European Organisation for Astronomical Research in the Southern Hemisphere

Programme: ELT

Project/WP: Cryo HMI Framework

ELT Cryo HMI Framework User Manual

Document Number: ESO-000000

Document Version: 1

Document Type: Manual (MAN)

Released on: 2024-08-01

Document Classification: ESO Internal [Confidential for Non-ESO Staff]

Owner:	Del Valle, Diego
Validated by WPM:	Kiekebusch, Mario
Approved by PM:	Kiekebusch, Mario
	Name

Release

This document corresponds to CryoHmiFramework[1] v1.1.0.

Authors

Name	Affiliation
Diego del Valle	ESO

Change Record from previous Version

Affected Section(s)	Changes / Reason / Remarks
TODO	TODO

[1] https://gitlab.eso.org/ifw/ifw-ll

Contents

1 Introduction

Cryo HMI Framework Control is a collection of TE2000 HMI Framework Controls that are made specifically to be used with IFW PLC's Cryo Library.

Warning: Usually, any user of Eso Cryo Framework should not directly use this package. IFW provides an HMI Template Project with pre-made user controls. But we leave this available for documentation purposes and, if for any reason, the developer decides to implement a project specific user control based on the framework.

ESO shall be contacted first to decide if there is a real need or not.

1.1 Scope

This document is the user manual for Cryo HMI Framework Control version 1.0.0. The intended audience are ELT users, consortia developers or Cryogenic Engineers.

This Package is one of the deliverables for ESO Cryo Framework, which is composed by:

- **Cryo Library**: PLC Library for standard ESO Cryogenic devices.
- **Cryo HMI Framework Controls**: Collection of HMI Framework controls that will interface with Cryo Library (this document)
- **Cryo HMI Template Project**: Start-up project to create an HMI Application. The project comes with a base project structure, UserControls and references to ESO libraries.

1.2 Acronyms

ELT	Extremely Large Telescope
FB	Function Block
GUI	Graphical User Interface
HMI	Human Machine Interface
IFW	Instrument Framework
PLC	Programming Logical Controller

1.3 Concepts

Quick guide of some recurring concepts in the documentation.

Color scheme

Cryo HMI Framework Controls use a color scheme to deliver visual messages, like states or events. The following table summarizes the ideas behind the chosen colors.

Table 1: General Color Scheme

Color	Description
Red	Abnormal, Alarm condition which requires attention.
Purple	Abnormal, Warning condition, raising awareness about a potential upcoming alarm condition.
White	Normal condition, Actuator Off/Closed.
Green	Normal condition, Actuator On/Opened.
Yellow	Manual Operation Mode activated
Orange	Engineering Mode activated / Interlock bypassed
Light blue	Simulation mode activated.

This general color scheme is supplemented by flashing between two colors. which is used to indicate changes and/or transitions. When in transition, the first color will be the color of the desired new state, the second color is a transition color (Normally is a Dark Gray color).

Data validity

Data Validity is a way to inform the user that the current data displayed or device cannot be trusted or is disabled.

When Data Validity is False, the controls will show a dark grey background with a diagonal yellow line across its diagonal.

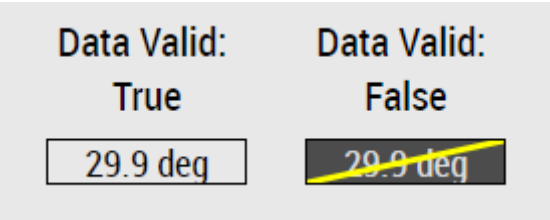


Fig. 1: Example of *Data Label* with Data *Valid* and *Not Valid*

2 Installation

The following section shows how to install Cryo HMI Framework Controls NuGet package into a project.

Warning: This manual assumes that the user has already knowledge of TE2000[1]

2.1 Requirements

- TwinCAT XAE 3.1.4024.n, where n > 25. The same as IFW.
- TwinCAT 3 HMI Engineering | TE2000: 12.760.59.

