Start Up Guide for Unity Pro
Installing an Application

UNY USE 40010V20E

eng September 2004

35008402.00

a brand of Schneider Electric

Telemecanique

Table of Contents

About the Book

At a Glance

Document Scope

This manual describes how to install an application using different types of variables, programming languages and an operator screen describing the operation of the application.

Validity Note

The application presented in this manual was developed using version V2.0 of Unity Pro software.

Related Documents

Title of Documentation	Reference Number
Unity Pro Online Help	
Application available in the documentation CD	Tank_management.XEF

User Comments

We welcome your comments about this document. You can reach us by e-mail at TECHCOMM@modicon.com

Description of the application

1

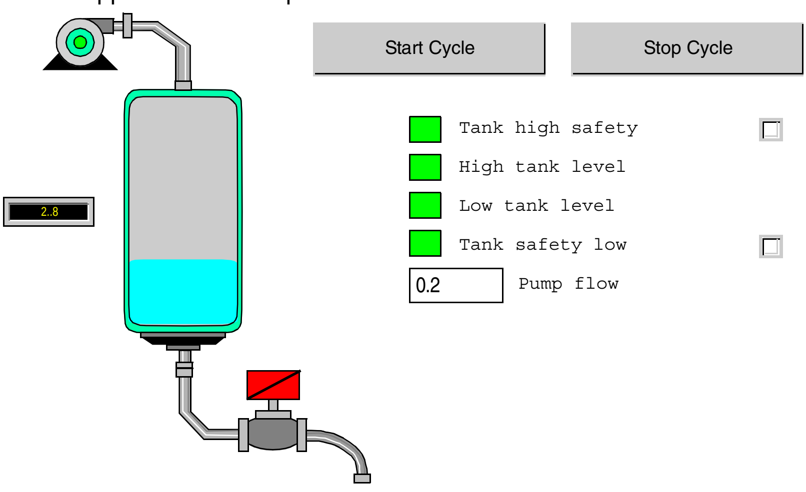
Presentation of the Application

At a Glance

The application described in this document is used to manage the level of a liquid in a tank. The tank is filled by a pump, and drained using a valve.
The different levels of the tank are measured with sensors placed on the tank.
The volume of the tank is shown by a digital display.
The application's operation control resources are based on an operator screen, which shows the status of the various sensors and actuators, as well as the volume of the tank.
Depending on the status of the tank level and the application, the user must be alerted by way of alarms, with all necessary information backed up each time these are triggered.

Illustration

This is the application's final operator screen:



Operating mode

The operating mode is as follows:

- A **Start Cycle** button is used to run filling cycles,
- When the high level of the tank is reached, the pump stops and the valve opens. When the low level of the tank is reached, the valve closes and the pump is activated until the high level is reached.
- A **Stop Cycle** button is used to interrupt the fill cycles. Pressing this button allows you to set the system to a safe level. The pump stops and the valve opens until the "Low safety" level is reached (tank empty). The valve closes and the cycle stops.
- The pump has a variable flow rate, the value of which can be accessed by the operator screen. The flow rate of the valve is equal to that of the pump.
- Safety measures must be installed:
  - Loss of tank's high level: another level, called "High safety" is activated, and the system is set to failsafe. The pump then stops and the valve opens until the "Low safety" level is reached (tank empty). The valve closes and the cycle stops.
  - Loss of tank's low level: another level, called "Low safety" is activated, and the system is set to failsafe. The valve then closes and the cycle stops.
- For both failsafe modes, a fault message must be displayed.
- The time that the valve is open and closed is monitored, with a fault message being displayed if either of these is exceeded.

Presentation of Unity Pro software

2

Presentation of Unity Pro Software

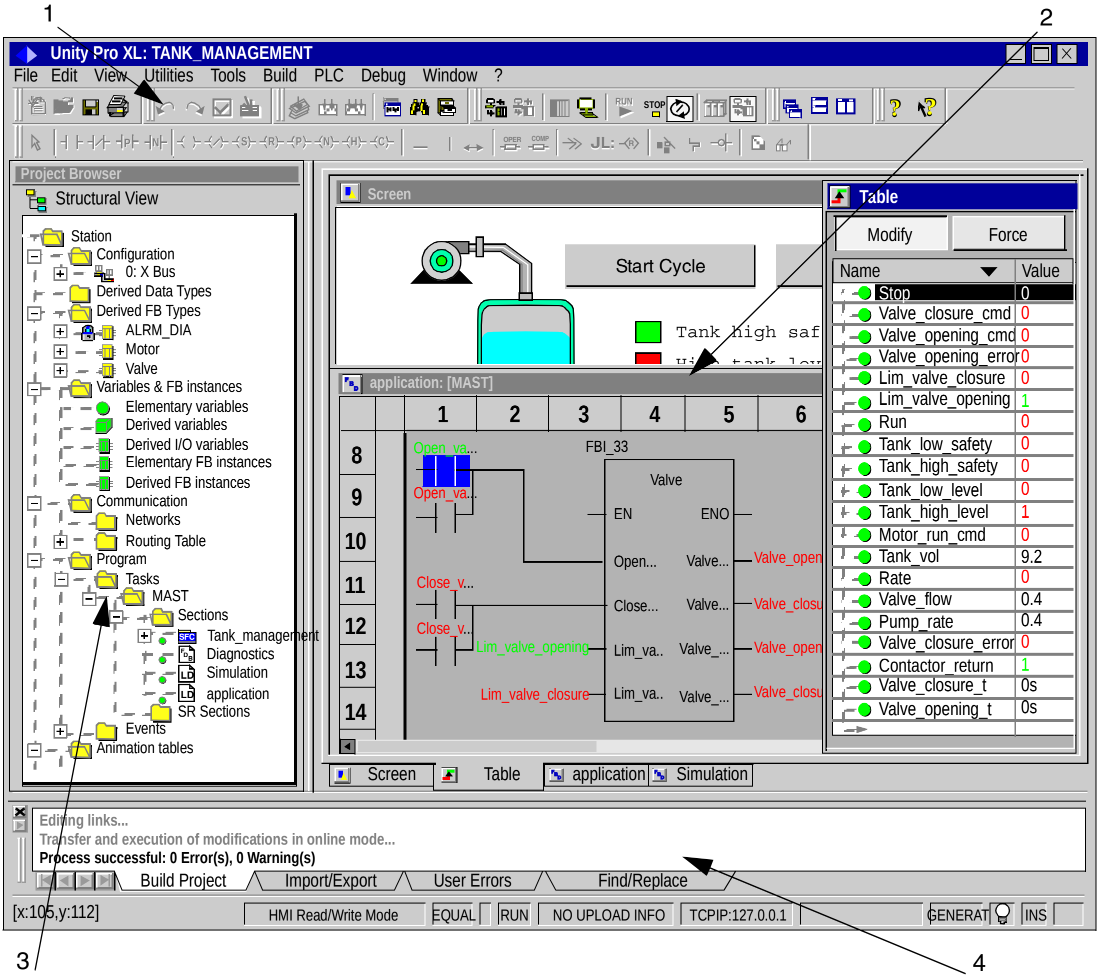
At a Glance

Unity Pro is a software workshop for programming Telemecanique Modicon Premium, Modicon Quantum and Modicon Atrium PLCs.
Below we provide a brief description of each of the blocks of Unity Pro required for application development.

Note: For more information, see Unity Pro online help.

User Interface

The screen below shows the Unity Pro user interface:



The user interface is divided into several areas:

Area	Description
1	Unity Pro toolbar .
2	Editor window (language editors, data editors, etc.).
3	Project browser.
4	Information window (provides information on errors, signal monitoring, import functions, etc.).

Project Browser

The project browser provides easy access to various editors (See *The Different Steps in the Process using Unity Pro, p. 18*) used in the application.
- Configuration (See *Configuration, p. 11*),
- Derived FB Types (See *DFB Editor, p. 13*),
- Variables & FB instances (See *Data Editor, p. 12*),
- Programs (See *Program Editor, p. 12*),
- Diagnostics (See *Diagnostics Viewer, p. 13*),
- Operator screens (See *Operator Screens, p. 14*).

Configuration

The configuration tool is used to:
- create\modify\save the elements used to configure the PLC station,
- set up the application-specific modules comprising the station,
- diagnose the modules configured in the station,
- assess the current consumed on the basis of the voltages supplied by the power supply module declared in the configuration,
- control the number of application-specific channels configured in relation to the capacities of the processor declared in the configuration,
- assess processor memory usage.

> **Note:** The configuration may be performed before or after the programming of the project; this has the advantage of being able to create generic projects without having to be concerned with the configuration in the initial stage.

> **Note:** For more information, see Unity Pro online help (click on  , then `Unity`, then `Unity Pro`, then `Operate modes`, and `Project configuration`).

Data Editor

The data editor offers the following functions:
- declaration of variable instances,
- definition of Derived Data Types (DDT), directly accessible via `Derived Data Types`,
- declaration of instances of Elementary and Derived Function Blocks (EFB/DFB),
- definition of parameters of Derived Function Blocks (DFB), directly accessible via `Derived FB Types` (See *DFB Editor, p. 13*).

To access the `Data editor`, simply double-click on `Variables & FB instances` in the project browser.

