pbsSoftLogic Beckhoff TwinCAT/BSD Configuration

May 2026

Ver. 1.0

1 - pbsSoftLogic Overview

pbsSoftLogic is an open, modular RTU/PLC programming environment designed for industrial automation , SCADA and real-time control applications. Its runtime kernel operates across multiple Linux-based operating systems, and pbsControl provides a fully compiled and optimized runtime specifically for the Beckhoff TwinCAT/BSD operating system.

pbsSoftLogic provides the following core capabilities:

- **Intuitive Function Block Programming**
  A modern, user-friendly environment for developing logic using standard and custom function blocks.
- **Extensive Function Block Library**
  More than **400 fully documented** blocks, including Timers, Counters, IEC 61131-3 logic, process blocks, signal generation, mathematical operations, and more.
- **Lua-Based User Function Blocks**
  Support for creating custom logic using Lua scripting, enabling advanced and dynamic behavior.
- **Isolated Protocol and Logic Configuration**
  Communication protocols are configured through the GUI, while logic is developed independently using function blocks—ensuring clean separation of concerns.
- **Comprehensive Protocol Support on TwinCAT/BSD**
  Includes TwinCAT ADS, EtherCAT Master, Modbus TCP/RTU Master and Slave, DNP3/IEC-104 Slave with TLS and IEC 62351 security, DNP3 Master, IEC-104 Master, BACnet Client/Server, and MQTT.
- **Advanced Logic Monitoring and Online Debugging**
  Real-time signal monitoring, forcing of variables, and both warm and cold logic updates without interrupting system operation.

2-Logical Connection Between TwinCAT and pbsSoftLogic

The figure – 1 illustrates the logical data flow between TwinCAT and pbsSoftLogic. Communication with the TwinCAT runtime is performed through the **ADS driver**, which enables direct read/write access to TwinCAT variables. For hardware-level interaction, pbsSoftLogic can also read and write signals directly from EtherCAT I/O modules using the integrated **EtherCAT Master Driver**.

pbsSoftLogic additionally supports data exchange through **Modbus TCP/RTU**, allowing communication with I/O modules, meters, external devices, inverters, and other field euipment.

All acquired data—whether from ADS, EtherCAT, or Modbus—can be mapped to **DNP3/IEC-104/MQTT** using either function-block-based mapping or the internal tag-buffering mechanism. This provides a flexible and unified pathway for exposing process data to SCADA, RTUs, or control centers.

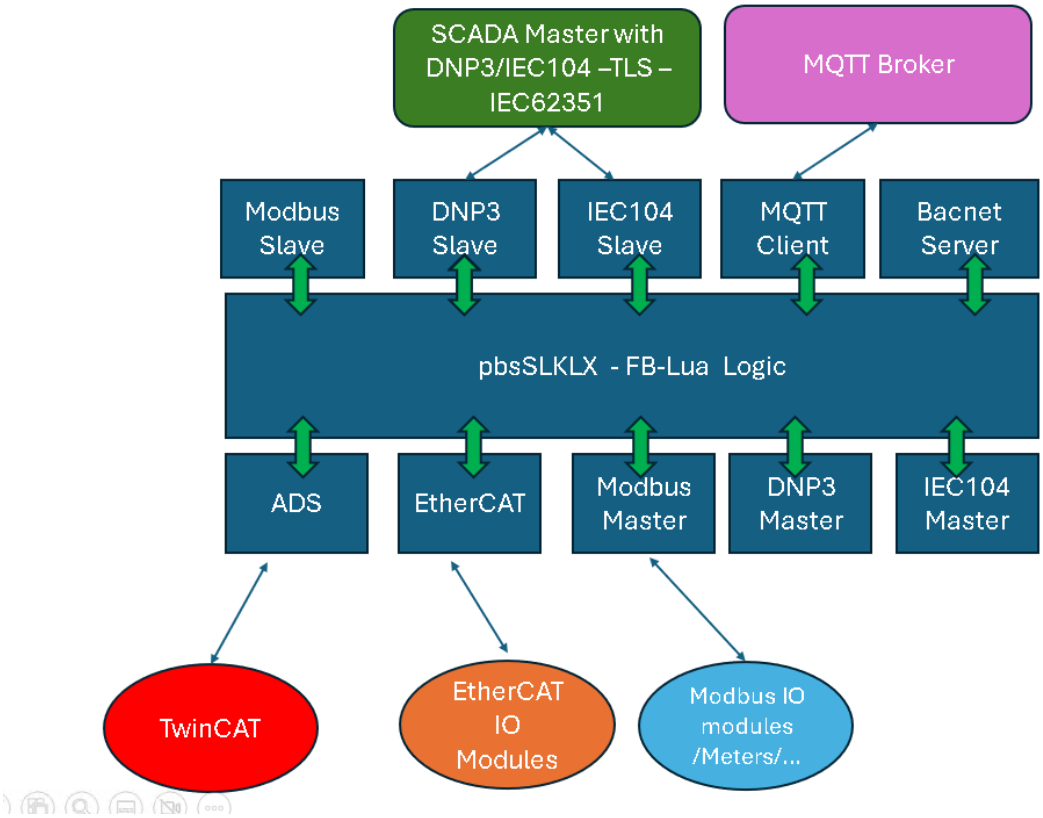


Figure 1

3 - Installation of Runtime kernel for TwinCAT/BSD

pbsSoftLogic uses secure SSH communication to transfer project configuration files directly to the target IPC during deployment. For systems configured to operate with the *root* account, SSH access must be explicitly enabled on TwinCAT/BSD. This is done by editing the SSH daemon configuration file located at */etc/ssh/sshd_config* and changing the *PermitRootLogin* directive from “no” to “yes.” After saving the modification and restarting the SSH service, the IPC becomes fully accessible to pbsSoftLogic for automated project upload, configuration synchronization, and runtime management using the root user credentials. Figure 2 ,3

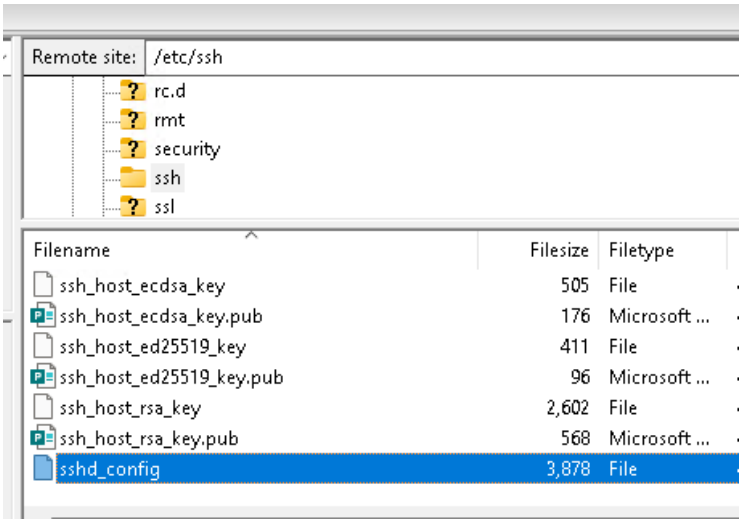


Figure 2

```
33
34  #LoginGraceTime 2m
35  PermitRootLogin yes
36  #StrictModes yes
37  #MaxAuthTries 6
38  #MaxSessions 10
39
```

Figure 3

In this document, we assume that the IPC is accessed using the default *root* user with the password *"root"* for simplicity and demonstration purposes. However, pbsSoftLogic does not require the use of the root account; the runtime kernel can operate under any valid system user. You may configure and deploy projects using any user and password combination permitted by your TwinCAT/BSD security policy, allowing you to align pbsSoftLogic with your preferred access-control and cybersecurity practices.