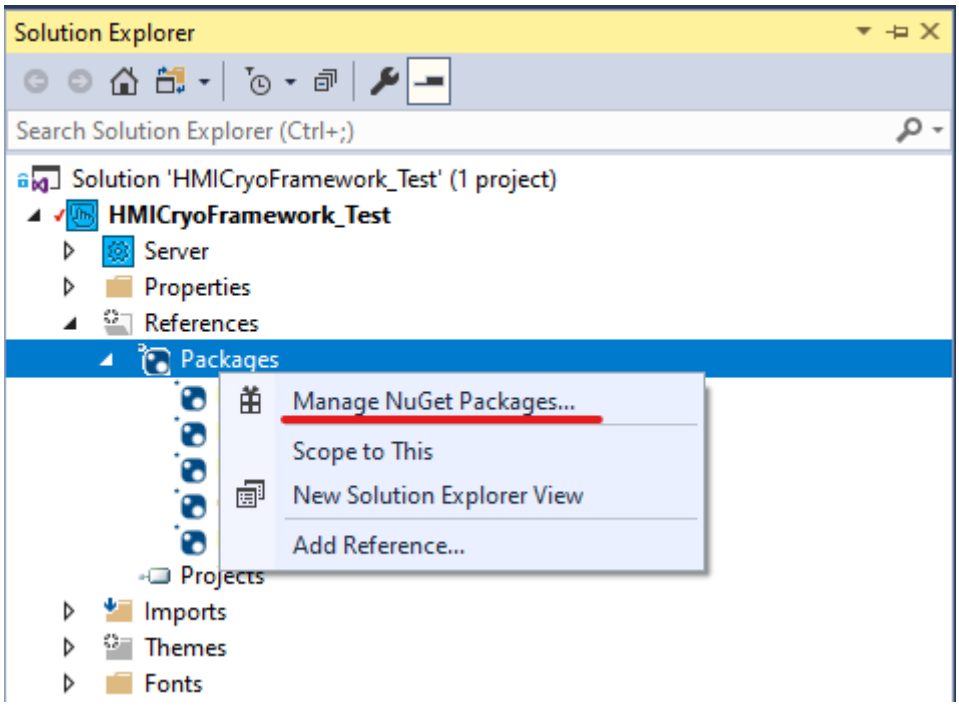
2.2 Install Cryo HMI Framework Controls NuGet Package

1. Download ESO **CryoFramework.<version>.nupkg** file from GitLab
2. Check on XAE the current configured **TwinCAT HMI Customer** package folder. Go to **Tools** → **NuGet Package Manager** → **Package Manager Settings** → **Package Sources**. It is usually located at $\mathrm{C:\backslash TwinCAT\backslash Functions\backslash TE2000\text{-}HMI\text{-}Engineering\backslash References}$.
3. Copy or move the downloaded file to **TwinCAT HMI Customer** folder.

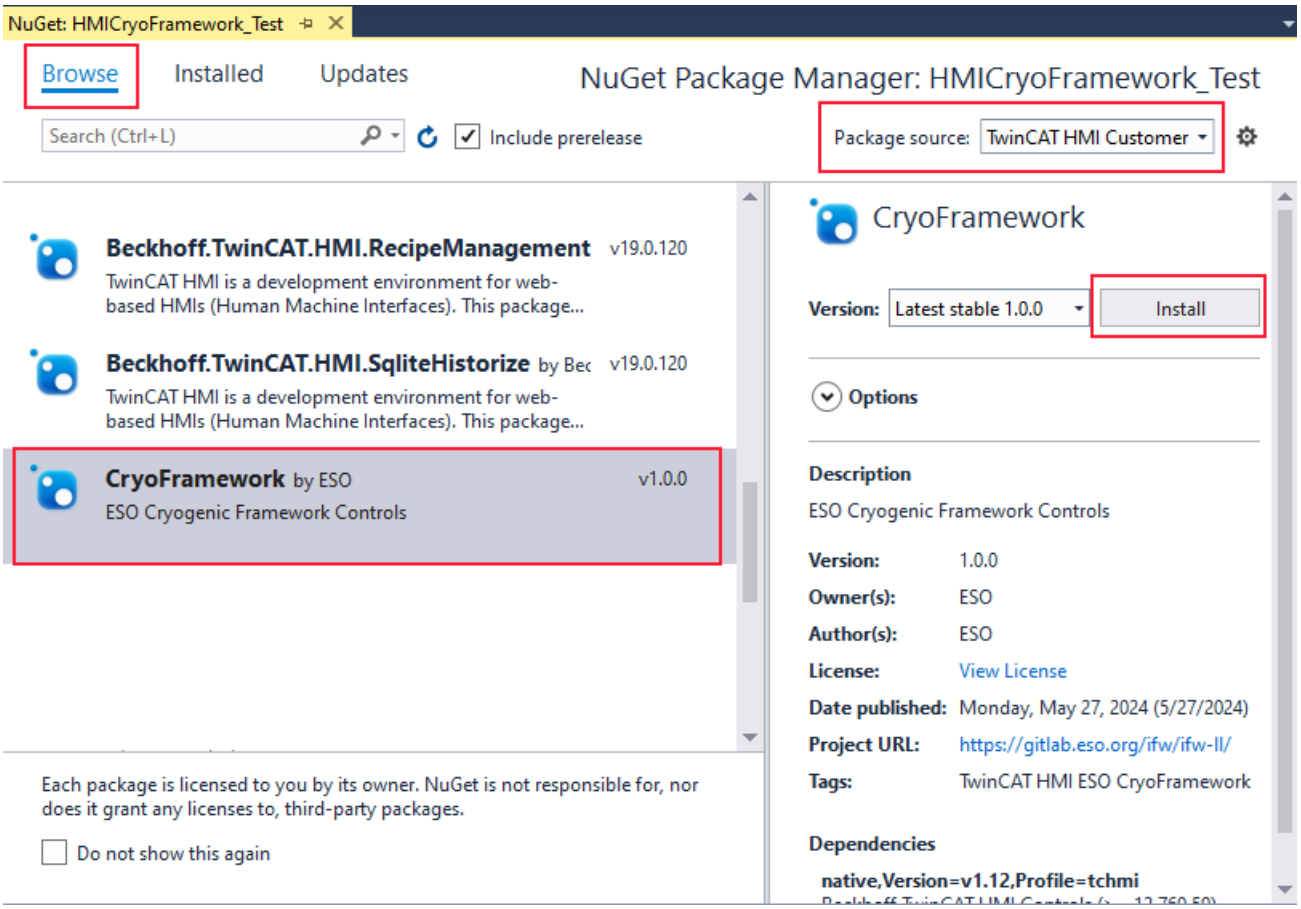
2.3 Installing NuGet Package on your project

