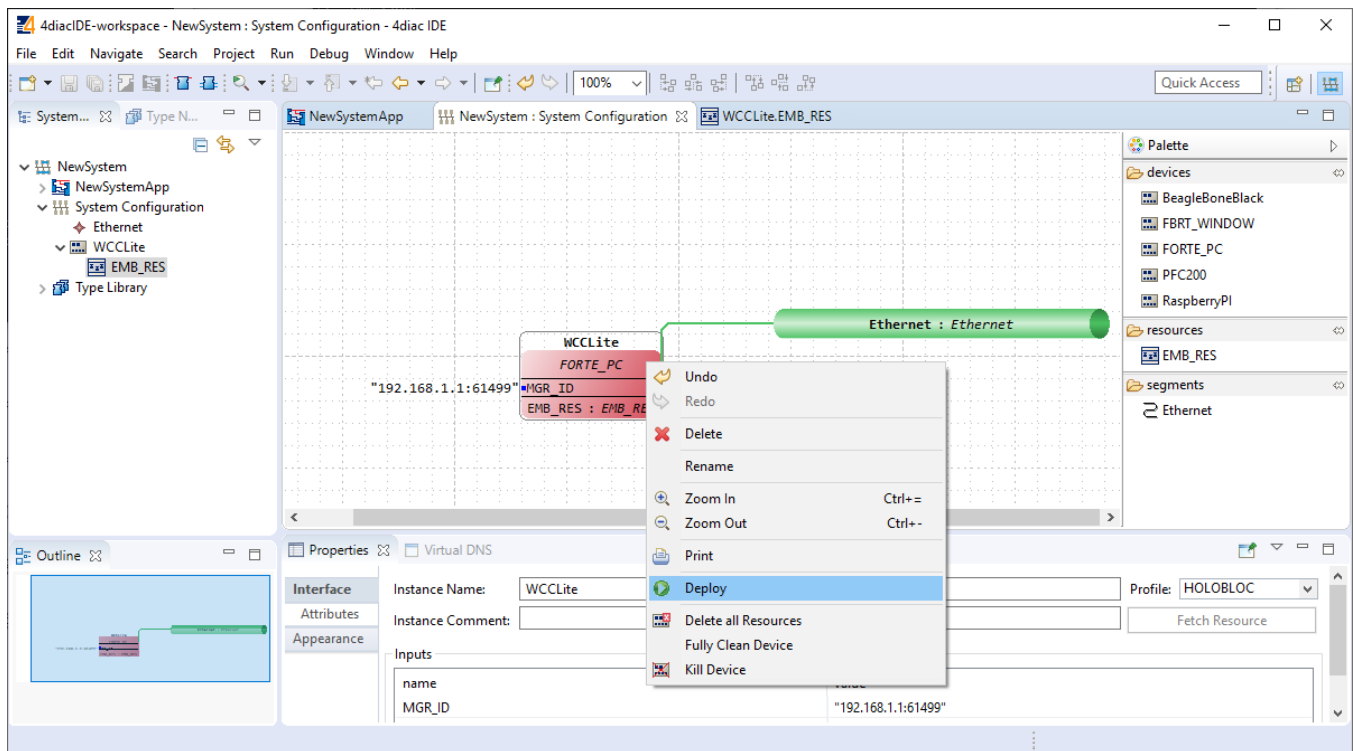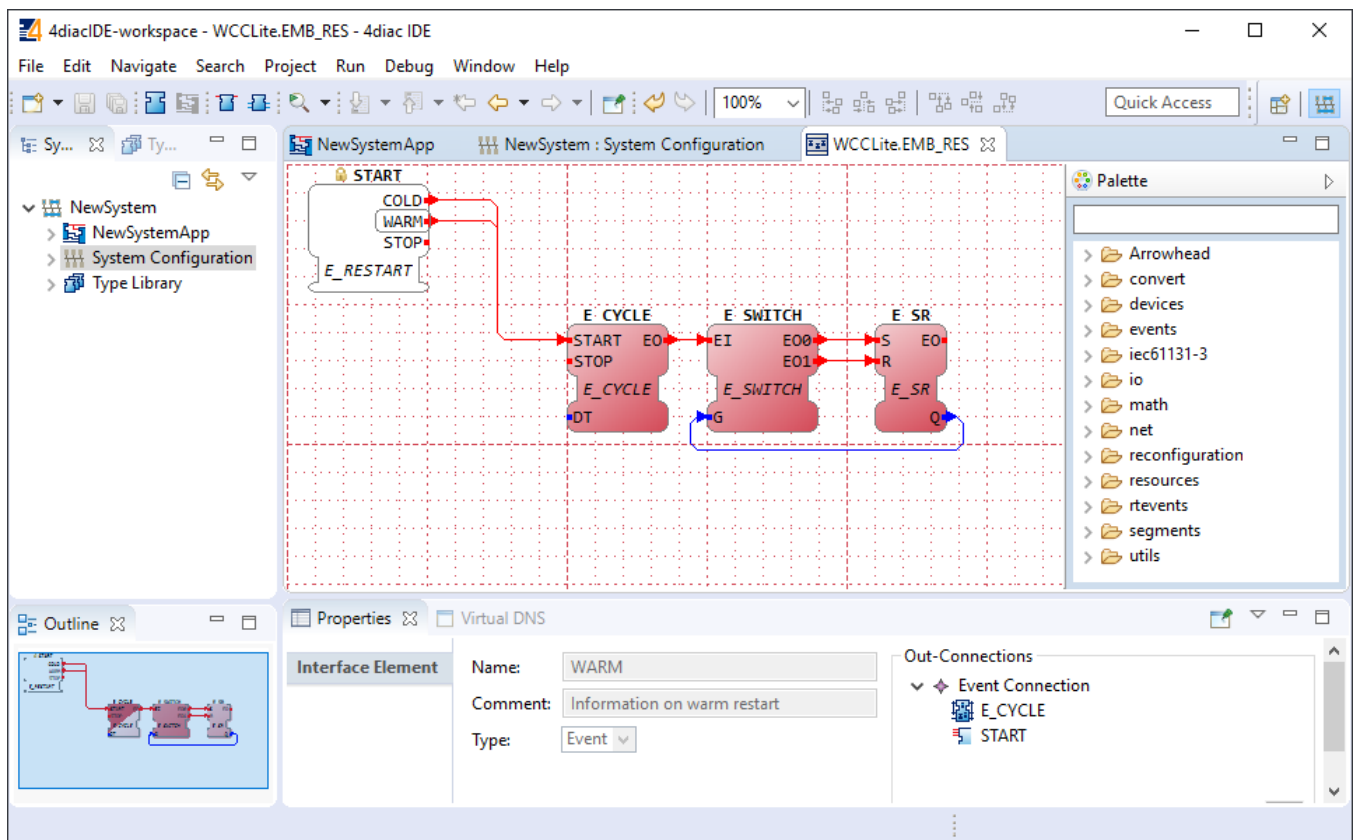

In the System Configuration section, drag and drop a FORTE_PC device, an Ethernet segment and link them. For debugging in the local (PC) runtime, leave the address "localhost:61499". For testing on a WCC Lite, enter the IP address of the device, along with the port number (which by default is 61499 as well).

In order to deploy the application, the circuit needs to be mapped to the controller. For a non-distributed application (distributed application cases will not be discussed in this chapter), all the FBs of the application need to be selected and mapped to the configured controller as shown:

To start the application execution, an initial trigger needs to be present. For a non-distributed application, the initial event trigger needs to be wired from the START function block in the resource section as shown:

To deploy the application, go to the System Configuration tab and simply select "Deploy" from the right-click menu of the controller device. If a running application exist in the runtime, you may be asked whether you want to replace it. This will only overwrite the application in the memory and not the storage. If the controller is restarted, the old application will be loaded from the non-volatile memory of the controller.

20.4 Configuring data endpoints