The pbsSoftLogic runtime kernel is located in the */home/pbsLX* directory on the IPC. After downloading the runtime package, unzip it on your development machine and transfer the extracted files to the IPC's */home/pbsLX* folder using FileZilla or any other SFTP-compatible utility. When using FileZilla, ensure that the file transfer mode is set to **Binary** in the *Edit → Settings* menu; transferring Linux executables in ASCII mode will corrupt the files and prevent the runtime kernel from operating correctly. Figure 4

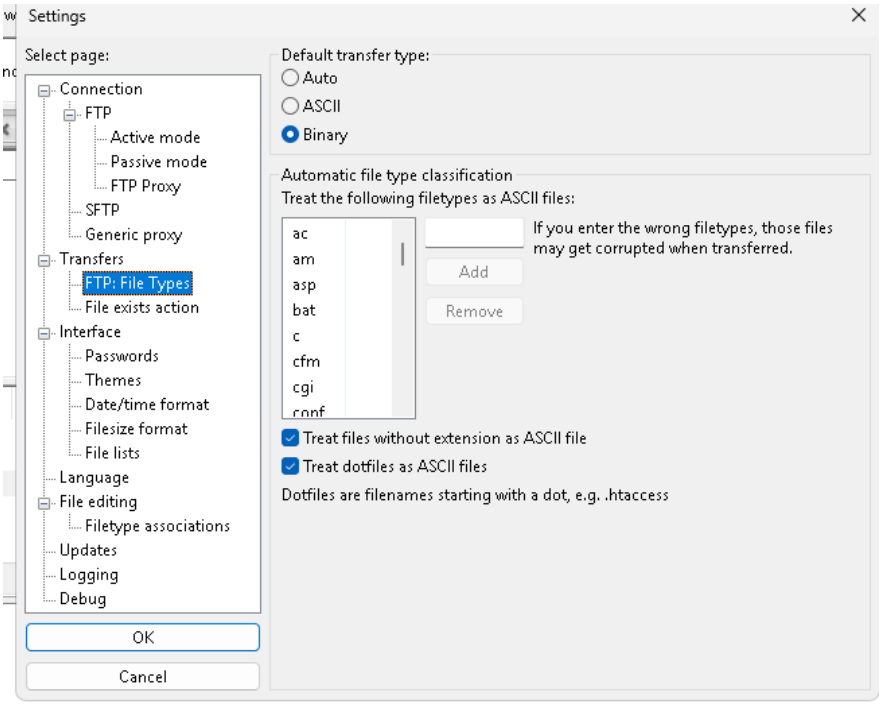


Figure 4

pbsSoftLogic runtime kernel has following structure: Figure 5

/home/pbsLX/

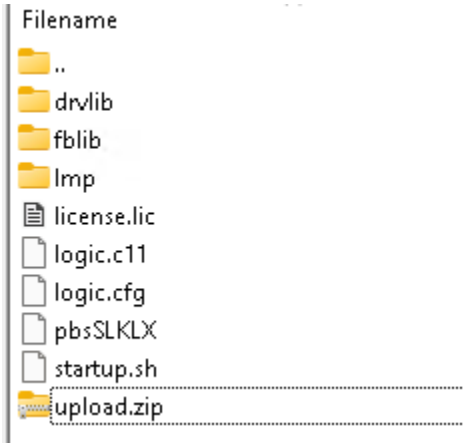


Figure 5

The pbsSoftLogic runtime kernel is organized into several key folders and files within the */home/pbsLX* directory:

- **drvlib** – Contains all communication protocol modules used by the runtime.

- **fblib** – Stores the complete library of function-block definitions.
- **lmp** – The Logic Monitoring Protocol (LMP) subsystem, responsible for online logic monitoring, tag forcing, and warm/cold restarts; this folder also includes the Lua interface for user-defined function blocks.
- **pbsSLKLX** – The main runtime kernel executable for pbsSoftLogic.
- **license.lic** – The license file required for each IPC. It is activated through the pbsSoftLogic IDE. System integrators and Beckhoff offices may use a free internal license, while deployed projects require a purchased license linked to the controller's UUID.
- **logic.cfg** – Contains project settings and protocol configuration for the deployed project.
- **logic.c11** – The compiled logic generated by the pbsSoftLogic IDE. Both *logic.cfg* and *logic.c11* are transferred to the IPC via SFTP during deployment.
- **startup.sh** – The startup script that loads the runtime kernel and applies system settings during IPC boot.
- **upload.zip** – The most recently transferred project package stored on the IPC.

pbsSoftLogic requires the */mnt/ramdisk* directory to store the static data used by Function Blocks during runtime. To prepare the system, create the */mnt/ramdisk* folder on the IPC and convert it into a real RAM-backed filesystem by adding the following line to the */etc/fstab* file:

```
tmpfs    /mnt/ramdisk    tmpfs    rw,size=10M    0 0
```

This ensures that pbsSoftLogic has a high-speed, volatile memory area for handling Function Block static data without causing unnecessary wear on the IPC's storage device.

Automatic startup of pbsSoftLogic kernel

To ensure that the pbsSoftLogic kernel starts only after the TwinCAT system services have fully initialized, pbsSoftLogic is launched through a dedicated startup script that is triggered once the TwinCAT runtime has completed its own boot sequence. TwinCAT/BSD starts its kernel using the system service *TcSystemService*, which runs early in the boot process and prepares the TwinCAT real-time environment.

Because we want to start pbsSoftLogic kernel after TwinCAT Services , we will add startup.sh at end of "/usr/local/etc/rc.d/ TcSystemService"

Edit "/usr/local/etc/rc.d/ TcSystemService" service and add /home/pbsLX/startup.sh & at the end of file . Figure 6

```
TcSystemService_poststart() {
    printf 'Waiting for ADS to become available..' >&2
    while ! /usr/local/bin/ads 127.0.0.1.1.1 state >/dev/null 2>&1; do
        printf '.' >&2
        sleep 1
    done
    printf '\n' >&2
}

TcSystemService_poststop() {
    rm -f "${pidfile}"
}

TcSystemService_stop() {
    pkill -QUIT -f "${pidfile}"
    pwait "$(cat "${pidfile}")"
}

load_rc_config $name
: "${TcSystemService_enable:=no}"
run_rc_command "${1}"
/home/pbsLX/startup.sh &
```