> **Note:** For more information, see Unity Pro online help (click on  , then `Unity`, then `Unity Pro`, then `Operate modes`, and `Data editor`).

Program Editor

The program editor is used to develop the different PLC tasks using different types of language, in particular:
- FBD (Function Block Diagram),
- LD (Ladder Diagram),
- SFC (Sequential Function Chart), only available for the MAST task,
- IL (Instruction List),
- ST (Structured Text).

To access the `Program editor`, simply double-click on `Program` in the project browser and select a `Task` or an `Event`.

> **Note:** For more information, see Unity Pro online help (click on  , then `Unity`, then `Unity Pro`, then `Operate modes`, and `Programming`).

DFB Editor

Unity Pro software enables you to create DFB user function blocks, using automation languages. A DFB is a program block that you develop to meet the specific requirements of your application. It includes:

- input/output parameters,
- public or private internal variables.
- one or more sections written in Ladder Diagram (LD), Instruction List (IL), Structured Text (ST) or Functional Block Diagram (FBD) language,

To access the `DFB editor`, simply double-click on `Derived FB Types` in the project browser.

> **Note:** For more information, see Unity Pro online help (click on  , then `Unity`, then `Unity Pro`, then `Language references`, and `User function block`).

Diagnostics Viewer

Unity Pro features a diagnostics tool for systems and projects.
If errors occur, they are displayed in a diagnostics window.

To access the `DFB editor`, simply double-click on `Derived FB Types` in the project browser.

> **Note:** For more information, see Unity Pro online help (click on , then `Unity`, then `Unity Pro`, then `Operate modes`, and `Diagnostics`).

Operator Screens

The operator screens are built into the software to aid operation of an automated process. In the Unity Pro software, they use:

- the project browser for browsing through the screens and launching different tools (the graphics editor, variables editor, messages editor, etc.),
- the graphics editor for creating or changing screens. In online mode, it also allows the viewing of animated screens and process driving,
- the library of objects which presents design objects and enables their insertion in the screens. It also allows users to create their own objects and insert them in a library family.

To access `Operator screens`, simply right-click on `Operator screens` in the project browser and select a new screen.

> **Note:** For more information, see Unity Pro online help (click on , then `Unity`, then `Unity Pro`, then `Operate modes`, and `Operator screens`).

Simulator

The PLC simulator enables you to simulate a project without having to connect to a real PLC.
All the project tasks (Mast, Fast, AUX and Event) are also available in the simulator. The difference in relation to a real API is that there are no I/O and communications modules.

To access the `Simulator`, simply select `Simulation mode` in the PLC menu and connect to the API.

> **Note:** For more information, see Unity Pro online help (click on , then `Unity`, then `Unity Pro`, then `Operate modes`, then `Debugging and adjustment` and `PLC simulator`).

3 Installing the Application using Unity Pro

At a Glance

Subject of this Chapter

This chapter describes the procedure for creating the application described. It shows, in general and in more detail, the steps in creating the different components of the application.

What's in this Chapter?

This chapter contains the following sections:

3.1 Presentation of the Solution Used

At a Glance

Subject of this Section

This section presents the solution used to develop the application. It explains the technological choices and gives the application’s creation timeline.

What's in this Section?

This section contains the following topics:

Technological Choices Used

At a Glance

There are several ways of writing an application using Unity Pro. The one proposed allows you to structure the application so as to facilitate its creation and debugging.

Technological Choices

The following table shows the technological choices used for the application:

Objects	Choices used
Use of the pump	Creation of a user function block (DFB) to facilitate management of the pump in terms of entering a program and speed of debugging. The programming language used to develop this DFB is a function block diagram (FBD)-based graphic language.
Use of the valve	Creation of a user function block (DFB) to facilitate management of the valve in terms of entering a program and speed of debugging. The programming language used to develop this DFB is a function block diagram (FBD)-based graphic language.
Supervision screen	Use of elements from the library and new objects.
Main supervision program	This program is developed using a sequential function chart (SFC), also called GRAFCET. The various sections are created in Ladder Diagram (LD) language, and use the different DFBs created.
Fault display	Use of the ALRM_DIA DFB to control the status of the variables linked with the faults.

Note: Using a DFB function block in an application enables you to:
- simplify the design and entry of the program,
- increase the legibility of the program,
- facilitate debugging the application,
- reduce the volume of generated code.

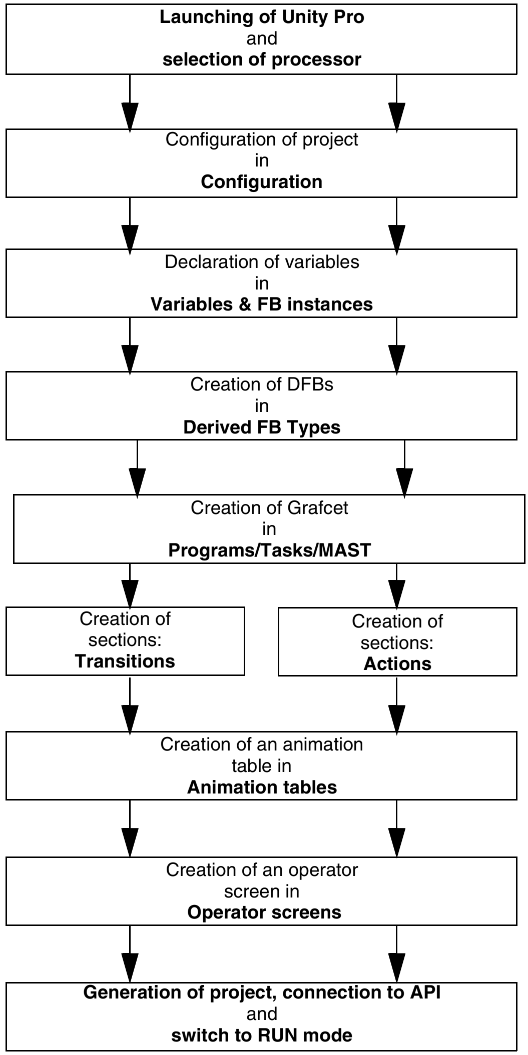
The Different Steps in the Process using Unity Pro

At a Glance

The following logic diagram shows the different steps to follow to create the application. A chronological order must be respected in order to correctly define all of the application elements.

Description

Description of the different types:



3.2 Developing the Application

At a Glance

Subject of this Section

This section gives a step-by-step description of how to create the application using Unity Pro.

What's in this Section?

This section contains the following topics:

Creating the Project

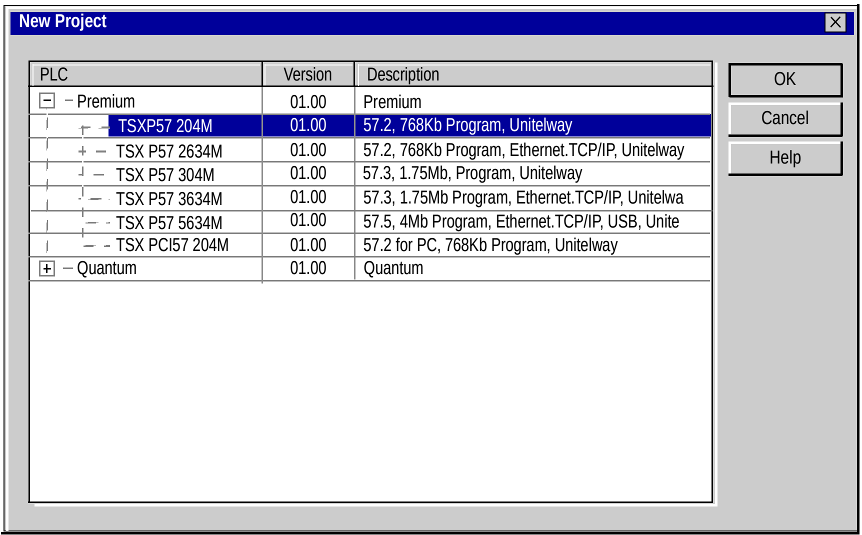
At a Glance

Developing an application using Unity Pro involves creating a project associated with a PLC.

> **Note:** For more information, see Unity Pro online help (click on , then `Unity`, then `Unity Pro`, then `Operate modes`, and `Project configuration`).

Procedure for Creating a Project

The table below shows the procedure for creating the project using Unity Pro.

Step	Action
1	Launch the Unity Pro software,
2	Click on `File` then `New` then select a PLC, 
3	Insert a module (See *Application Hardware Configuration, p. 49*) or network to terminate your configuration.
4	Confirm with `OK`. You can now develop your application in Unity Pro.

Declaration of variables

At a Glance

All of the variables used in the different sections of the program must be declared. Undeclared variables cannot be used in the program.

Note: For more information, see Unity Pro online help (click on  , then `Unity`, then `Unity Pro`, then `Operate modes`, and `Data editor`).

Procedure for Declaring Variables

The table below shows the procedure for declaring application variables:

Step	Action
1	In `Project browser` / `Variables & FB instances`, double-click on `Elementary variables`.
2	In the `Data editor` window, select the box in the `Name` column and enter a name for your first variable.
3	Now select a Type for this variable.
4	When all your variables are declared, you can close the window.