To use WCC Lite as a programmable logic controller, it needs to be configured in a particular way. The PLC functionality of the WCC Lite only allows for the use of data that is has been configured in the Excel configuration spreadsheet. This has been done for security purposes and to preserve transmission medium only for data that is available. Only topics defined in the configuration can post or get data. If a certain data entry exists but it has not been linked to a PLC program, all calls from PLC runtime application to redis database will be ignored. Therefore it is highly advised to prepare and upload the Excel configuration before using this signal in the PLC application.

Some parameters are mandatory for PLC usage. These parameters are shown in two tables below (one for Devices, one for Signals tab). Please note that other parameters can be used as well, but are not covered because they aren't specific to PLC functionality.

Table. Mandatory parameters for Devices tab

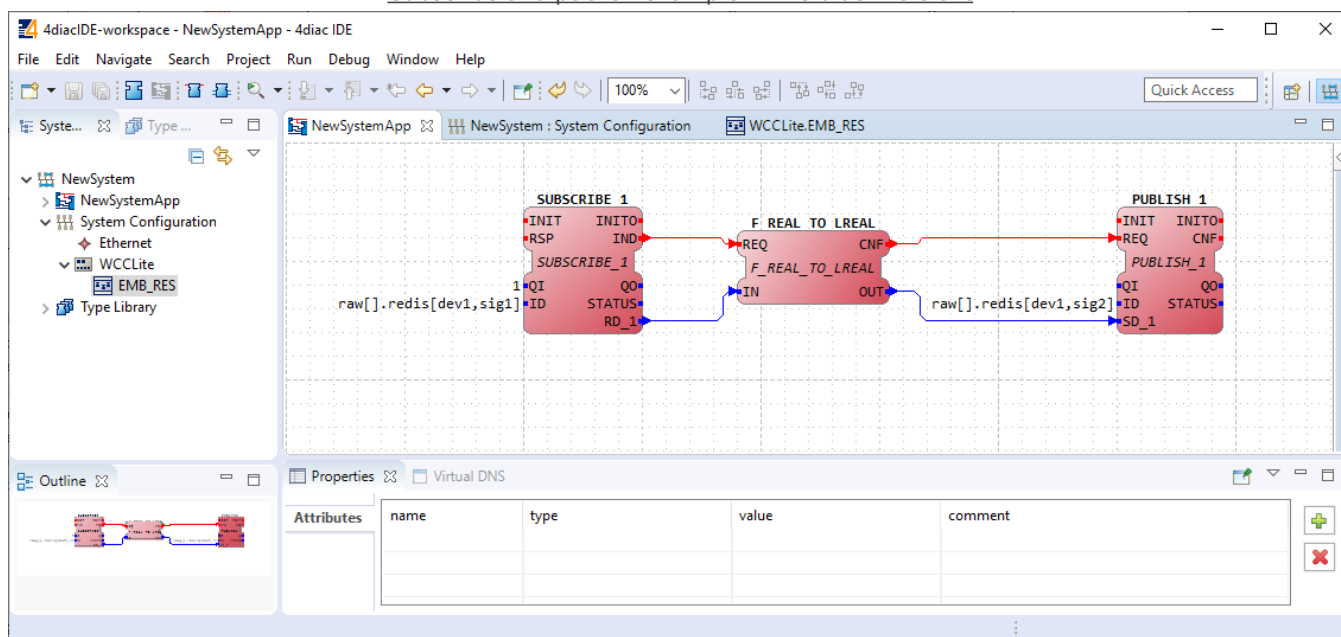| Parameter	Type	Description
name	string	User-friendly device name
device_alias	string	Device alias to used in configuration
enable	boolean	Enabling/disabling of a device
protocol	string	Selection of protocol (IEC 61499)

Table. Mandatory parameters for Signals tab

Parameter	Type	Description
signal_name	string	User-friendly signal name
device_alias	string	Device alias from a Devices tab
signal_alias	string	Unique signal name to be used
source_device_alias	string	device_alias of a source device
source_signal_alias	string	signal_alias of a source signal