Figure 6

Adding necessary ports to firewall

pbsSoftLogic requires several communication ports to be accessible through the TwinCAT/BSD firewall. TwinCAT/BSD uses the FreeBSD *pf* packet filter, and all firewall rules are defined in the system configuration file /etc/pf.conf. To enable external access to the Modbus/TCP port (502), DNP3 port (20000), pbsSoftLogic LMP port (20001), and IEC-104 port (2404), the corresponding rules must be added to the firewall configuration. This is done by editing /etc/pf.conf and inserting pass rules for each required TCP port. After saving the changes, the firewall must be reloaded to apply the new configuration. Once these rules are active, TwinCAT/BSD will allow inbound connections on ports 502, 20000, 20001, and 2404, ensuring that all pbsSoftLogic protocol drivers operate correctly across the network. Figure 7

```
set skip on lo0
scrub in all
# allow dynamic NAT configuration (e.g. luemuctl)
nat-anchor "bhf-nat/*"
# allow dynamic port forwarding configuration (e.g. luemuctl)
rdr-anchor "bhf-rdr/*"

# block all incoming and allow all outgoing traffic
block return in all
pass out quick all

# allow icmp6
pass in quick proto icmp6 all

# allow icmp4 (ping)
pass in quick inet proto icmp all icmp-type { echoreq, unreach }
# allow pbsSoftLogic Protocols
pass in on em0 proto tcp from any to any port 48898
pass in on em0 proto tcp from any to any port 20001
pass in on em0 proto tcp from any to any port 502
pass in on em0 proto tcp from any to any port 20000
pass in on em0 proto tcp from any to any port 2404
```

Figure 7

The rule pass in on em0 proto tcp from any to any port 20001 instructs the firewall to allow inbound TCP traffic arriving on the network interface *em0*—the primary Ethernet interface used on TwinCAT/BSD. This rule accepts connections from any remote IP address and forwards them to the device on port 20001, ensuring that the pbsSoftLogic communication service (LMP) is reachable across the network.

At this stage, the installation and system integration of pbsSoftLogic on TwinCAT/BSD are fully completed. The runtime kernel is configured to start automatically after the TwinCAT services, and all required communication ports are enabled through the firewall. You can now launch the pbsSoftLogic IDE from your development workstation, connect to the target device, and begin creating your control logic. All protocol drivers—including Modbus/TCP, DNP3, IEC-104, BACnet/IP, ADS, and MQTT—can now be configured directly within the IDE, allowing you to build, test, and deploy complete automation projects on the TwinCAT/BSD platform.

4-ADS Driver Configuration

In this section, we will create a simple pbsSoftLogic project and demonstrate how to read and write tags from a TwinCAT PLC using the ADS Driver. Begin by launching the pbsSoftLogic IDE on the same engineering workstation where TwinCAT XAE is installed. The ADS Configurator inside the IDE requires the Beckhoff ADS library to communicate with the IPC and access PLC symbols. Once the IDE starts, the

main workspace will appear, allowing you to create a new project, configure communication drivers, and define logic. The following image shows the initial IDE interface that appears after startup, from which you can begin configuring the ADS connection and mapping TwinCAT tags into your pbsSoftLogic application. Figure 8

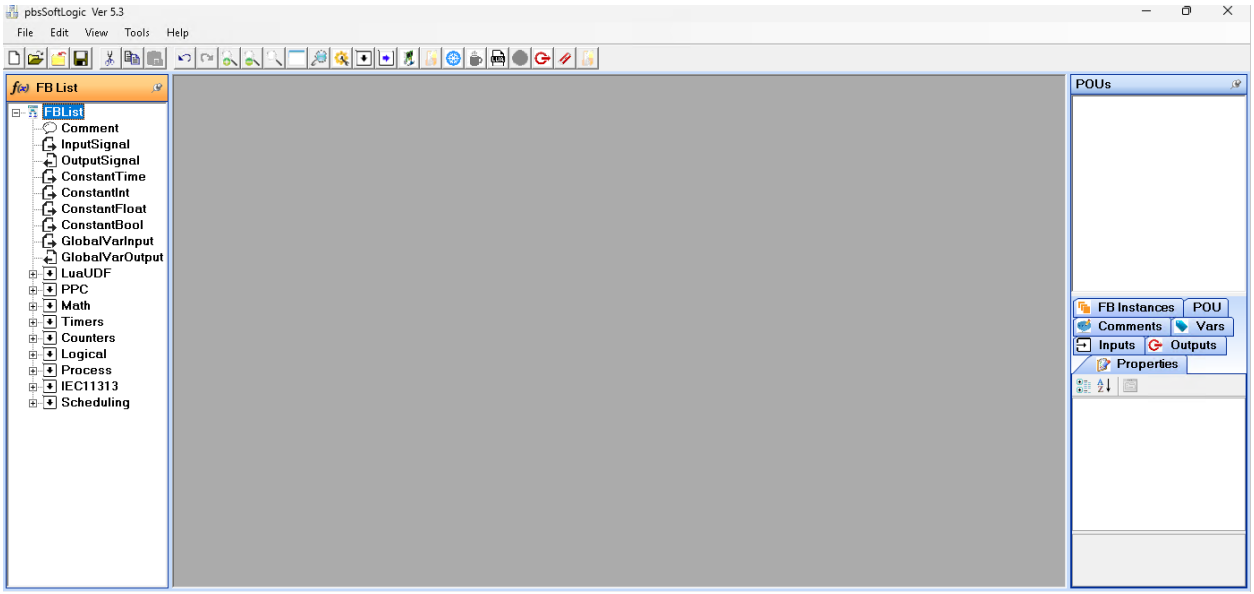


Figure 8

From the *File* menu, select **New Project** to begin creating your first application. Choose the folder where you want to store the project files, then enter a name for the project. The IDE automatically creates a new directory using the same name you provide and saves all project-related files inside that folder. In the example shown in the following figure, a new directory named **"Test1"** is created by the IDE, and the project itself is also named **"Test1"**. This structure keeps each project self-contained and makes it easy to manage multiple applications on the same development workstation. Figure 9