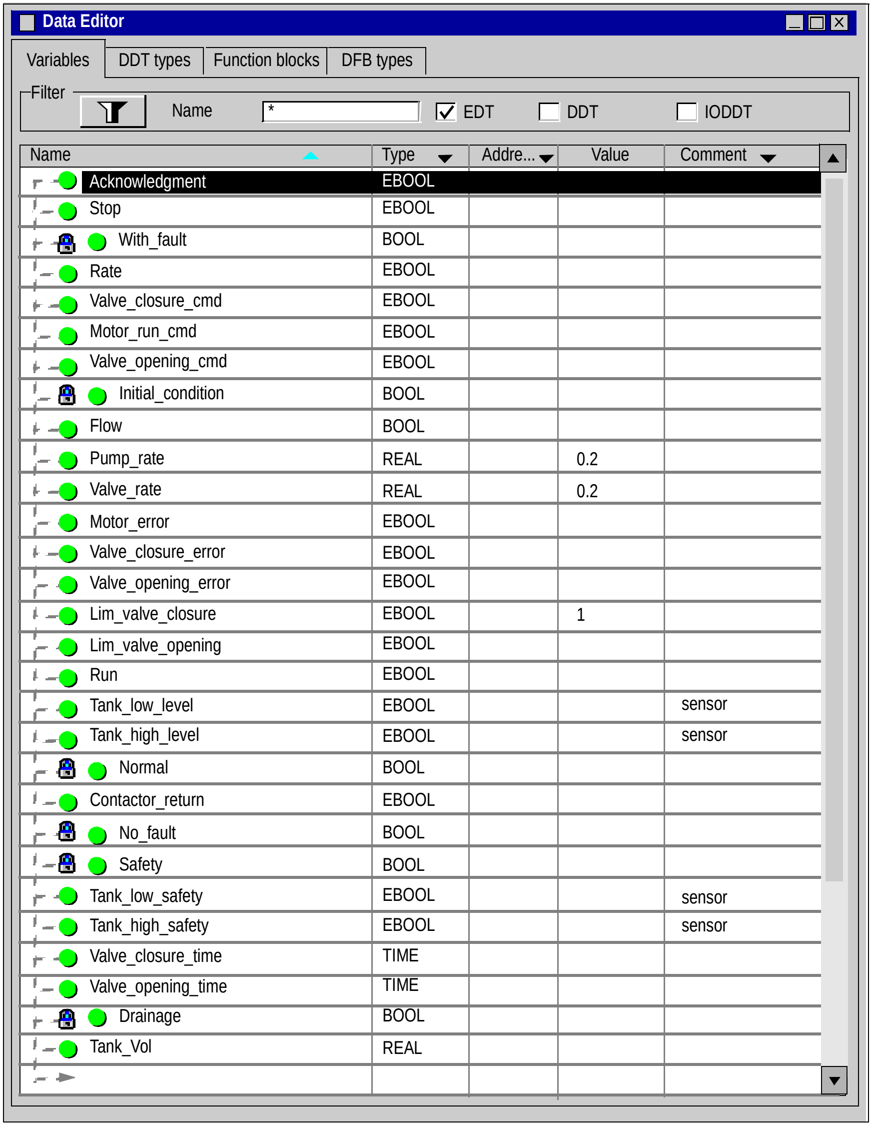
Variables Used for the Application

The following table shows the details of the variables used in the application:

Variable	Type	Definition
Acknowledgement	EBOOL	Acknowledgement of a fault (Status 1).
Stop	EBOOL	Stop cycle at end of draining (Status 1).
Run	EBOOL	Startup request for filling cycles (Status 1).
Motor_run_cmd	EBOOL	Startup request for filling cycles (Status 1).
Motor_error	EBOOL	Error returned by the motor.
Contactor_return	EBOOL	Error returned by the contactor in the event of motor error.
Pump_rate	REAL	Pump flow rate value.
Flow rate	BOOL	Intermediate variable for simulating the application.
Rate	EBOOL	Variable used to calculate the volume of the tank (same as %S6 in our project). This variable is used to simulate the project, and must be deleted for real-life cases.
Valve_opening_cmd	EBOOL	Opening of the valve (Status 1).
Valve_closure_cmd	EBOOL	Closing of the valve (Status 1).
Valve_opening_error	EBOOL	Error returned by the valve on opening.
Valve_closure_error	EBOOL	Error returned by the valve on closing.
Lim_valve_opening	EBOOL	Valve in open position (Status 1).
Lim_valve_closure	EBOOL	Valve in closed position (Status 1).
Valve_closure_time	TIME	Valve closure time.
Valve_opening_time	TIME	Valve opening time.
Tank_low_level	EBOOL	Tank volume at low level (Status 1).
Tank_high_level	EBOOL	Tank volume at high level (Status 1).
Tank_low_safety	EBOOL	Tank volume at low safety level (Status 1).
Tank_high_safety	EBOOL	Tank volume at high safety level (Status 1).
Tank_vol	REAL	Variable used to calculate the volume of the tank. This variable is used to simulate the project, and must be deleted for real-life cases.

Note: EBOOL types can be used for I/O modules, unlike BOOL types.

The following screen shows the application variables created using the data editor:



Name	Type	Addre...	Value	Comment
Acknowledgment	EBOOL			
Stop	EBOOL			
With_fault	BOOL			
Rate	EBOOL			
Valve_closure_cmd	EBOOL			
Motor_run_cmd	EBOOL			
Valve_opening_cmd	EBOOL			
Initial_condition	BOOL			
Flow	BOOL			
Pump_rate	REAL		0.2	
Valve_rate	REAL		0.2	
Motor_error	EBOOL			
Valve_closure_error	EBOOL			
Valve_opening_error	EBOOL			
Lim_valve_closure	EBOOL		1	
Lim_valve_opening	EBOOL			
Run	EBOOL			
Tank_low_level	EBOOL			sensor
Tank_high_level	EBOOL			sensor
Normal	BOOL			
Contactor_return	EBOOL			
No_fault	BOOL			
Safety	BOOL			
Tank_low_safety	EBOOL			sensor
Tank_high_safety	EBOOL			sensor
Valve_closure_time	TIME			
Valve_opening_time	TIME			
Drainage	BOOL			
Tank_Vol	REAL			

Creation and Use of DFBs

At a Glance

DFB types are function blocks that can be programmed by the user ST, IL, LD or FBD. Our application uses a motor DFB and a valve DFB.
We will also be using existing DFB from the library for monitoring variables. Particularly "safety" variables for tank levels, and "error" variables returned by the valve. The status of these variables will be visible in `Diagnostics display`.

> **Note:** Function blocks can be used to structure and optimize your application. They can be used whenever a program sequence is repeated several times in your application, or to set a standard programming operation (for example, an algorithm that controls a motor).
> Once the DFB type is created, you can define an instance of this DFB via the variable editor or when the function is called in the program editor.

> **Note:** For more information, see Unity Pro online help (click on  , then `Unity`, then `Unity Pro`, then `Language references`, and `User function block`).

Procedure for Creating a DFB

The table below shows the procedure for creating application DFBs.

Step	Action
1	In the `Project browser`, right click on `Derived FB types` and select `Open`.
2	In the `Data editor` window, select the box in the `Name` column and enter a name for your DFB and confirm with `Enter`. The name of your DFB appears with the sign "Works" (unanalyzed DFB).
3	Open the structure of your DFB (see figure below) and add the inputs, outputs and other variables specific to your DFB.
4	When the variables of the DFB are declared, analyze your DFB (the sign "Works" must disappear). To analyze your DFB, select the DFB and, in the menu, click `Build` then `Analyze`. You have created the variables for your DFB, and must now create the associated section.
5	In the `Project browser`, double-click on `Derived FB types` then on your DFB. Under the name of your DFB, the `Sections` field will appear.
6	Right click on `Sections` then select `New section`.
7	Give your section a name, then select the language type and confirm with `OK`. Edit your section using the variables declared in step 3. Your DFB can now be used by the program (DFB Instance).

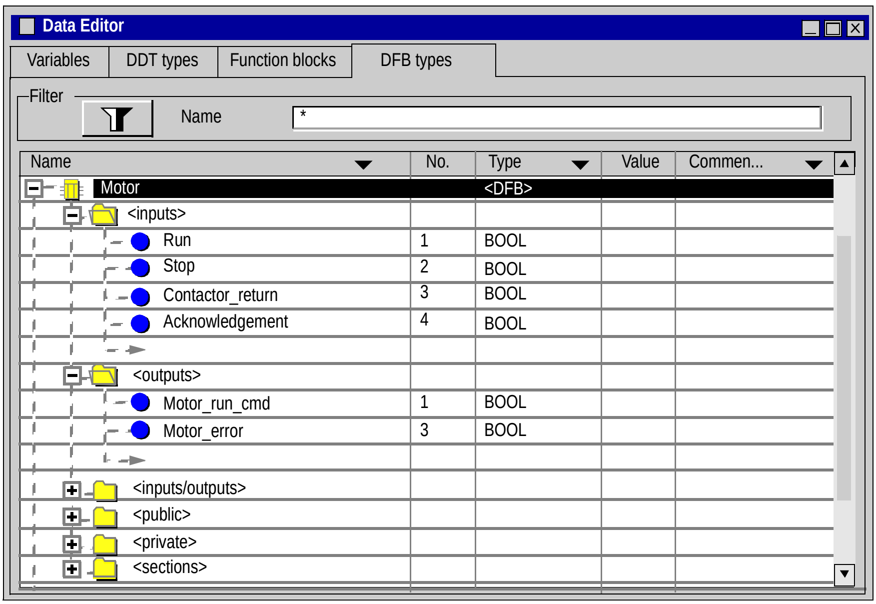
Variables Used by the Motor DFB

The following table lists the variables used by the Motor DFB:

Variable	Type	Definition
Run	Input	Motor run command.
Stop	Input	Motor stop command.
Contactor_return	Input	Contactor feedback in the event of motor run problem.
Acknowledgement	Input	Acknowledgement of the Motor_error output variable.
Motor_run_cmd	Output	Start of motor.
Motor_error	Output	Display in the "Diagnostics display" window of an alarm linked to a problem with the motor.