If an upload consisting of configuration for IEC 61499 has been successful, one should be able to access a configuration stored in /etc/iec61499.json file where protocol-specific parameters are shown in a JSON format. If the file is missing, make sure you have a correct firmware version installed and haven't made any typing errors.
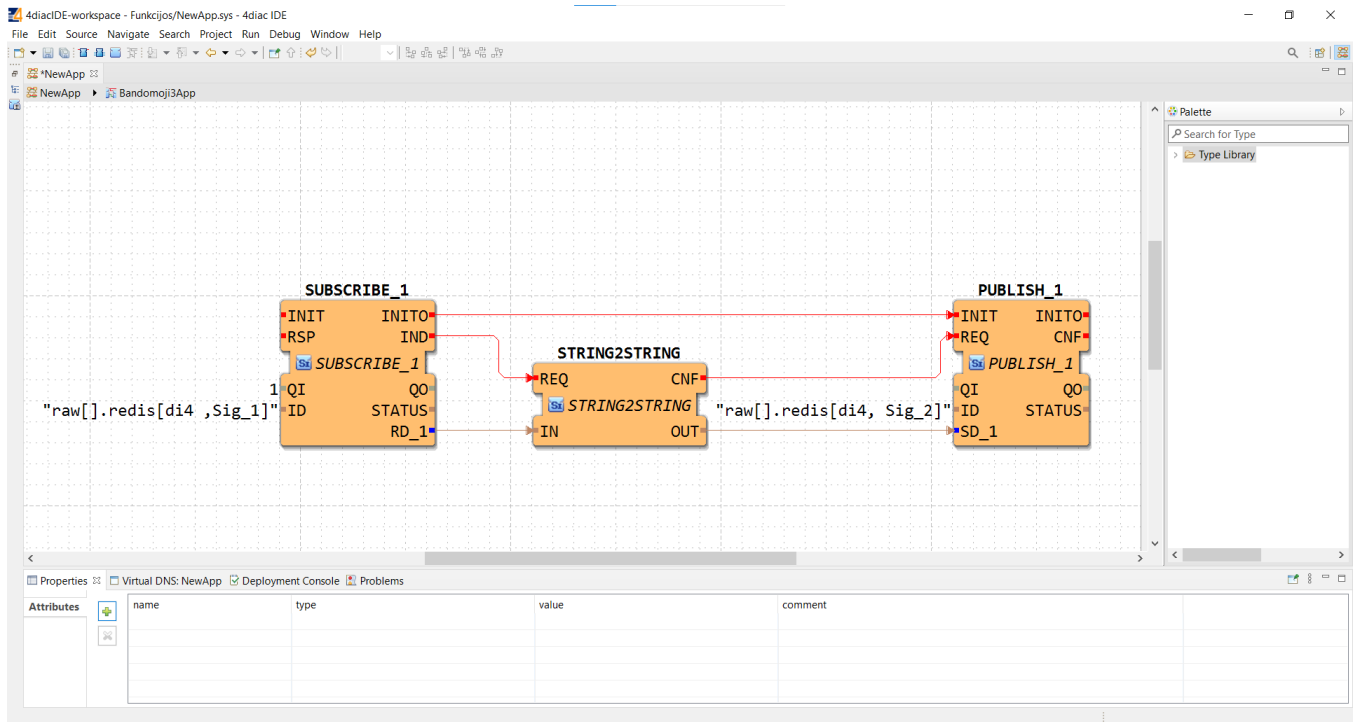
Parameters mentioned earlier, namely device_alias and signal_alias, are the only parameters one needs to fill to bind Excel configuration to 4Diac framework. Two types of blocks are used for data transmission - PUBLISH blocks to write data to REDIS database and SUBSCRIBE blocks to acquire data from database as soon as it changes its value. Both of them have an ID connection. To connect a block to a data point, one should set this pin as raw[].redis[device_alias,signal_alias], e.g. raw[].redis[example_plc_device,example_plc_signal_alias]. As of version 2.0.0 the pin values must be surrounded by double commas.

Examples with SUBSCRIBE and PUBLISH function blocks are shown in the images below.

Subscribe and publish example in the older versions

⚠ Outputs of variable type ANY cannot be directly wired to inputs of the same type and therefore need to be explicitly typed using transitional function blocks.

⚠ No more than 20 tags should be published over a period of 5 seconds, as this may overfill the queue. A "publish only on change" policy is advised.

⚠ Currently only PUBLISH_1 and SUBSCRIBE_1 function blocks are supported.

⚠ The functionality of the subscribe function is dependent on the presence of the "F_STRING_TO_" function block, for example (F_STRING_TO_REAL). Likewise, the publish function's proper operation relies on the availability of the "F__TO_STRING" FB, for example (F_REAL_TO_STRING).

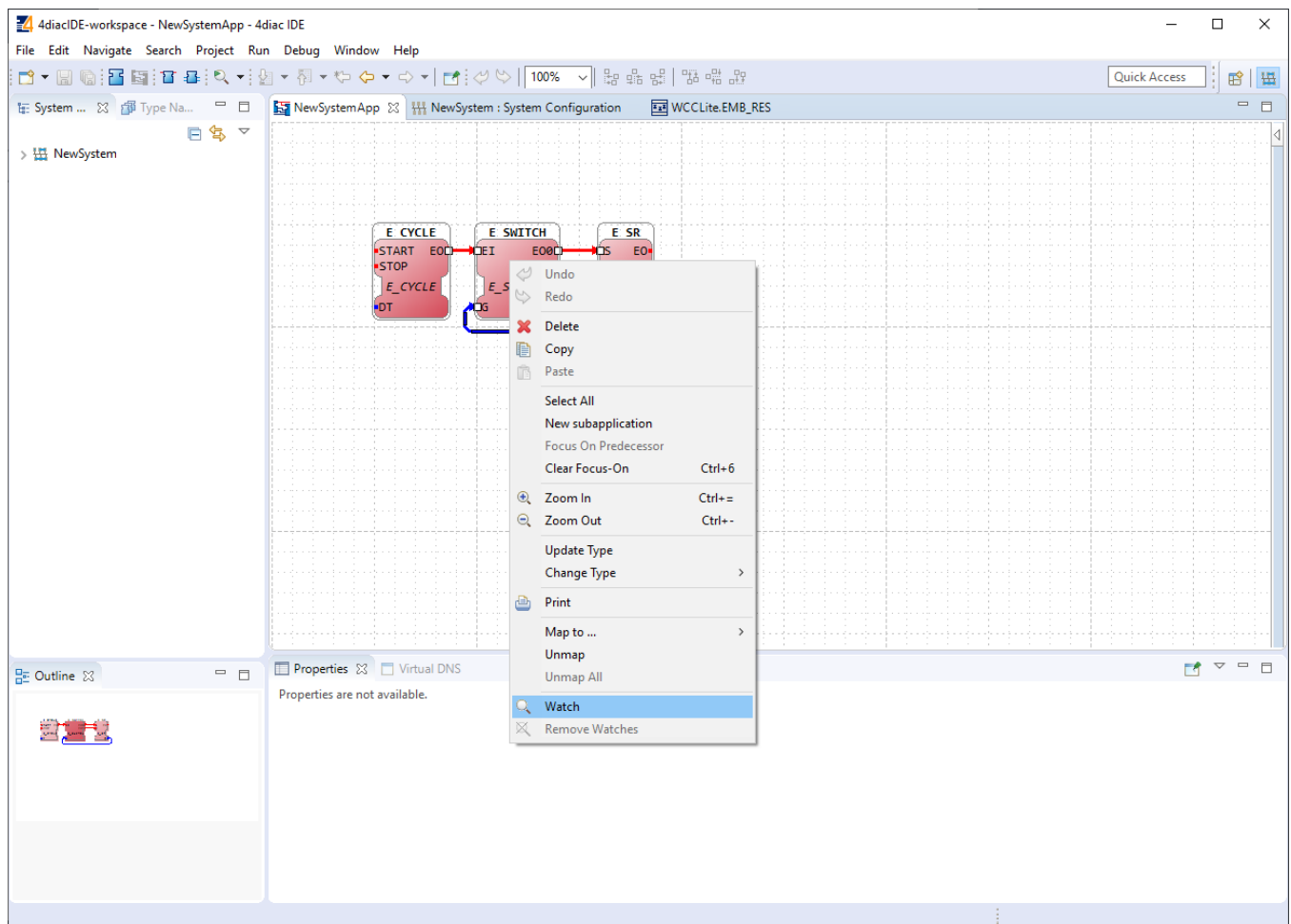If every step until now has been successful, the user can now proceed to debug the PLC application.