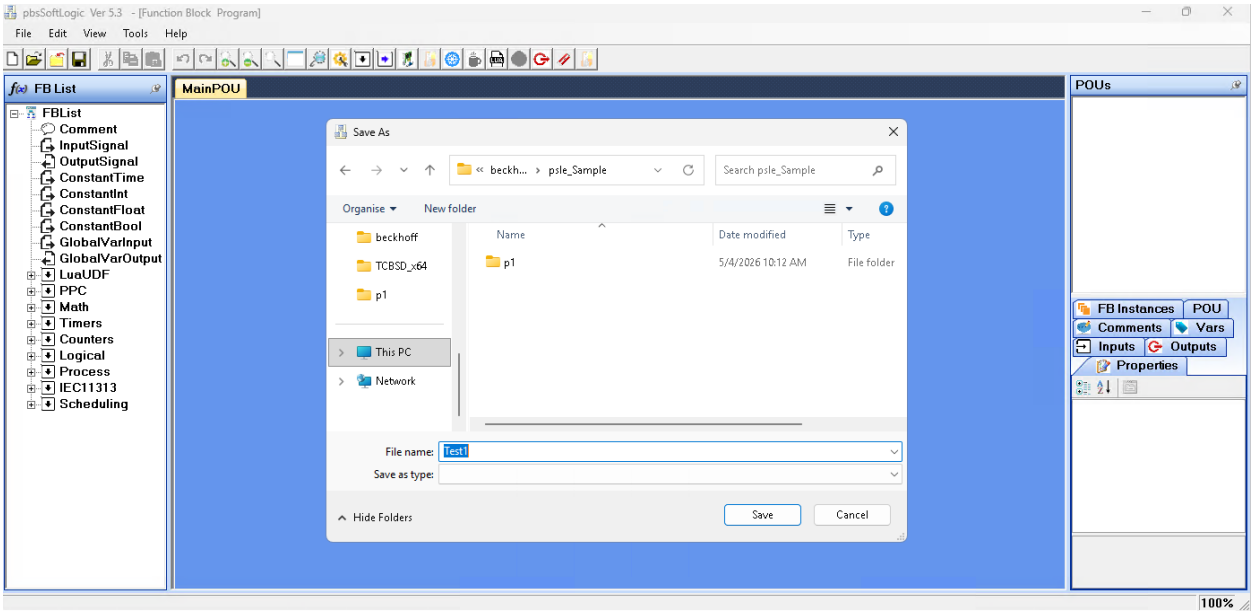


Figure 9

After entering the project name and selecting the destination folder, click the **Save** button to finalize the creation of your new project. The IDE will automatically generate the required project structure, create all initial configuration files, and prepare the workspace for development. Once the project is saved, the environment is fully initialized and you can proceed with driver configuration, logic programming, and adding communication interfaces such as the ADS Driver. At this point, your project is ready for the next steps in building and testing your automation application.

The pbsSoftLogic IDE is organized into three main panels to simplify development and navigation. On the **left panel**, you will find the complete list of supported Function Blocks and all FB programming elements that can be used in your application. The **right panel** displays the list of POUs (Program Organization Units) within the project, along with the properties of the currently selected element. At the center of the workspace is the **main POU editor**, which is always open and cannot be closed. This central panel serves as the primary area for writing logic, placing function blocks, and building your control program.

pbsSoftLogic allows you to define an unlimited number of POUs (Program Organization Units) within a project, giving you full flexibility to structure and organize your application. Each POU can be named according to its function, making large projects easier to maintain and navigate. POUs can also be individually enabled or disabled, allowing you to isolate parts of the logic during development or testing. When the project is compiled, the final application does not preserve the POU hierarchy; instead, all Function Blocks and POUs are internally sorted and executed according to their assigned **Execution Number**, ensuring deterministic and predictable runtime behavior.

By clicking on **Project Settings**, you can access all configuration options related to the current project. This section allows you to define the communication protocols used by the controller, such as ADS, Modbus/TCP, DNP3, IEC-104, BACnet/IP, and MQTT. In addition to protocol configuration, the Project Settings panel provides access to general controller parameters, execution settings, and runtime

behavior options. Any changes made here apply directly to the project and will be included in the compiled application, ensuring that the pbsSoftLogic runtime operates with the correct communication interfaces and controller settings. Figure 10

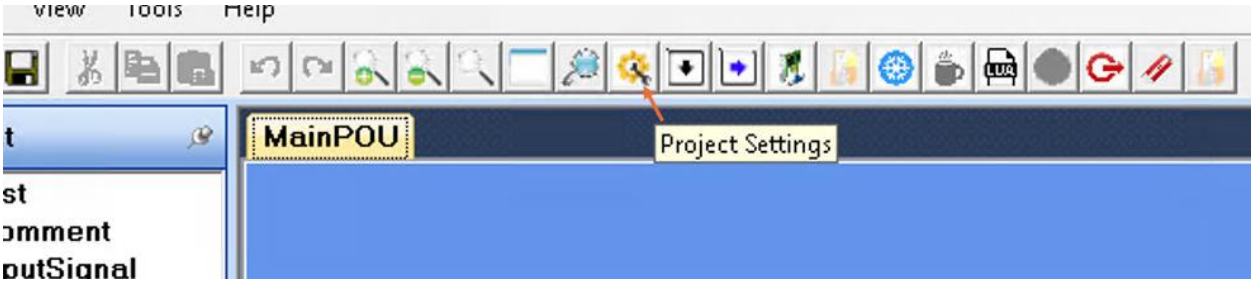


Figure 10

In the Project Settings window, select **"TCBSD"** as the RTU type and enter the IP address of the TwinCAT IPC that will communicate with pbsSoftLogic. The **Logic Scan Time** parameter defines the execution period of the controller. During each scan cycle, the pbsSoftLogic kernel performs a complete sequence of operations: it reads all input values from every enabled communication driver, executes the user logic defined in the POUs, and then writes the resulting output values back to the corresponding drivers. This cyclic execution model ensures deterministic behavior and consistent data flow between the logic engine and all connected field devices or TwinCAT systems. Figure 11

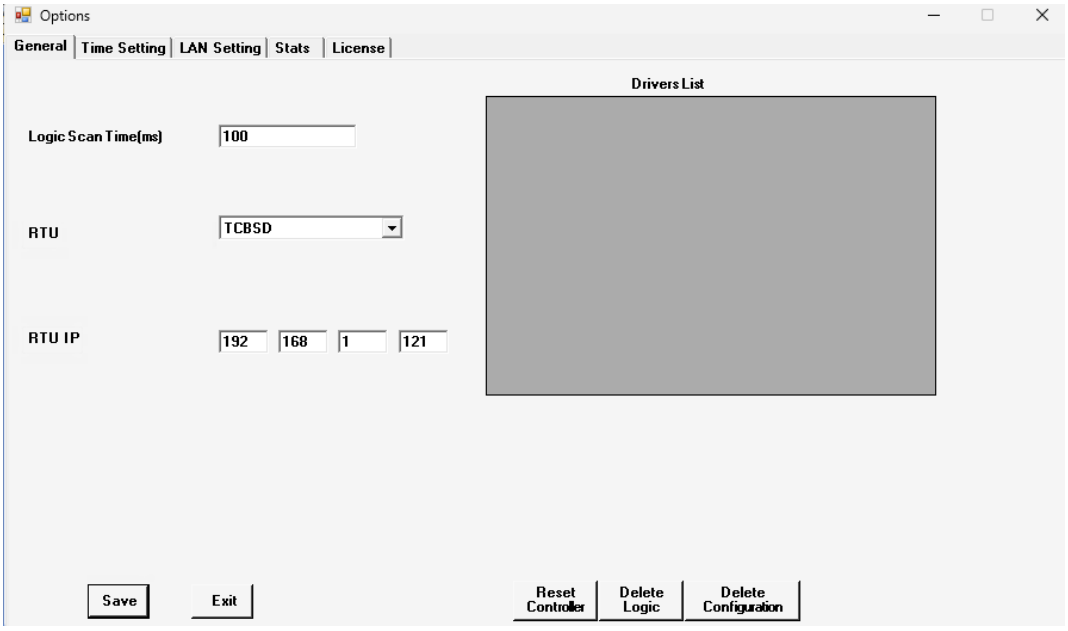


Figure 11

To configure communication with TwinCAT, you must add an ADS driver to your pbsSoftLogic project. In the ”Drivers List” part , right-click and select **Add New Driver**. From the list of available drivers, choose **“BEC_ADS”** and assign a unique name to the driver instance. If your project contains multiple drivers, each driver must have a distinct name to avoid conflicts during configuration and execution. After creating the ADS driver, it will appear under the Drivers List, ready for parameter configuration and TwinCAT tag mapping. Figure 12