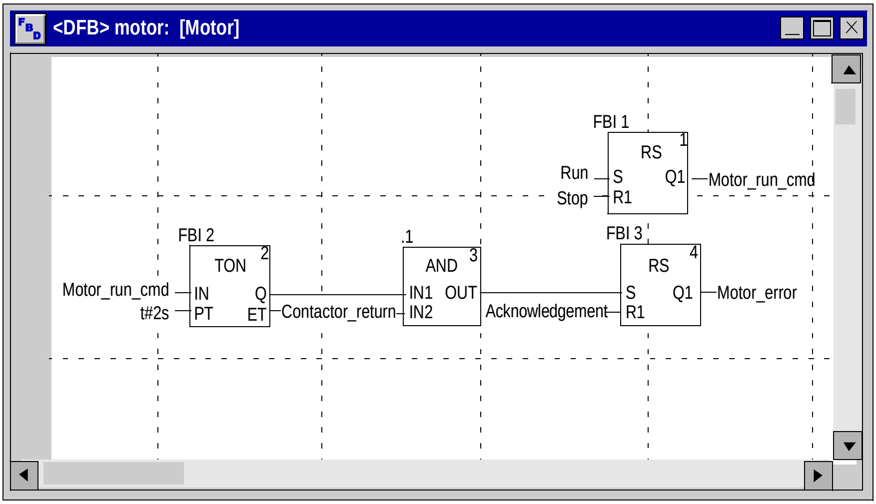
Illustration of the Motor DFB variables declared in the data editor

The following screen shows the Motor DFB variables used in this application to control the motor:



Operating Principle of the Motor DFB

The following screen shows the Motor DFB program written by the application in FBD for controlling the motor:



When Run = 1 and Stop = 0, the motor can be controlled (Motor_run_cmd = 1). The other part monitors the Contactor_return variable. If Contactor_return is not set to "1" after the Discrete counter counts two seconds, the Motor_error output switches to "1".

Note: For more information on creating a section, consult the Unity Pro online help (click  , then `Unity`, then `Unity Pro`, then `Operate Modes` and `Programming` and select the required language).

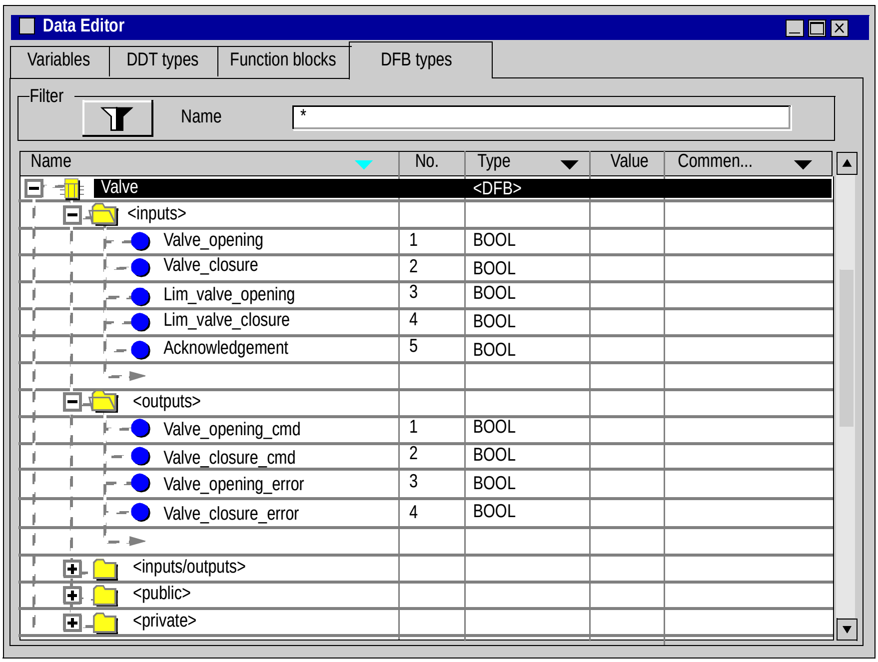
Variables Used by the Valve DFB

The following table lists the variables used by the Valve DFB:

Variable	Type	Definition
Valve_opening	Input	Valve opening command.
Valve_closure	Input	Valve closure command.
Lim_valve_opening	Input	Status of valve limit.
Lim_valve_closure	Input	Status of valve limit.
Acknowledgement	Input	Acknowledgement of variables Valve_closure_error or Valve_opening_error.
Valve_opening_cmd	Output	Opening of the valve.
Valve_closure_cmd	Output	Closure of the valve.
Valve_opening_error	Output	Display in the "Diagnostics display" window of an alarm linked to a problem with the valve opening.
Valve_closure_error	Output	Display in the "Diagnostics display" window of an alarm linked to a problem with the valve closure.

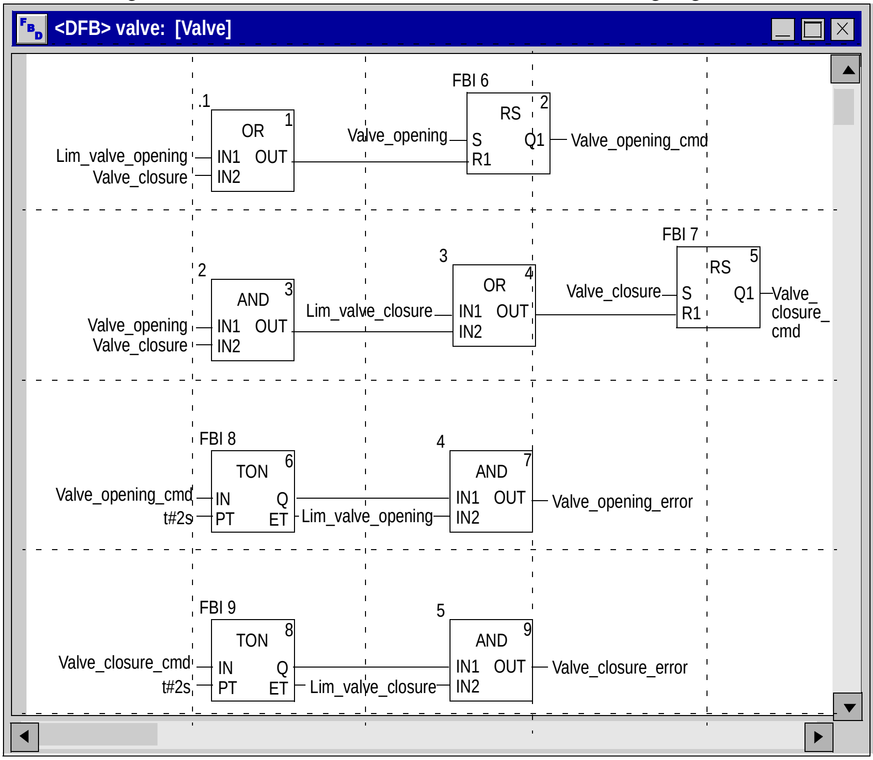
Illustration of the Valve DFB variables declared in the data editor

The following screen shows the Valve DFB variables used in this application to control the valve:



Operating Principle of the Valve DFB

The following screen shows the Valve DFB written in FBD language:



This DFB authorizes the command to open the valve (Valve_opening_cmd) when the inputs Valve_closure and Lim_valve_opening are set to "0". The principle is the same for closure, with an additional safety feature if the user requests the opening and closing of the valve at the same time (opening takes priority).
In order to monitor opening and closing times, we use the TON timer to delay the triggering of a fault. Once the valve opening is enabled (Valve_opening_cmd = 1), the timer is triggered. If Lim_valve_opening does not switch to "1" within two seconds, the output variable Valve_opening_error switches to "1". In this case a message is displayed (See *Diagnostics Viewer, p. 51*).

Note: The PT time must be adjusted according to your equipment

Note: For more information on creating a section, consult the Unity Pro online help (click , then `Unity`, then `Unity Pro`, then `Operate Modes` and `Programming` and select the required language).

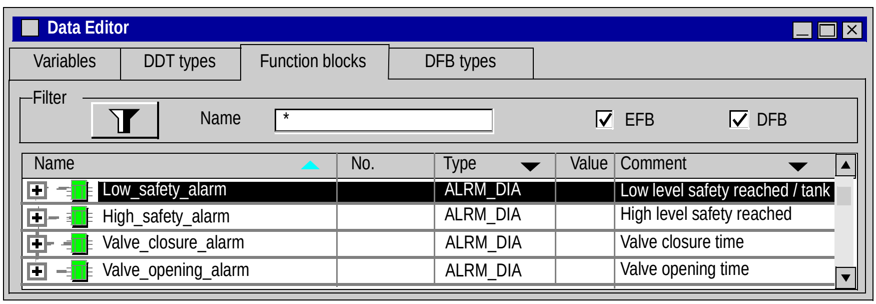
Procedure for Customizing an Existing DFB from a Library DFB

The table below shows the procedure for using library ALRM_DIA DFBs.