20.5 Debugging an IEC 61499 application

After a project has been built and binded to an existing Excel configuration, a user would normally want to check if every part is working according to the prior requirements before compiling finished project and uploading it to production. Both 4Diac framework and WCC Lite offer tools for flexible debugging.
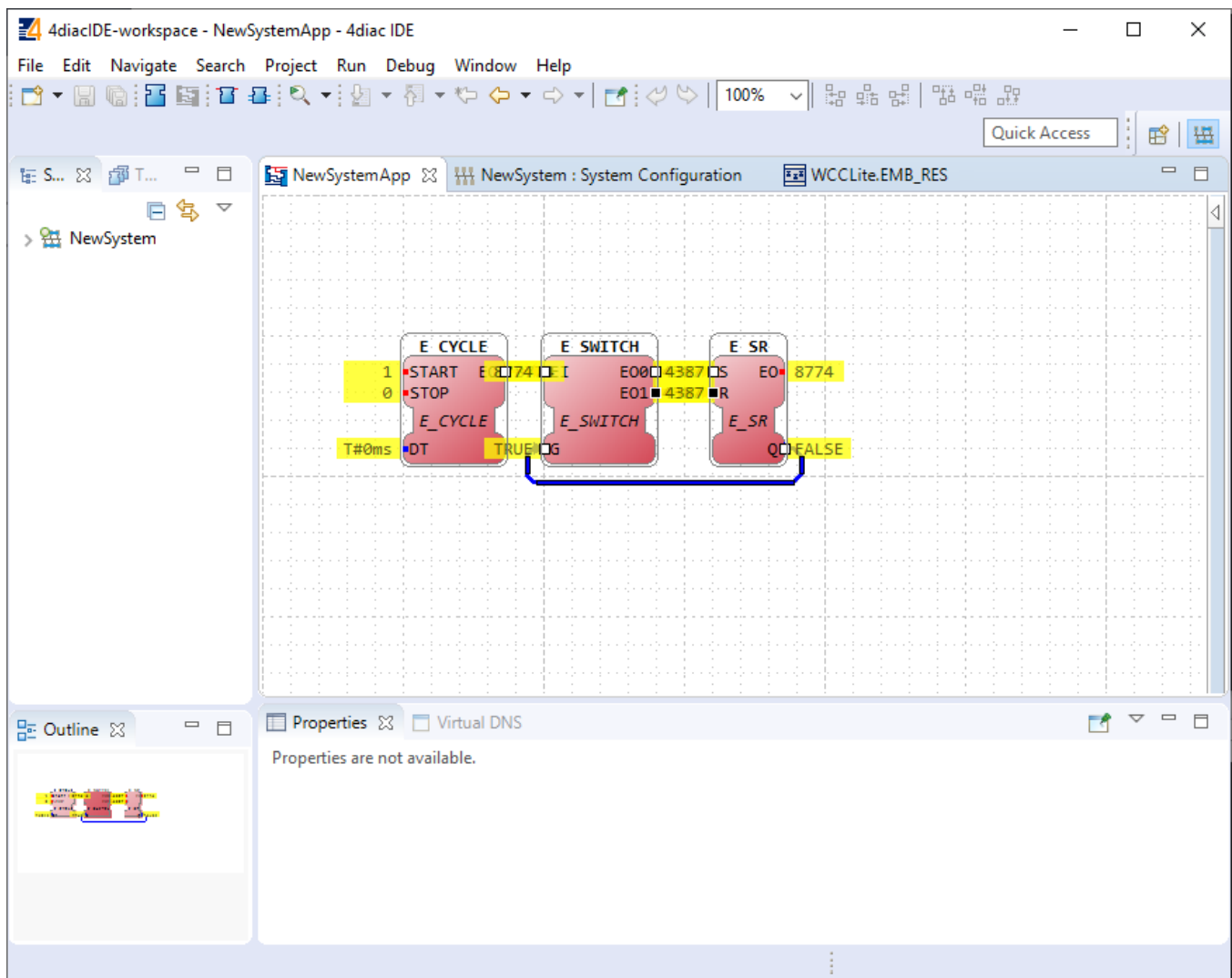
There is a possibility that 4Diac FORTE might not start as a process. It may happen if multiple faults occurred and process has stopped. Process is also programmed to not start if no excel configuration file is found, therefore a user should make sure that Excel configuration is uploaded and ready for use.