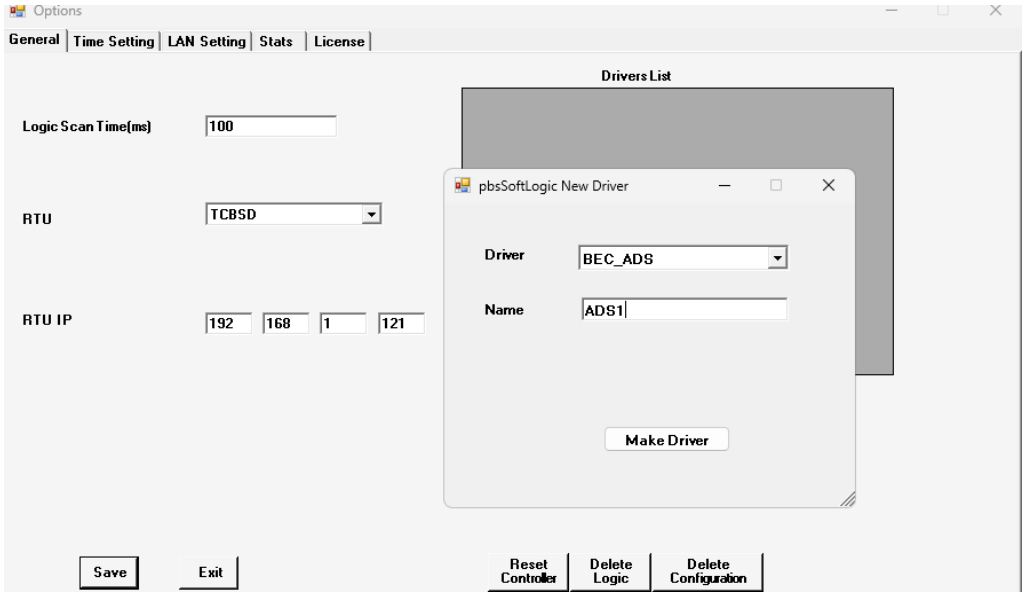


Figure 12

After selecting the ADS driver and assigning a unique name, click the **Make Driver** button. The IDE will automatically generate the basic configuration structure for the driver and add it to the project. This process creates the initial parameter set, allocates the driver entry under the Drivers List, and prepares the configuration fields required for communication with the TwinCAT IPC. Once the driver is generated, you can proceed to define ADS connection parameters, import TwinCAT symbols, and map PLC tags for read/write operations. Figure 13

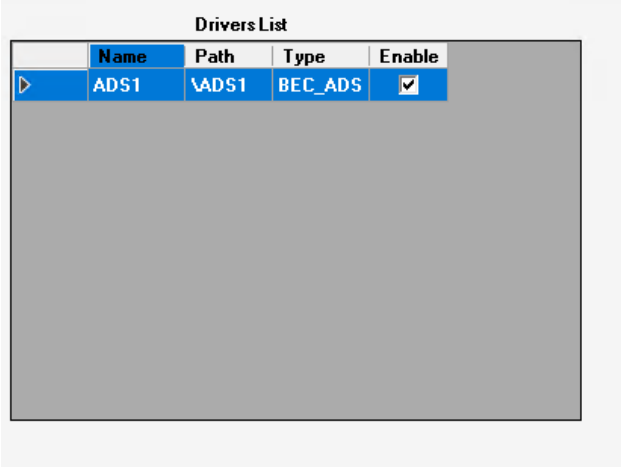


Figure 13

By right-clicking on the driver in the **Drivers List**, you can either open the driver folder using **Explorer** or edit the driver parameters through the graphical **Edit** interface. For every driver added to the project, the IDE automatically creates a dedicated folder using the same name as the driver. This folder contains two essential configuration files: **ADSTags.xml**, which stores all defined driver tags, and **Options.xml**, which contains the communication parameters and connection settings for the driver. These files are maintained by the IDE and updated whenever you modify driver parameters or add new tags, ensuring that the driver configuration remains consistent with the project settings. Figure 14

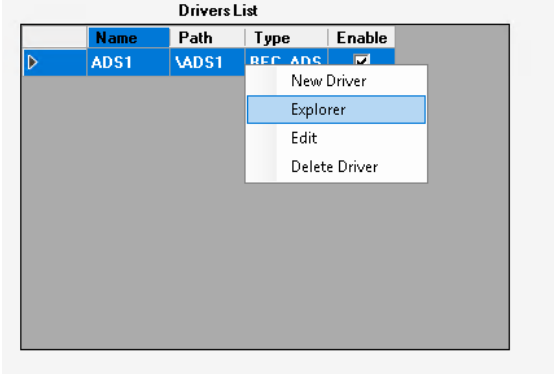


Figure 14

At this stage, we assume that you have already developed a TwinCAT PLC project and downloaded it to the IPC. A TwinCAT project typically contains **global variables**, which are automatically exposed through the ADS protocol and can be accessed by external applications such as pbsSoftLogic. These global tags form the interface between the TwinCAT runtime and the ADS driver, allowing pbsSoftLogic to read and write values directly from the PLC. With the TwinCAT project running on the IPC and its global tags available, you can now proceed to import these symbols into the ADS driver and map them into your pbsSoftLogic application.

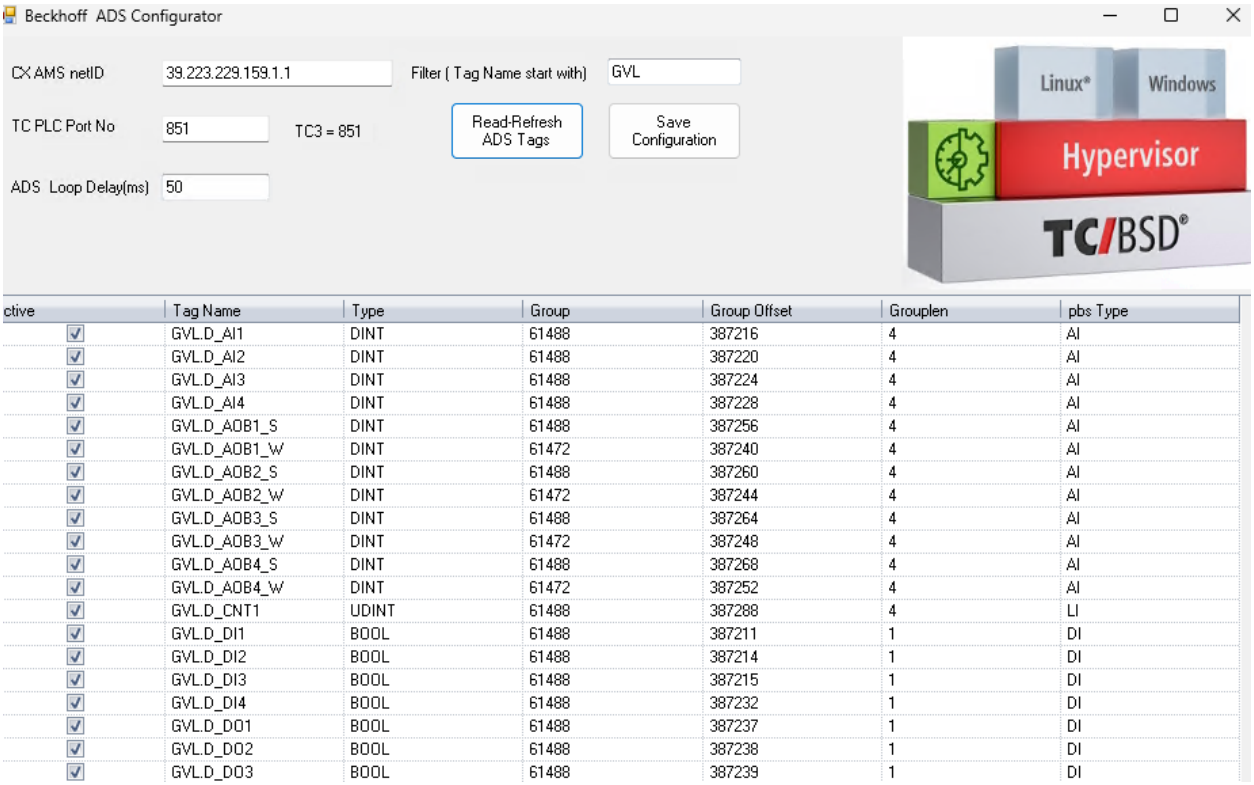
Open the ADS driver configuration GUI to begin setting up communication with the TwinCAT IPC. In the configuration window, enter the **IPC AMS Net ID** exactly as defined in the TwinCAT system. You may apply a filter to limit the list of symbols or clear the filter to retrieve all available global tags. Once the AMS Net ID is entered, click the **Read ADS Tags** button. The IDE will connect to the IPC through the ADS protocol, read the exported TwinCAT symbols, and populate the tag list with all accessible global variables.