Step	Action
1	In the `Project browser`, double-click on `Elementary variables`, then select the `Function Blocks` tab.
2	In the `Data editor` window, select the cell in the `Name` column and enter a name for your Function block and confirm with `Enter`.
3	The FB type selection window appears, in `Libraries/Families` select `Libraries` then `Diagnostics` and click on `ALRM_DIA` then confirm with `Enter`.
4	In the `Data editor` window, add comments in the `Comment` field in order to display them in `Diagnostics viewer`. Your Function block can now be used by the program (DFB Instance).

Illustration of the Function Blocks Used by the Application

The following screen shows the different ALRM_DIA Function blocks used in the application for displaying information in the `Diagnostics viewer` window:



Creating the Program in SFC for Managing the Tank

At a Glance

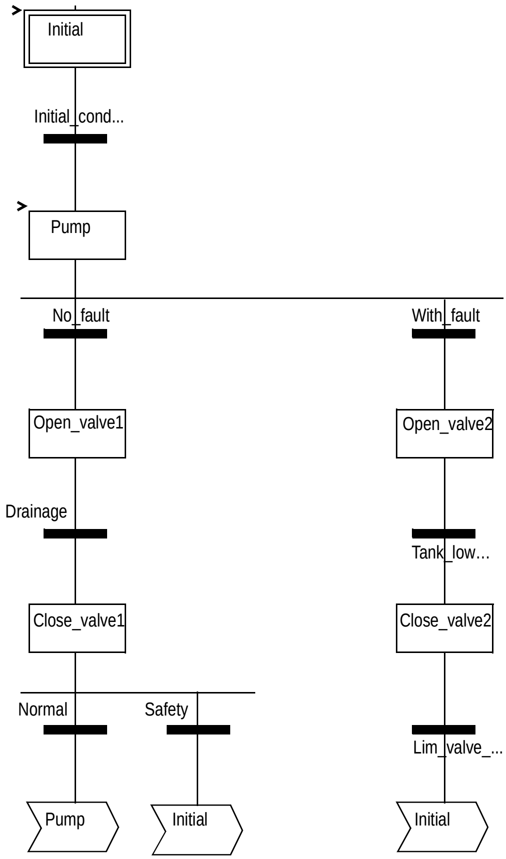
The main program is written in SFC (Grafcet). The different sections of the grafcet steps and transitions are written in LD. This program is declared in a MAST task, and will depend on the status of a Boolean variable.
The main advantage of SFC language is that its graphic animation allows us to monitor in real time the execution of an application.
Several sections are declared in the MAST task:

- The **Tank_management** (See *Illustration of the Tank_management Section, p. 31*) section, written in SFC and describing the operate mode,
- The **Application** (See *Creating a Program in LD for Application Execution, p. 34*) section, written in LD, which executes the pump start-up using the motor DFB, as well as the opening and closure of the valve.
- The **Simulation** (See *Creating a Program in LD for Application Simulation, p. 36*) section, written in LD, which simulates the application. This section must be deleted in the case of connection to a PLC.
- The **Diagnostics** (See *Creating a Program in FBD for Application Diagnostics, p. 39*) section, written in FBD, for returning application errors to the diagnostics display.

Note: The LD, SFC and FBD-type sections used in the application must be animated in online mode (See *Starting the Application, p. 47*), with the PLC in RUN.

Illustration of the Tank_management Section

The following screen shows the application Grafcet:



Note: For more information on creating an SFC section, see Unity Pro online help (click on  , then `Unity`, then `Unity Pro`, then `Operate modes`, then `Programming` and `SFC editor`).

Description of the Tank_management Section

The following table describes the different steps and transitions of the Tank_management Grafcet:

Step / Transition	Description
Initial	This is the initial step.
Initial_condition	This is the transition that starts the pump. The transition is valid when the variables: ● Stop_cycle = 0, ● Run_cycle = 1, ● Tank_high_safety = 0, ● Lim_valve_closure = 1
Pump	This is the step that starts the pump and filling of the tank until the high level is reached. This step activates the motor DFB in the Application section, which controls the activation of the pump.
No_fault	This transition is active when the tank's high level is reached and the safety high level is set to 0.
Open_valve1	This step opens the valve to drain the tank. This step activates the valve DFB in the Application section, which controls the opening of the valve.
Drainage	This transition is active when the tank's low level or safety low level is set to 1.
Close_valve1	This is the valve closure step. This step activates the valve DFB in the Application section, which controls the closure of the valve.
Normal	This transition is valid when the low level of the tank and Lim_valve_closure are set to 1. In this case we skip to step S_1_2.
Safety	This transition is valid when the low safety level of the tank and Lim_valve_closure are set to 1. Where this is the case, we return to the start of the cycle and wait for the safety variable to be reset, and the cycle to be restarted.
With_fault	This transition is active when the High safety level of the tank has been reached, or the Stop_cycle button has been activated (Stop_cycle = 1).
Open_valve2	This step is identical to Open_valve1.
Tank_low_safety	This transition is active when the low safety level of the tank is set to 1 (after the tank is drained following a stop cycle command, or following activation of the high safety level).
Close_valve2	This step is identical to Close_valve1.
Lim_valve_closure	This transition is valid when Lim_valve_closure is set to 1. Where this is the case, we return to the start of the cycle and wait for the safety variable to be reset, and the cycle to be restarted.

> **Note:** You can see all the steps and actions of your SFC by clicking on ⊞ in front of the name of your SFC section.

Procedure for Creating an SFC Section

The table below shows the procedure for creating an SFC section for the application.

Step	Action
1	In `Project Browser\Program\Tasks`, double-click on `MAST`.
2	Right click on `Section` then select `New section`. Give your section a name (Tank_management for the SFC section) then select SFC language.
3	The name of your section appears, and can now be edited by double clicking on it.
4	The SFC edit tools appear in the window, which you can use to create your Grafcet. For example, to create a step with a transition: ● To create the step, click on  then place it in the editor, ● To create the transition, click on  then place it in the editor (generally under the preceding step).

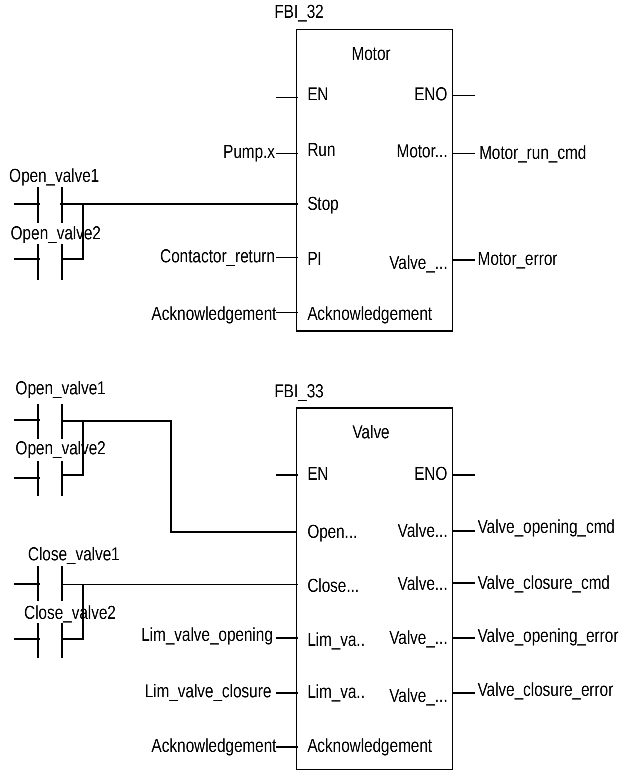
Creating a Program in LD for Application Execution

At a Glance

This section controls the pump and the valve using the DFBs created (See *Creation and Use of DFBs, p. 24*) earlier.

Illustration of the Application Section

The section below is part of the MAST task. It has no condition defined for it so it is permanently executed:



Description of the Application Section

- When the Pump step is active, the Run input of the motor DFB is at 1. If the Stop input of the motor DFB is at 0, the Motor_run_cmd switches to "1" and the pump supply is activated.
- the same principle applies to the steps Open_valve1 and Open_valve2 and to the rest of the section.

Procedure for Creating an LD Section

The table below describes the procedure for creating part of the **Application** section.

Step	Action
1	In `Project Browser\Program\Tasks`, double-click on `MAST`.
2	Right click on `Section` then select `New section`. Name this section Application, then select the language type LD. The edit window opens.
3	To create the contact Open_valve1.x, click on then place it in the editor. Double-click on this contact then enter the name of the step with the suffix ".x" at the end (signifying a step of an SFC section) and confirm with `OK`.
4	To use the motor DFB you must instantiate it. Right click in the editor then click on `Select data` and on ... . Click on the `Function and Function Block Types` tab and select your DFB then confirm with `OK` and position your DFB. To link the Open_valve1.x contact to the stop input of the DFB, align the contact and the input horizontally, click on and position the link between the contact and the input.