Individual function blocks can be set to Watch mode: events can be triggered and values can be forced at inputs or outputs (look into images bellow). To monitor the function blocks, the application should be deployed and the IDE should be in Online mode (Debug --> Monitor System --> NewSystem).

Selecting watch mode:

Function blocks in watch mode:

Seeing information dynamically updated on 4Diac IDE might be very informative, however, some applications might require accessing WCC Lite via command-line interface. For example, in case of information not being updated one would want to assure that 4Diac FORTE in WCC Lite is not filtering data out but sending it to internal database (Redis). To run 4Diac FORTE debug from command-line interface, a user should write forte and press Enter. All possible choices are shown by adding -h flag. A debugging example would be: '/etc/init.d/forte stop' to stop any forte processes, 'forte -d7 -r -f /etc/forte/yourfbootfilename.fboot' to launch the debug. More flags are shown in a Table bellow. Make sure to stop any running process that could use the address that 4Diac framework is going to use.
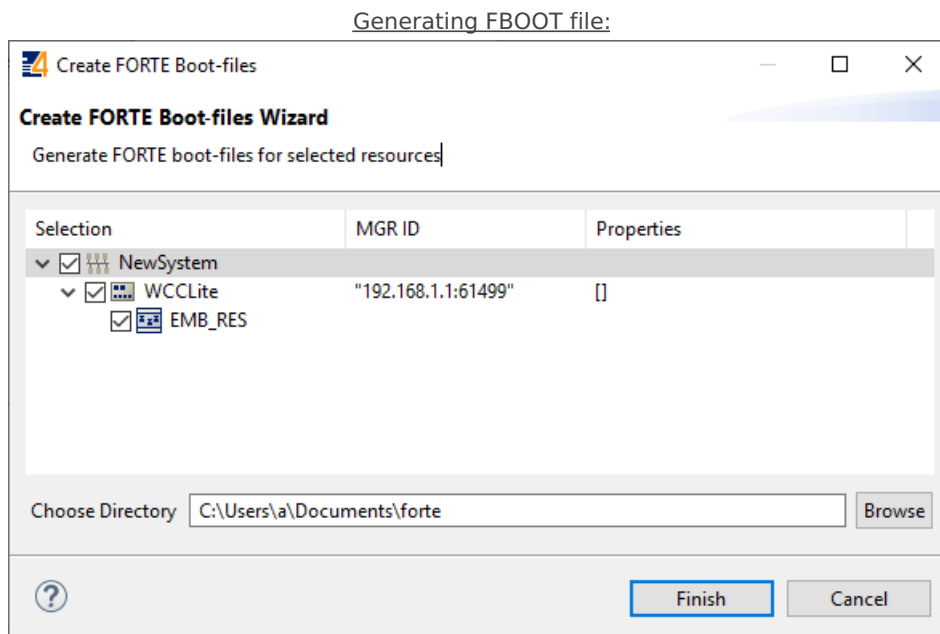
Table. 4Diac FORTE command line debugging options:

```
 -h - Display help information
 -c <IP>:<port> - Set the listening IP and port for the incoming connections
 -r - Show redis messages
 -d <debug level> - Set debugging level
 -f - Set the boot-file where to read from to load the applications
```

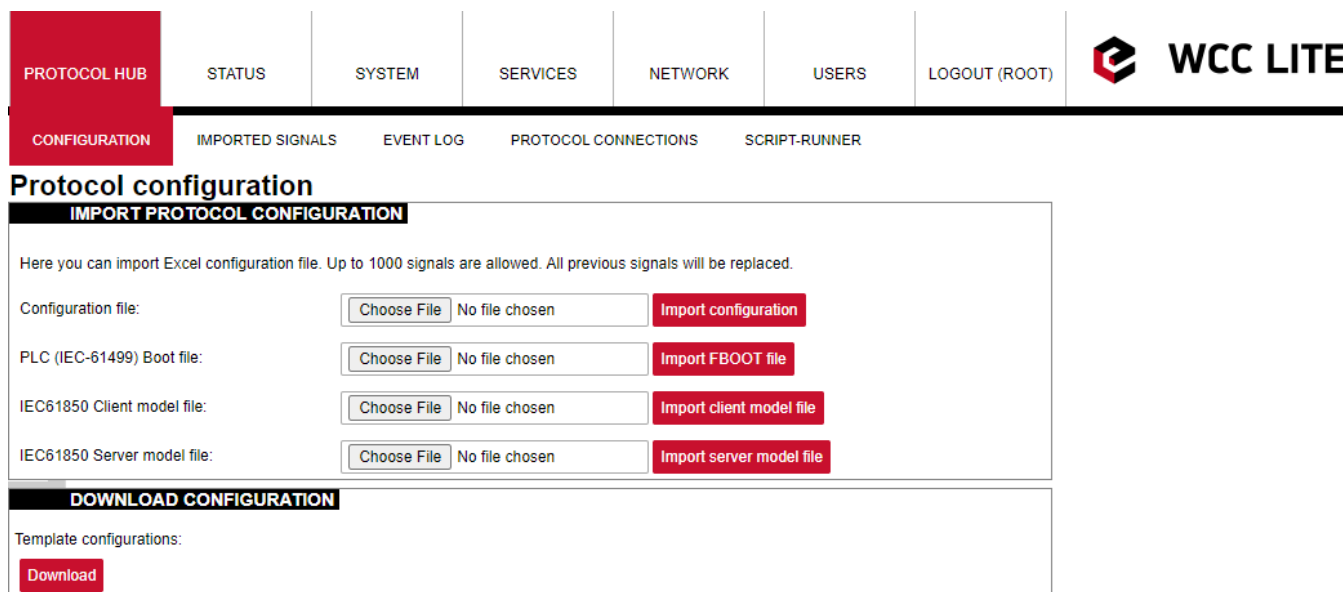# 20.6 Generating and uploading FORTE logic file

After the PLC design is finished and debugged, such design can be compiled into FBOOT file and uploaded to one or multiple devices to be used in production. As application being debugged is not automatically considered as a default application, one should be uploaded explicitly via web interface.

To generate FORTE boot-files a user should select Run->Create FORTE boot-file.... After that one should select devices which should have their boot files created as well as additional devices' properties and directory where these files should be stored as in picture bellow.