For TwinCAT 3 systems, the default ADS communication port is **851**, and this value must be entered in the driver configuration to allow proper access to PLC symbols. The **ADS Loop Delay** parameter defines the waiting time, in milliseconds, that the driver pauses during each read/write cycle. This delay controls how frequently the driver polls the TwinCAT runtime: after completing a full read of all configured tags and writing any updated values, the driver waits for the specified delay before starting the next cycle. Adjusting this value allows you to balance communication load and responsiveness based on the requirements of your application.

In the following example, the filter is set to **"GVL"**, so only TwinCAT tags whose names begin with *GVL* are displayed in the list. For each tag, the ADS driver shows the **Tag Name , ADS data type**, **Group**, **Group Offset**, and **Group Length**, which are required for correct communication with the TwinCAT runtime. Each tag is also assigned a **pbsType**, which defines how pbsSoftLogic will interpret and use the tag during read/write operations. The available pbsType values are:

- **DI** — Driver reads a digital (BOOL) tag from TwinCAT
- **AI** — Driver reads a DINT tag from TwinCAT
- **LI** — Driver reads a UDINT tag from TwinCAT
- **FI** — Driver reads a REAL tag from TwinCAT
- **DO** — Driver writes a digital (BOOL) tag to TwinCAT
- **AO** — Driver writes a DINT tag to TwinCAT
- **LO** — Driver writes a UDINT tag to TwinCAT
- **FO** — Driver writes a REAL tag to TwinCAT

These mappings ensure that each TwinCAT variable is handled with the correct data type and direction, allowing reliable read/write communication between pbsSoftLogic and the TwinCAT PLC. Figure 15



ctive	Tag Name	Type	Group	Group Offset	Grouplen	pbs Type
☑	GVL.D_AI1	DINT	61488	387216	4	AI
☑	GVL.D_AI2	DINT	61488	387220	4	AI
☑	GVL.D_AI3	DINT	61488	387224	4	AI
☑	GVL.D_AI4	DINT	61488	387228	4	AI
☑	GVL.D_AOB1_S	DINT	61488	387256	4	AI
☑	GVL.D_AOB1_W	DINT	61472	387240	4	AI
☑	GVL.D_AOB2_S	DINT	61488	387260	4	AI
☑	GVL.D_AOB2_W	DINT	61472	387244	4	AI
☑	GVL.D_AOB3_S	DINT	61488	387264	4	AI
☑	GVL.D_AOB3_W	DINT	61472	387248	4	AI
☑	GVL.D_AOB4_S	DINT	61488	387268	4	AI
☑	GVL.D_AOB4_W	DINT	61472	387252	4	AI
☑	GVL.D_CNT1	UDINT	61488	387288	4	LI
☑	GVL.D_DI1	BOOL	61488	387211	1	DI
☑	GVL.D_DI2	BOOL	61488	387214	1	DI
☑	GVL.D_DI3	BOOL	61488	387215	1	DI
☑	GVL.D_DI4	BOOL	61488	387232	1	DI
☑	GVL.D_DO1	BOOL	61488	387237	1	DI
☑	GVL.D_DO2	BOOL	61488	387238	1	DI
☑	GVL.D_DO3	BOOL	61488	387239	1	DI

Figure 15

After reading the TwinCAT symbols, review the list and **uncheck any tags that you do not want pbsSoftLogic to read or write**. Only the checked tags will be included in the driver configuration and used during runtime. Once you have selected the required tags, click **Save** to store the configuration and close the ADS driver GUI. The selected tags are now added to the driver's *ADSTags.xml* file and are immediately available for use inside your logic. You can reference these tags directly within any POU or Function Block in your project.

5 – Function Block programming in pbsSoftLogic

As described earlier in the pbsSoftLogic manual, the IDE fully supports Function Block programming, and all concepts such as inputs, outputs, execution order, and POU structure have already been covered in detail. In this section, we will create a simple program and demonstrate how to use the ADS tags imported from the TwinCAT IPC inside your logic.

From the **left panel**, open the **Logical** Function Block group and locate the **MAP8** Function Block. This block allows you to map **eight signals** using a single FB instance, making it ideal for grouping related inputs or outputs. Drag and drop the **MAP8** block into the **Main POU** to begin building your logic.

Drag and drop an **Input Signal** element from the left panel and connect it to the **in1** input of the **MAP8** Function Block. The *Input Signal* element is used for **reading driver tags**, while the *Output Signal* element is used for **writing values back to driver tags**. In pbsSoftLogic, all driver inputs must be connected using **Input Signal** elements placed on the left side of Function Blocks. Conversely, all outputs that write to drivers must use **Output Signal** elements connected on the right side of the Function Blocks. Each FB input can accept only **one** Input Signal element, ensuring a clear and deterministic data source. However, the right side of an FB may drive **multiple Output Signal** elements, allowing the same processed value to be written to several driver tags if required.

To assign a driver tag to the **Input Signal** element, right-click on the Input Signal and select **"DRV Signals"** from the context menu. A list of all defined drivers will appear, along with the tags available inside each driver. Select the desired tag from the list, and the IDE will automatically link it to the selected Input Signal element. Once linked, the Input Signal will read the value of that driver tag during every logic scan cycle, making it available as an input to the connected Function Block.