Note: For more information on creating an LD section, see Unity Pro online help (click on , then `Unity`, then `Unity Pro`, then `Operate modes`, then `Programming` and `LD editor`).

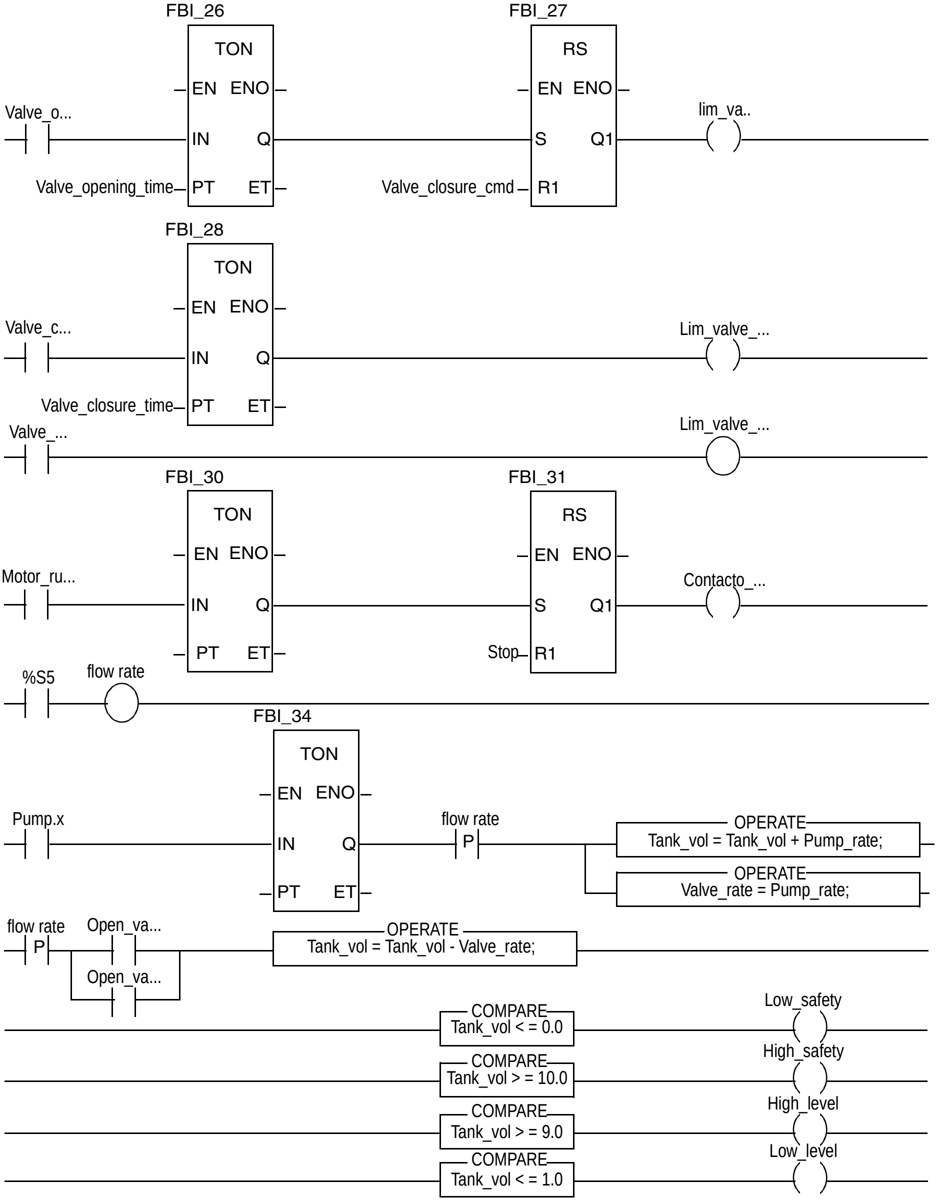
Creating a Program in LD for Application Simulation

At a Glance

This section is only used for application simulation. It should therefore not be used if a PLC is connected.

Illustration of the Simulation Section

The section below is part of the MAST task. It has no condition defined for it so it is permanently executed:



> **Note:** For more information on creating an LD section, see Unity Pro online help (click on  , then `Unity`, then Software`Unity Pro`, then `Operate modes` , then `Programming` and `LD editor`).

Description of the Simulation Section

- the first line is used to simulate the value of the Lim_valve_opening variable. If the valve opening command is given (Valve_opening_cmd = 1), a TON timer is triggered. When the PT time is reached, the TON output switches to "1" and increments the Lim_valve_opening output to "1" unless the valve closure command is given at the same time,
- same principle applies to the Lim_valve_closure and Contactor_return outputs.
- the last part of the section is used for the simulation of the tank level and for triggering the different tank levels. The OPERATE and COMPARE blocks from the library can be used to do this.

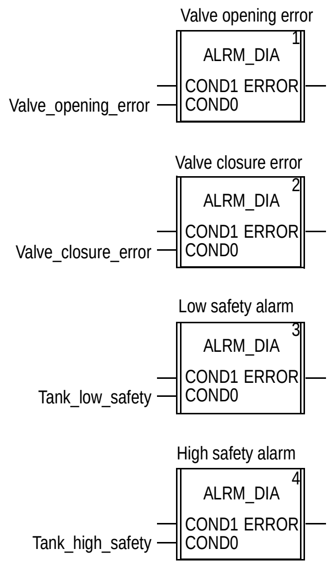
Creating a Program in FBD for Application Diagnostics

At a Glance

This section is used to declare variables which will be sent to the diagnostics viewer in the event of an error.

Illustration of the Diagnostics Section

The screen below shows the FBD section using the Function blocks (See *Illustration of the Function Blocks Used by the Application, p. 29*) Low_safety_alarm, High_safety_alarm and valve_error:



Description of the Diagnostics Section

The principle of this section is based on the use of ALMR_DIA function blocks. All the blocks monitor changes in the state of the input variable. As the inputs are always connected to COND0, display in the Diagnostics Viewer window will be triggered when the input variable switches to 1.

Procedure for Creating an FBD Section

The table below describes the principle for the **Diagnostics** section:

Step	Action
1	In `Project Browser\Program\Tasks`, double-click on `MAST`.
2	Right click on `Section` then select `New section`. Name this section Diagnostics, then select the language type FBD. The edit window opens.
3	To use the ALRM_DIA function block you created, you must instantiate it. Right click in the editor then click on `Select data` and on ... . Click on the `Function Blocks` tab and select your function block then confirm with `OK` and position it in the FBD editor. To assign a variable to an input or an output, double-click on it, click on ... and select your variable from the `Variable` tab.

Note: For more information on creating an LD section, see Unity Pro online help (click on  , then `Unity`, then `Unity Pro`, then `Operate modes`, then `Programming` and `FBD editor`).

Creating an Animation Table

At a Glance

An animation table is used to monitor the values of variables, and modify and/or force these values. Only those variables declared in `Variables & FB instances` can be added to the animation table.

Note: For more information, consult the Unity Pro online help (click  , then `Unity`, then `Unity Pro`, then `Operate modes`, then `Debugging and adjustment` then `Viewing and adjusting variables` and `Animation tables`).

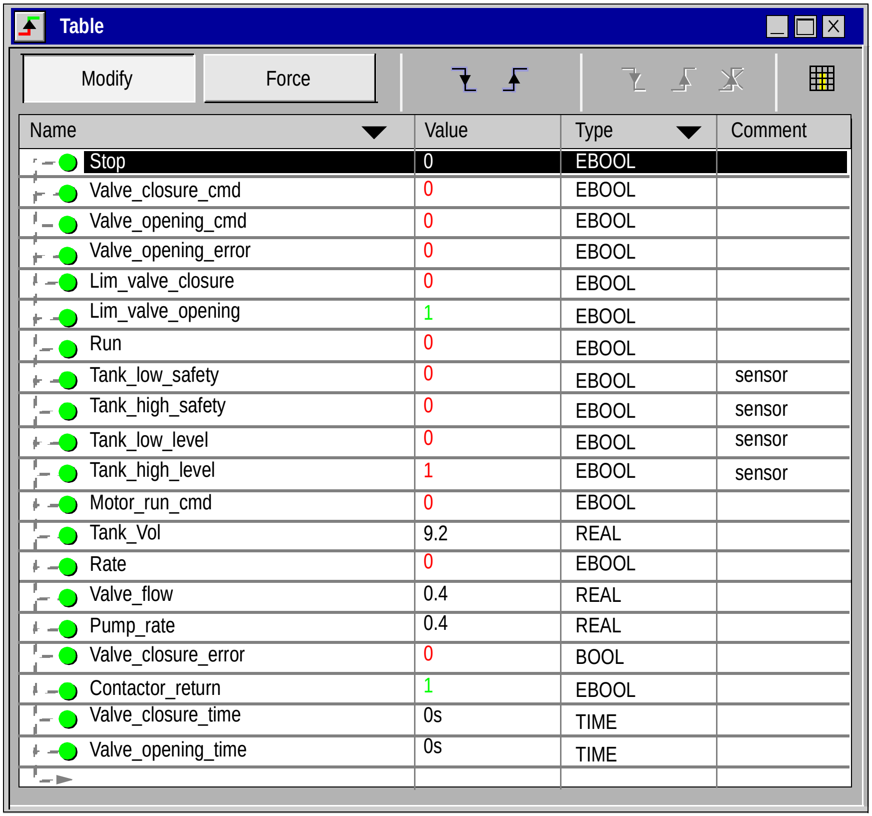
Procedure for Creating an Animation Table

The table below shows the procedure for creating an animation table.