1. In the TwinCAT HMI project right-click on **References** node, then click **Manage NuGetPackages**

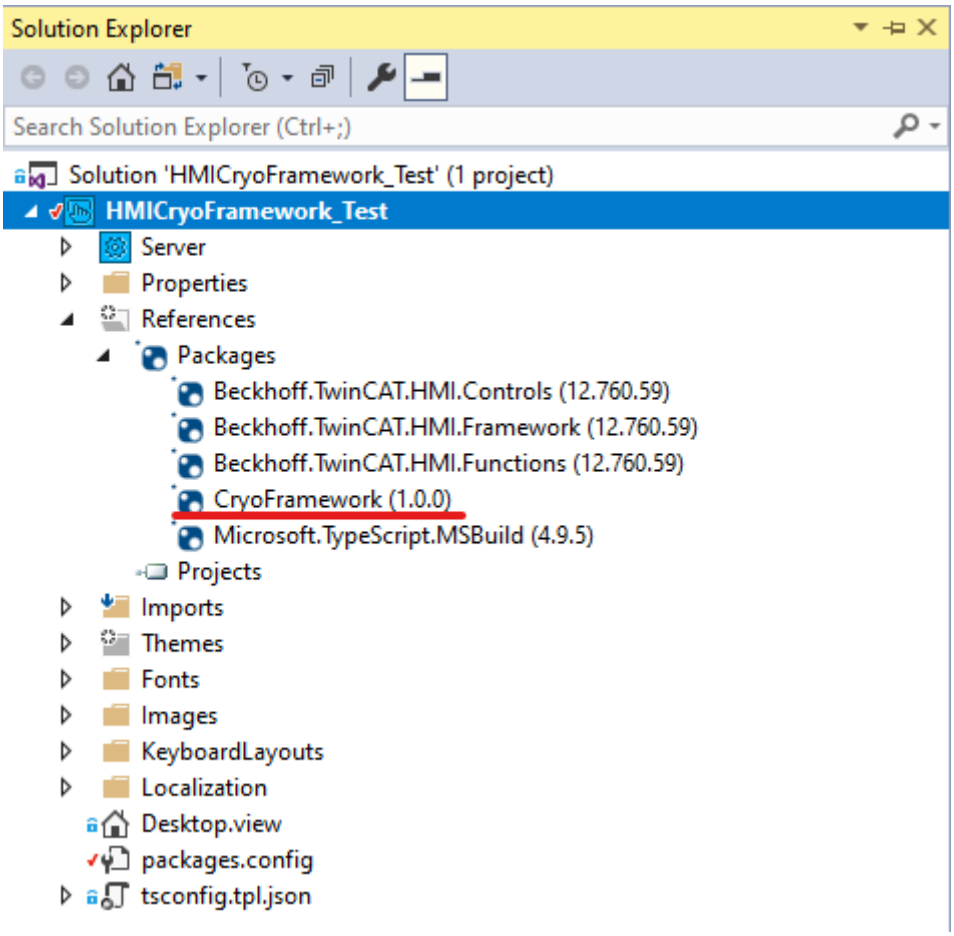
[1] https://infosys.beckhoff.com/english.php?content=../content/1033/te2000_tc3_hmi_engineering/index.html



2. In the **NuGet Package Manager** select the **Browse** Tab.
3. Make sure Package source is **TwinCAT HMI Customer**.
4. Search for **CryoFramework by ESO** and press the Install Button. Press **OK** button in the **Preview Changes** window.