Repeat the previous steps to assign and connect **eight Input Signal** elements to the corresponding inputs of the **MAP8** Function Block. Each Input Signal should be linked to one ADS tag from the TwinCAT driver, allowing the MAP8 block to process all eight values during each logic scan. Once all signals are connected, you can **compile** the project and **transfer** the application to the IPC. After deployment, the logic will execute inside the IPC according to the configured scan time, and you can monitor the behavior of your MAP8 Function Block and the ADS signals in real time using the pbsSoftLogic monitoring tools.

Figure 16 , 17

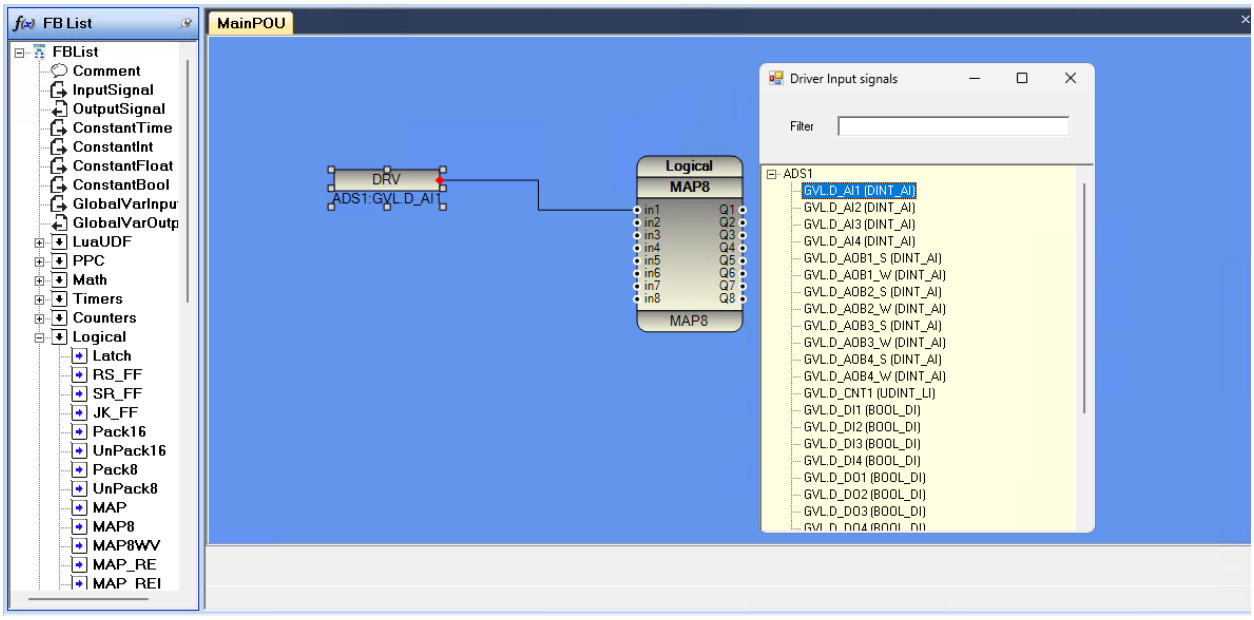


Figure 16

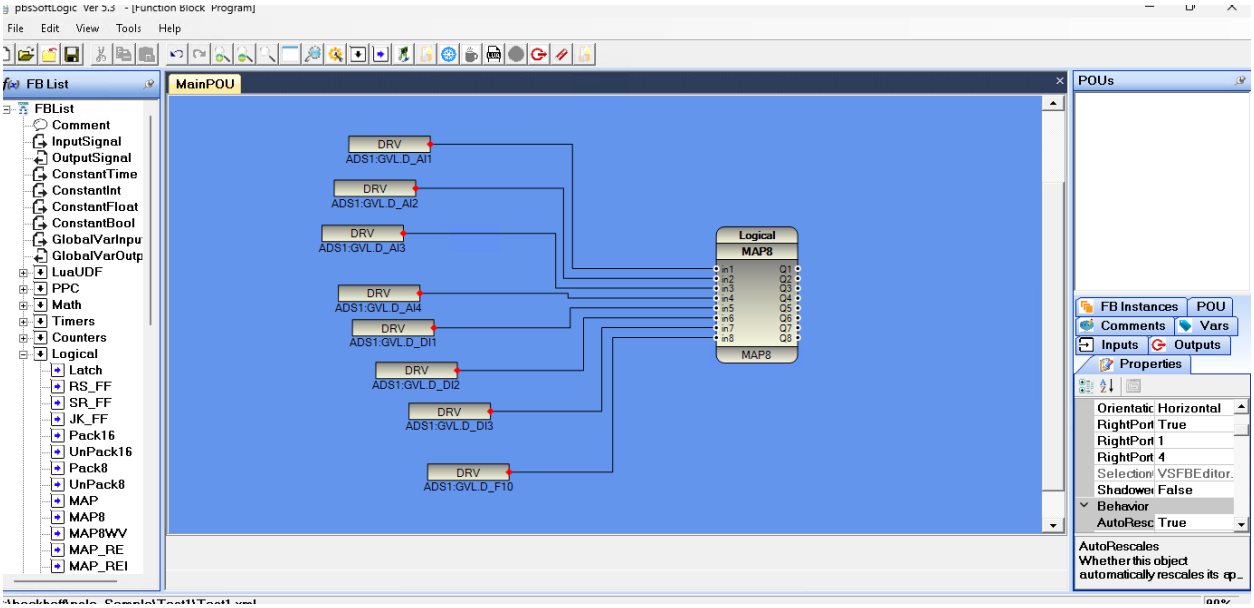


Figure 17

In the same way, drag and drop **four Output Signal** elements and connect them to the output terminals of the **MAP8** Function Block. To assign a driver tag to an Output Signal, right-click on the element and select **"DRV Signal"**. The list of available driver tags will appear, and you should select only those tags that support **write operations**, such as **DO**, **BO**, **AO**, **LO**, and **FO** types. Once selected, the Output Signal element will write the processed value from the Function Block to the corresponding TwinCAT tag during each logic scan cycle. This completes the mapping of MAP8 outputs to ADS write-enabled variables. Figure 18