Step	Action
1	In the `Project browser`, right click on `Animation tables`. The edit window opens.
2	Click on first cell in the Name column, then on the button, and add the variables you require.

Animation Table Created for the Application

The following screen shows the animation table used by the application:



Name	Value	Type	Comment
Stop	0	EBOOL	
Valve_closure_cmd	0	EBOOL	
Valve_opening_cmd	0	EBOOL	
Valve_opening_error	0	EBOOL	
Lim_valve_closure	0	EBOOL	
Lim_valve_opening	1	EBOOL	
Run	0	EBOOL	
Tank_low_safety	0	EBOOL	sensor
Tank_high_safety	0	EBOOL	sensor
Tank_low_level	0	EBOOL	sensor
Tank_high_level	1	EBOOL	sensor
Motor_run_cmd	0	EBOOL	
Tank_Vol	9.2	REAL	
Rate	0	EBOOL	
Valve_flow	0.4	REAL	
Pump_rate	0.4	REAL	
Valve_closure_error	0	BOOL	
Contactor_return	1	EBOOL	
Valve_closure_time	0s	TIME	
Valve_opening_time	0s	TIME	

Note: The animation table is dynamic only in online mode (display of variable values).

Creating the Operator Screen

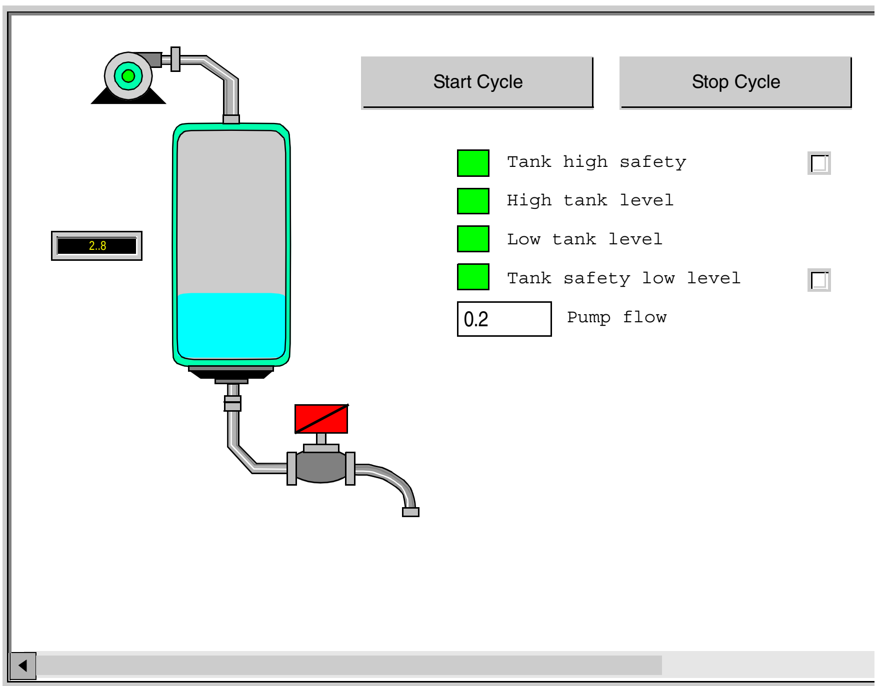
At a Glance

The operator screen is used to animate graphic objects that symbolize the application. These objects can belong to the Unity Pro library, or can be created using the graphic editor.

Note: For more information, see Unity Pro online help (click on , then `Unity`, then `Unity Pro`, then `Operate modes`, and `Operator screens`).

Illustration on an Operator Screen

The following illustration shows the application operator screen:



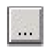
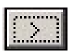
Note: To animate objects in online mode, you must click on . By clicking on this button, you can validate what is written.

Procedure for Creating an Operator Screen

The table below shows the procedure for creating the `Start Cycle` button:

Step	Action
1	In the `Project browser`, right click on `Operator screens` and click on `New screen`. The operator screen editor appears.
2	● Click on  and place it in the operator screen editor. Double-click on the button and, in the `Control` tab, select the `Run` variable by clicking on , and confirm with `OK`, then enter the name of the button in the `Text` area. The button is presently assigned to the `Run` variable.

The table below shows the procedure for inserting and animating the tank.

Step	Action
1	In the `Project browser`, right click on `Operator screens` and click on `New screen`. The operator screen editor appears.
2	● In the `Tools` menu, select `Operator Screen Library`. The window opens. Double click on `Fluids` then `Tank`. Select the dynamic tank from the runtime screen, and `Copy` (Ctrl + C) then `Paste` (Ctrl + V) it into the drawing in the operator screen editor (to return to your screen, click on `Window` then `Screen`). ● The tank is now in your operator screen. You now need a variable to animate the level. In the `Tools` menu, click on `Variables Window`. The window appears to the left, and in the `Name` column we see the word %MW0. To obtain the animated part of the graphic object (in this case the tank), double click on %MW0. A part of the tank is selected. Right click on this part, then click on `Characteristics`. Select the `Animation` tab and enter the variable concerned by clicking the  button (in the place of %MW0). In our application, this will be Tank_vol. ● You must define the tank's minimum and maximum values. In the `Type of animation` tab, click `Bar chart` then the  button, and fill in the entry fields according to the tank. ● Confirm with `Apply` and `OK`.

The table below shows the procedure for inserting and animating the valve.

Step	Action
1	In the `Project browser`, right click on `Operator screens` and click on `New screen`. The operator screen editor appears.
2	• In the `Tools` menu, select `Operator Screen Library`. The window opens. Double click on `Actuators` then `Valve`. Select a dynamic valve (from the runtime screen), and `Copy` (Ctrl + C) then `Paste` (Ctrl + V) it into the drawing in the operator screen editor (to return to your screen, click on `Window` then `Screen`). • Select the valve, right click on it then click on Detach. Select the red rectangle and move it so you can see the green rectangle underneath it. Double click on the green rectangle, then click on the Animation tab and add the Valve_opening_cmd variable. Still in the `Object properties` window, in the `Display condition` area, select `Bit = 1`. This setting makes the green rectangle visible when %M2 = 1, otherwise this rectangle is invisible. • Same procedure for the red rectangle, only with the display condition `Bit = 0`. If the animation does not work, put the foreground rectangle into the background.

Starting the Application

4

At a Glance

Subject of this Section

This chapter shows the procedure for starting the application. It describes the different types of application executions.

What's in this Chapter?

This chapter contains the following topics:

Execution of Application in Simulation Mode

At a Glance

You can connect to the API simulator which enables you to test an application without a physical connection to the PLC and other devices.

Note: For more information, see Unity Pro online help (click on  , then `Unity`, then `Unity Pro`, then `Operate modes`, then `Debugging and adjustment` and `PLC simulator`).

Application Execution

The table below shows the procedure for launching the application in simulation mode:

Step	Action
1	In the `PLC` menu, click on `Simulation Mode`,
2	In the `Build` menu, click on `Rebuild All Project`. Your project is generated and is ready to be transferred to the simulator. When you generate the project, you will see a results window. If there is an error in the program, Unity Pro indicates its location if you double-click on the highlighted sequence.
3	In the `PLC` menu, click on `Connection`. You are now connected to the simulator.
4	In the `PLC` menu, click on `Transfer project to PLC`. The `Transfer project to PLC` window opens. Click on `Transfer`. The application is transferred to the PLC simulator.
5	In the `PLC`, click on `Execute`. The `Execute` window opens. Click on `OK`. The application is now being executed (in RUN mode) on the PLC simulator.

Execution of Application in Standard Mode

At a Glance

To work in standard mode you need to use a PLC and Discrete and Analog I/O modules to assign outputs to different sensors and actuators.
The variables used in simulation mode must be modified. In standard mode, variables must be located to be associated to physical I/Os.

> **Note:** For more information on addressing, see Unity Pro online help (click on , then `Unity`, then `Unity Pro`, then `Languages reference`, then `Data description` and `Data instances`).

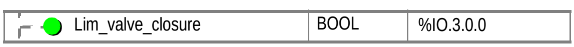
Application Hardware Configuration

The table below shows the procedure for configuring the application.

Step	Action
1	In the `Project browser` double-click on Configuration then on `0:Bus X` and on `0:TSX RKY` ••• (where 0 is the rack number).
2	In the `Bus X` window, select a slot, for example 3 and double-click on it.
3	Insert a discrete input module, for example `TSX DEY 16A5`.
4	Confirm with `OK`. This input module is used to connect the application's EBOOL inputs.