Generating FBOOT file:

Upload button for FORTE file in web interface can be found in Protocol Hub tab, Configuration screen (FORTE boot file upload supported for versions v1.4.0 and above). You should see a view as in picture below.

WCC Lite Web interface. Upload and download of 4Diac configuration files:

After the file has been imported one should be able to download it from the same screen as seen in the picture before.

⚠ Please note that only files with *.fboot extension are allowed.

Uploading a file saves it's name and shows it in the web interface. It is advised to carefully choose a filename to separate different versions of PLC application files.

20.7 Distributed control application

IEC 64199 standard introduced requirements for a distributed control. This means that multiple devices can change information between them and make their own decisions based on the data they receive from other sources. This enables distributed applications between multiple WCC Lite devices and all other devices that support IEC 61499.

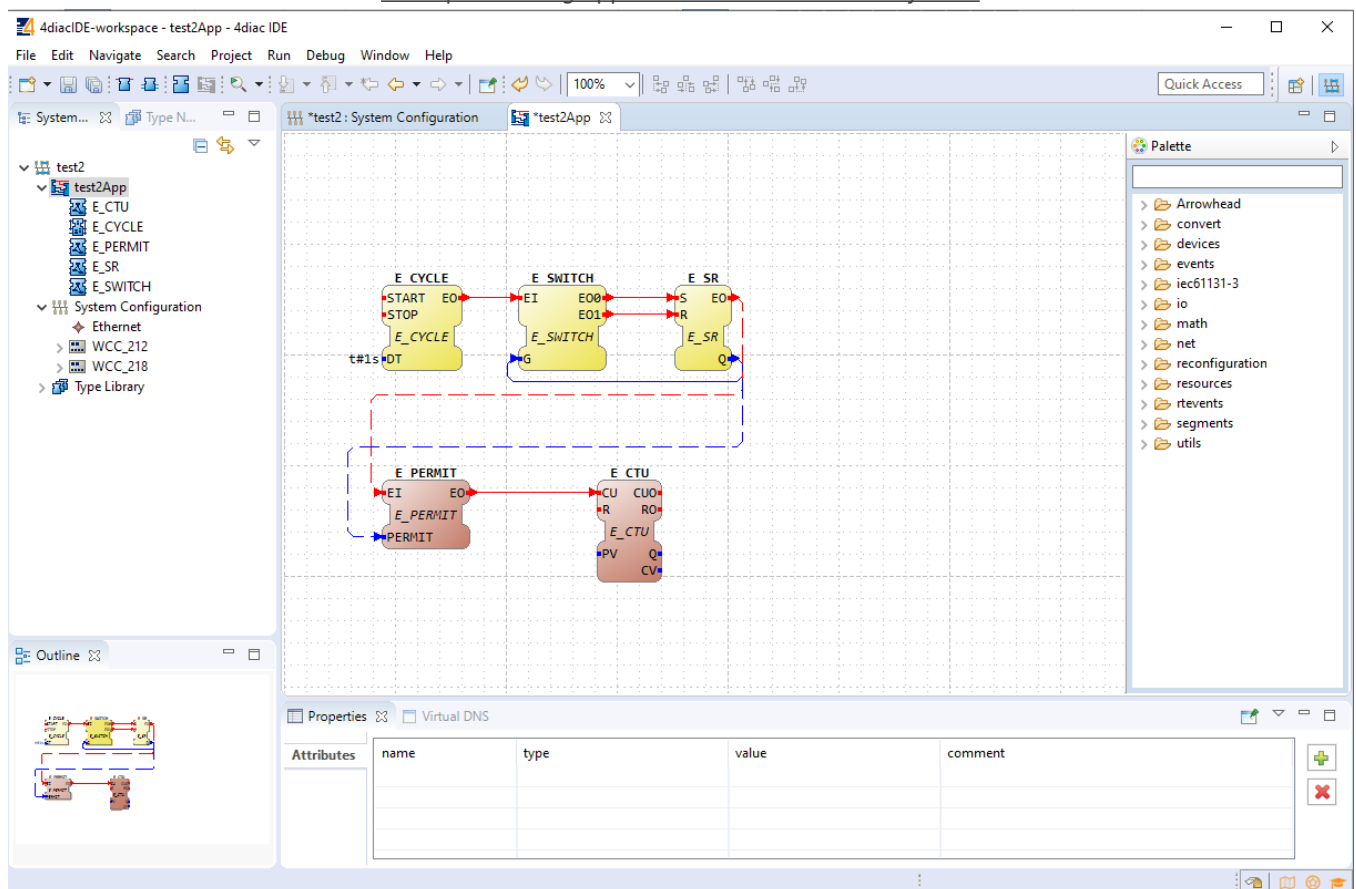Communication between devices can be configured using:

- Publish/Subscribe function blocks (via UDP packets);
- Client/Server function blocks (via TCP packets).