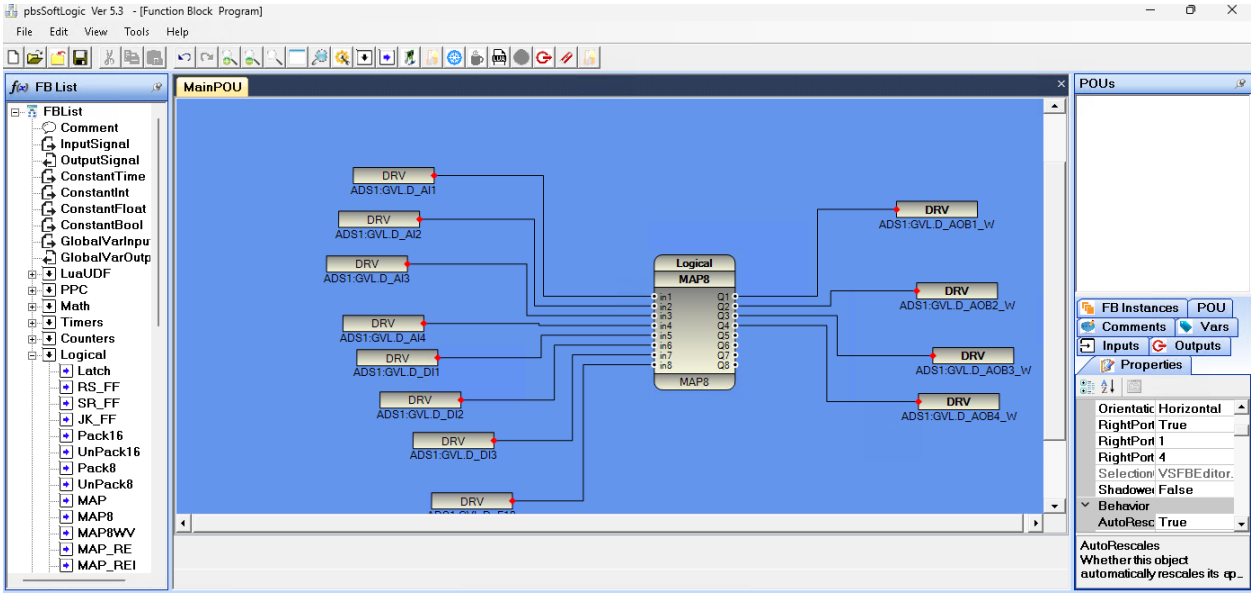


Figure 18

After completing the signal mappings and connecting all Input and Output Signal elements, save the project to ensure all configuration changes are stored. Use the **button bar** at the top of the IDE to **compile** the project. The compiler will check the Function Block logic, driver connections, and tag assignments for errors. If the compilation completes successfully, the project is ready to be transferred to the IPC for execution. Figure 19

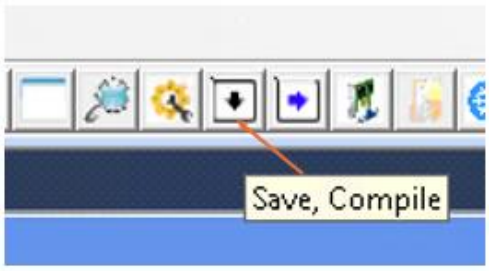


Figure 19

By clicking the **Save, Compile, and Transfer** button on the button bar, the IDE automatically performs all three steps in sequence: it saves the current project, compiles the logic, and transfers the compiled logic file to the IPC. This provides a fast and convenient workflow during development. **Note that this button transfers only the logic program**—it does **not** transfer or update any protocol configuration files such as driver settings or communication parameters. Protocol configuration must be transferred separately when changes are made. Figure 20

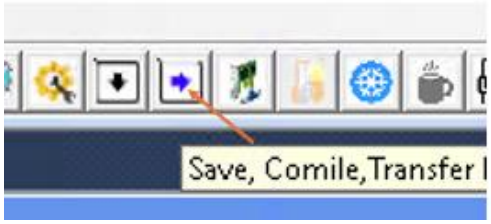


Figure 20

You can transfer the protocol configuration to the IPC by clicking the **"Transfer Configuration"** button. Logic and protocol settings are transferred independently, allowing you to update the logic without affecting the communication drivers. This separation enables the use of **warm** and **cold** logic updates without restarting the runtime kernel. However, when any modification is made to the protocol configuration—such as changes in driver parameters, communication settings, or tag definitions—the kernel must be restarted for the new protocol settings to take effect. In contrast, logic-only changes can be applied immediately using warm or cold update, ensuring fast and efficient development cycles. Figure 21

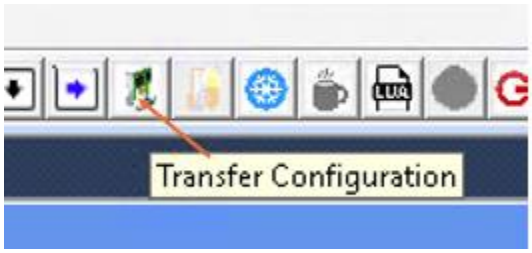


Figure 21

After transferring both the logic and the protocol configuration to the IPC and restarting the kernel, you can connect to the running application by clicking the **"Connect to Controller"** button. If the kernel is active on the IPC, the IDE automatically switches to **Monitoring Mode**. In this mode, the Main POU becomes fully observable, and you can view real-time values of all ADS tags, Function Block inputs and outputs, and internal logic signals. This allows you to verify correct communication with the TwinCAT IPC and monitor the execution of your logic directly from the IDE. Figure 22

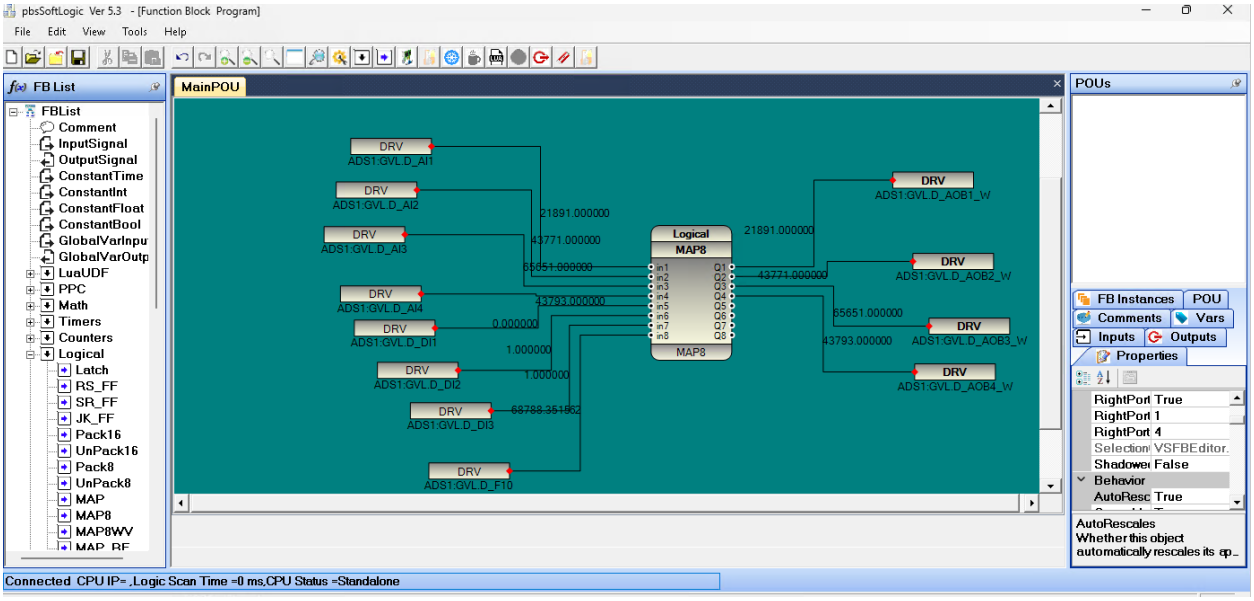


Figure 22

For configuration of other communication protocols and IDE functions, please refer to the dedicated pbsSoftLogic manuals for **DNP3**, **IEC-104**, **Modbus**, **MQTT**, **BACnet**, and all other supported drivers. The configuration workflow, logic integration, and monitoring features are identical across all supported hardware platforms. Once you understand the ADS driver setup, the same principles apply to every protocol in pbsSoftLogic, ensuring a unified and consistent development experience.

EOD