Assignment of Variables to Input Module

The table below shows the procedure for direct addressing of variables:

Step	Action
1	In the `Project browser` and in `Variables & FB instances`, double-click on `Elementary variables`.
2	In the `Address` column, enter the address associated with the variable in the form Rack\Module\Channel\Data. **Example:** On the TSX DEY 16A5 module, there are 2 channels, channel 0 and channel 8. Channel 0 handles inputs 0 to 7 and channel 8 handles inputs 8 to 15. If the valve closure limit switch output is connected to input 0 of the module, the address %I0.3.0.0 is displayed in the address column of the editor for the Lim_valve_closure variable Illustration: 
3	Repeat the same procedure for all located variables.

Application Execution

The table below shows the procedure for launching the application in standard mode:

Step	Action
1	In the `PLC` menu, click on `Standard Mode`,
2	In the `Build` menu, click on `Rebuild All Project`. Your project is generated and is ready to be transferred to the PLC. When you generate the project, you will see a results window. If there is an error in the program, Unity Pro indicates its location if you click on the highlighted sequence.
3	In the `PLC` menu, click on `Connection`. You are now connected to the PLC.
4	In the `PLC` menu, click on `Transfer project to PLC`. The `Transfer project to PLC` window opens. Click on `Transfer`. The application is transferred to the PLC.
5	In the `PLC`, click on `Execute`. The `Execute` window opens. Click on `OK`. The application is now being executed (in RUN mode) on the PLC.

Diagnostics Viewer

At a Glance

The diagnostics viewer enables you to monitor variables when they are associated to diagnostics function blocks (ALMR_DIA for example).

Note: For more information on the declaration of these variables for diagnostics purposes, go to the DFB section (See *Procedure for Customizing an Existing DFB from a Library DFB, p. 29*).

Creation of Diagnostics

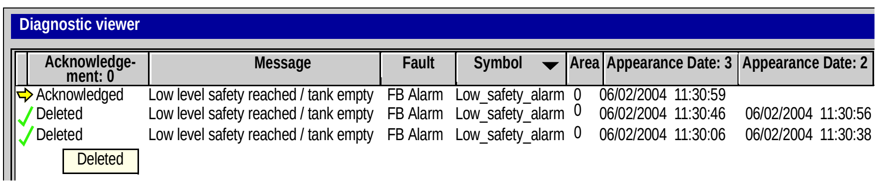
The table below shows the procedure for displaying the diagnostics window:

Step	Action
1	In the `Tools` menu, click on `Diagnostics Viewer`. The window is displayed on-screen.
2	As soon as the `Tank_low_safety` or `Tank_high_safety` or `Valve_opening_error` or `Valve_closure_error` variables switch from 0 to 1, a message is displayed in the diagnostics viewer.

Note: For more information, see Unity Pro online help (click on , then `Unity`, then `Unity Pro`, then `Operate modes`, and `Diagnostics`).

Illustration of the Diagnostics Viewer

The illustration below shows an example of what is displayed when the Tank_low_safety variable switches from 0 to 1:



Glossary

!

%I According to the IEC standard, `%I` indicates a discrete input-type language object.

%M According to the IEC standard, `%M` indicates a memory bit-type language object.

%MW According to the IEC standard, `%MW` indicates a memory word-type language object.

%Q According to the IEC standard, `%Q` indicates a discrete output-type language object.

B

BIT This is a binary unit for a quantity of information which can represent two distinct values (or statuses): 0 or 1.

BOOL `BOOL` is the abbreviation of Boolean type. This is the elementary data item in computing. A `BOOL` type variable has a value of either: 0 (`FALSE`) or 1 (`TRUE`).
A `BOOL` type word extract bit, for example: `%MW10.4`.

BYTE When 8 bits are put together, this is called a `BYTE`. A `BYTE` is either entered in binary, or in base 8.
The `BYTE` type is coded in an 8 bit format, which, in hexadecimal, ranges from `16#00` to `16#FF`

D

DFB

DFB is the abbreviation of Derived Function Block.
DFB types are function blocks that can be programmed by the user ST, IL, LD or FBD.
By using DFB types in an application, it is possible to:
- simplify the design and input of the program,
- increase the legibility of the program,
- facilitate the debugging of the program,
- reduce the volume of the generated code.

DFB instance

A DFB type instance occurs when an instance is called from a language editor.
The instance possesses a name, input/output interfaces, the public and private variables are duplicated (one duplication per instance, the code is not duplicated).
A DFB type can have several instances.

E

EBOOL

`EBOOL` is the abbreviation of Extended Boolean type. It can be used to manage rising or falling edges, as well as forcing.
An `EBOOL` type variable takes up one byte of memory.

EFB

Is the abbreviation for Elementary Function Block.
This is a block which is used in a program, and which performs a predefined software function.
EFBs have internal statuses and parameters. Even where the inputs are identical, the output values may be different. For example, a counter has an output which indicates that the preselection value has been reached. This output is set to 1 when the current value is equal to the preselection value.

F

FBD

FBD is the abbreviation of Function Block Diagram.

FBD is a graphic programming language that operates as a logic diagram. In addition to the simple logic blocks (`AND`, `OR`, etc.), each function or function block of the program is represented using this graphic form. For each block, the inputs are located to the left and the outputs to the right. The outputs of the blocks can be linked to the inputs of other blocks to form complex expressions.

Function view View making it possible to see the program part of the application through the functional modules created by the user (see Functional module definition).

I

IEC 61131-3 International standard: Programmable Logic Controls
Part 3: Programming languages.

IL IL is the abbreviation of Instruction List.
This language is a series of basic instructions.
This language is very close to the assembly language used to program processors.
Each instruction is composed of an instruction code and an operand.

Instantiate To instantiate an object is to allocate a memory space whose size depends on the type of object to be instantiated. When an object is instantiated, it exists and can be manipulated by the program.

INT `INT` is the abbreviation of single integer format (coded on 16 bits).
The lower and upper limits are as follows: -(2 to the power of 31) to (2 to the power of 31) - 1.
Example:
`-32768`, `32767`, `2#1111110001001001`, `16#9FA4`.

L

LD LD is the abbreviation of Ladder Diagram.
LD is a programming language, representing the instructions to be carried out in the form of graphic diagrams very close to a schematic electrical diagram (contacts, coils, etc.).

Located variable

A located variable is a variable for which it is possible to know its position in the PLC memory. For example, the variable `Water_pressure`, is associated with `%MW102`. `Water_pressure` is said to be located.

M

Master task

Main program task.
It is obligatory and is used to carry out sequential processing of the PLC.

O

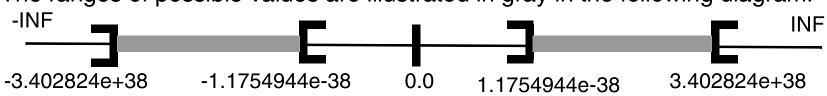
Operator screen

This is an editor that is integrated into Unity Pro, which is used to facilitate the operation of an automated process. The user regulates and monitors the operation of the installation, and, in the event of any problems, can act quickly and simply.

R

REAL

Real type is a coded type in 32 bits.
The ranges of possible values are illustrated in gray in the following diagram:



When a calculation result is:

- between -1.175494e-38 and 1.175494e-38 it is considered as a `DEN`,
- less than -3.402824e+38, the symbol `-INF` (for -infinite) is displayed,
- greater than +3.402824e+38, the symbol `INF` (for +infinite) is displayed,
- undefined (square root of a negative number), the symbol `NAN` is displayed.

S

Section

Program module belonging to a task which can be written in the language chosen by the programmer (FBD, LD, ST, IL, or SFC).

A task can be composed of several sections, the order of execution of the sections corresponding to the order in which they are created. This order is modifiable.

SFC

SFC is the abbreviation of Sequential Function Chart.
SFC enables the operation of a sequential automation device to be represented graphically and in a structured manner. This graphic description of the sequential behavior of an automation device, and the various situations which result from it, is provided using simple graphic symbols.

SFC objects

An SFC object is a data structure representing the status properties of an action or transition of a sequential chart.

ST

ST is the abbreviation of Structured Text language.
Structured Text language is an elaborated language close to computer programming languages. It enables you to structure series of instructions.

Structure

View in the project navigator with represents the project structure.

Subroutine

Program module belonging to a task (Mast, Fast) which can be written in the language chosen by the programmer (FBD, LD, ST, or IL).
A subroutine may only be called by a section or by another subroutine belonging to the task in which it is declared.

T

Task

A group of sections and subroutines, executed cyclically or periodically for the MAST task, or periodically for the FAST task.
A task possesses a level of priority and is linked to inputs and outputs of the PLC. These I/O are refreshed in consequence.

TIME

The type `TIME` expresses a duration in milliseconds. Coded in 32 bits, this type makes it possible to obtain periods from 0 to (2 to the power of 32)-1 milliseconds.

U

Unlocated variable

An unlocated variable is a variable for which it is impossible to know its position in the PLC memory. A variable which have no address assigned is said to be unlocated.

V

Variable

Memory entity of the type `BOOL`, `WORD`, `DWORD`, etc., whose contents can be modified by the program during execution.

W

WORD

The `WORD` type is coded in 16 bit format and is used to carry out processing on bit strings.
This table shows the lower/upper limits of the bases which can be used:

Base	Lower limit	Upper limit
Hexadecimal	16#0	16#FFFF
Octal	8#0	8#177777
Binary	2#0	2#1111111111111111

Representation examples

Data content	Representation in one of the bases
0000000011010011	16#D3
1010101010101010	8#125252
0000000011010011	2#11010011

Index