A Publish block can publish data messages using UDP multi-cast addresses meaning that multiple devices would be able to simultaneously get the same data. However, one would have to make sure that all of the devices support multi-cast option.
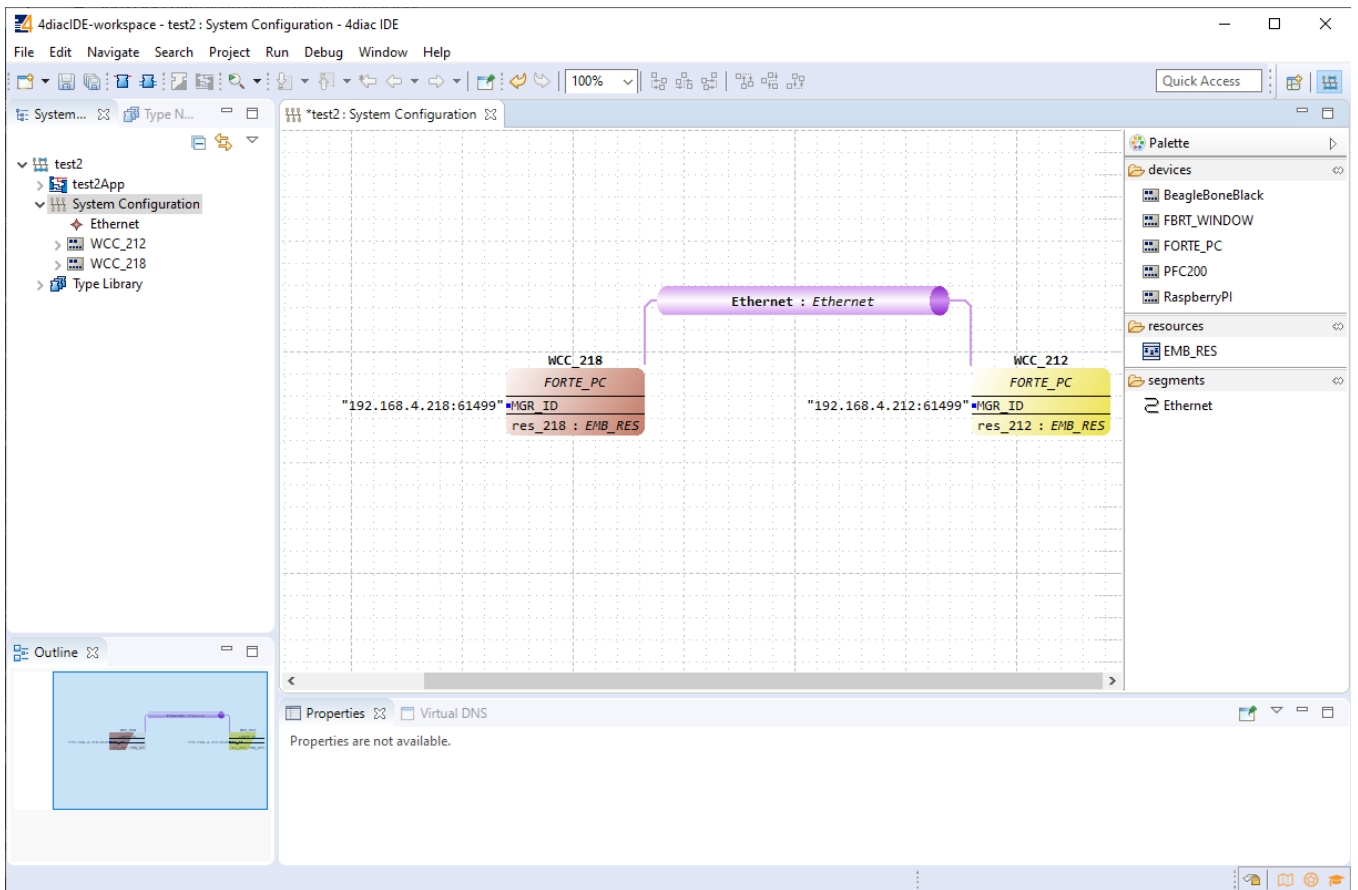
This user manual will only cover setting up point-to-point communication between devices via Publish/Subscribe blocks. For more information on communication between several IEC 61499 devices please check documentation for **Eclipse 4diac framework**.

Let's say we would like to count how many times the light has been turned on. For this we can add counting functionality to application shown in picture below. The application should run on 2 devices. The blinking part of the application will run on a 4diac FORTE and the count on another 4diac FORTE, see the architecture below. The two different programs running on two separate WCC Lite devices emulate two PLCs. Two different devices can be identified by different colors of function blocks. One can identify device and it properties by accessing System Configuration screen as seen below. Yellow function blocks belong to WCC_212 device which can be accessed through 192.168.4.212 (port number 61499) whereas brown function blocks belong to WCC_218 device which can accessed through 192.168.4.218 (port number 61499).
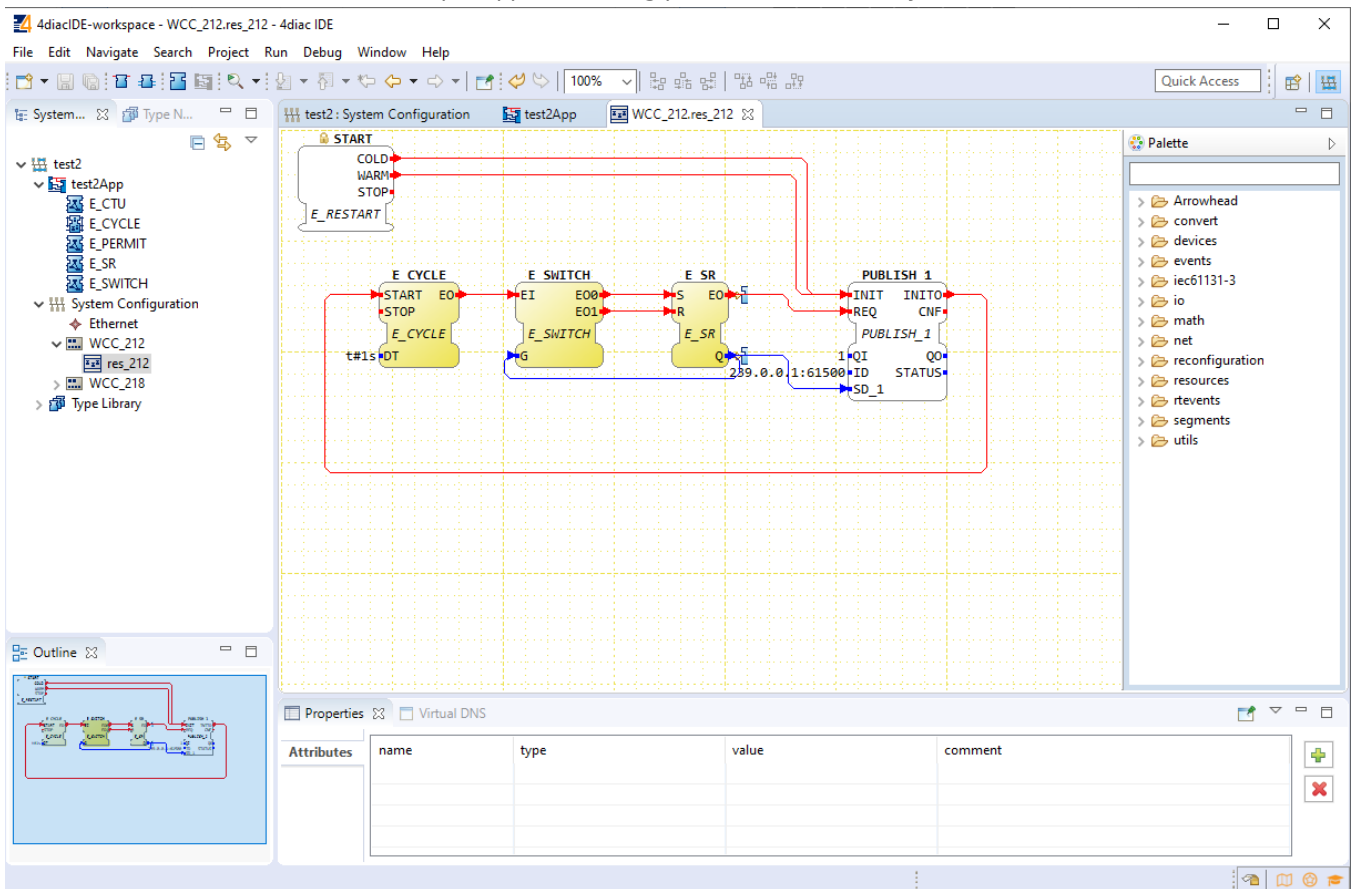
Example blinking application as a distributed system:

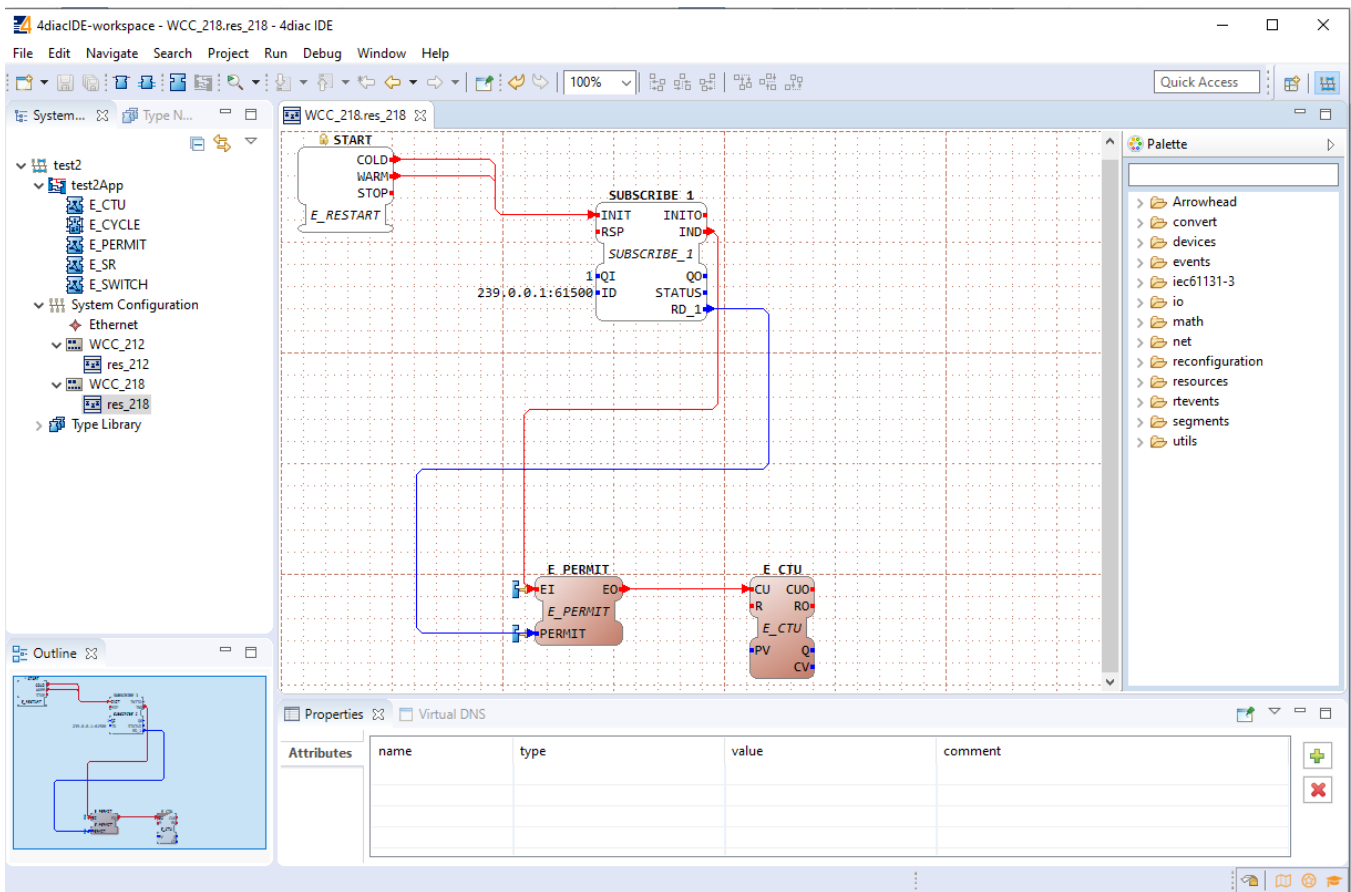Example system configuration for a distributed system:

Example app for blinking part of a distributed system:

Example app for counting part of a distributed system:

To count the blinking, two new Function Blocks (FBs) have been added to the existing application for a different device (WCC_218):

- E_PERMIT
- E_CTU

To communicate between devices, an additional PUBLISH_X/SUBSCRIBE_X pair must be used. As one can identify, these blocks are not seen when looking at a whole distributed system and should be seen as an intermediary between devices.

The PUBLISH_X FB is used to send messages over the network which are received by an according SUBSCRIBE_X FB. Every time a REQ is triggered, a message is sent according to the ID input. With the value of the ID input you can specify what specific network protocol you would like to use (e.g., MQTT). If you don't specify a dedicated protocol the default as defined in the "IEC 61499 Compliance Profile for Feasibility Demonstrations" is used. The number X in PUBLISH_X is the number of data elements that you want to send in the message. Since we are only sending one value we used PUBLISH_1.

The used ID value specifies an IP:PORT pair.