5. Make sure the package is installed by looking at the HMI project **Reference** → **Packages** list.



2.4 Checking the Installation

Open any view or content page from your HMI project. Then open the Toolbox (Alternatively from the menu **View** → **Toolbox**)

Four new categories will appear, each one with a set of Framework Controls:

- ESO Cryo Controls
- ESO Cryo Components
- ESO Cryo Sequences
- ESO Cryo Symbols

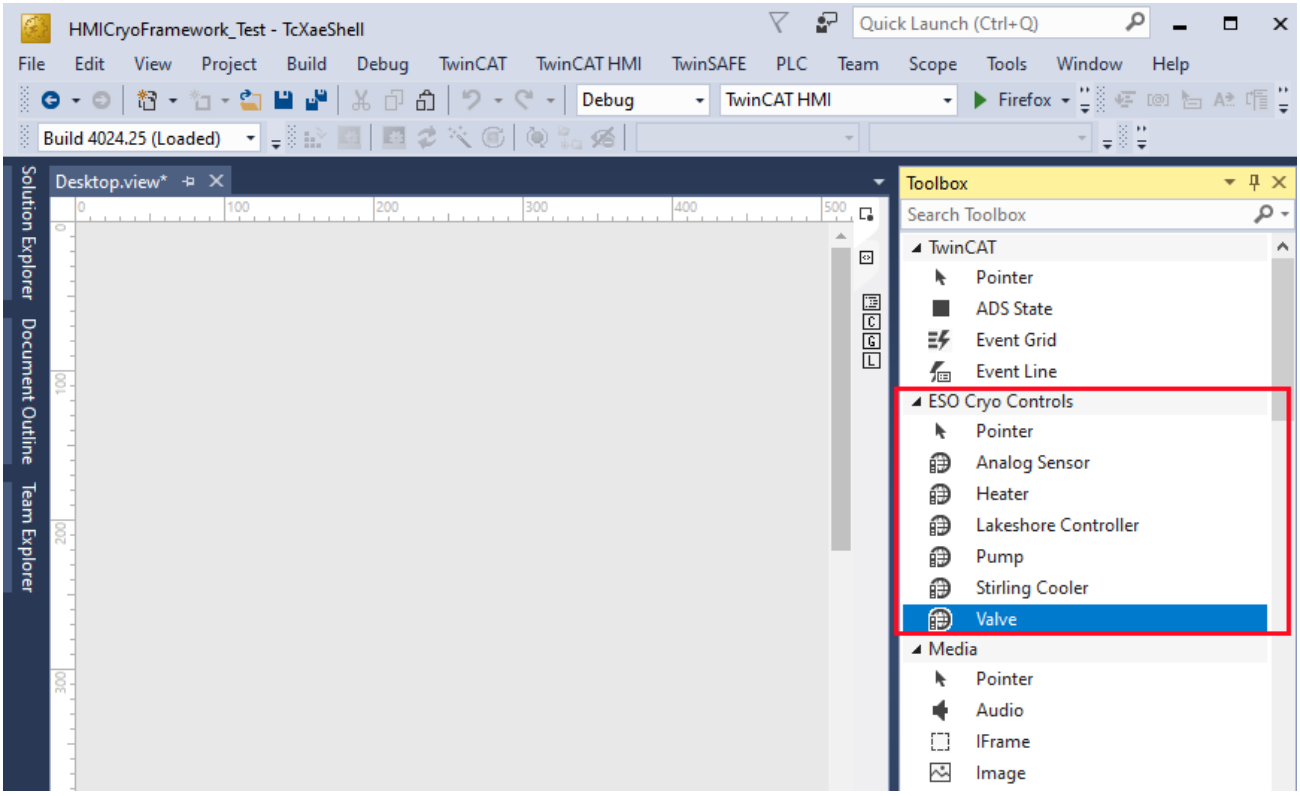


Fig. 1: **ESO Cryo Controls** loaded in the project after installing CryoFramework.nupkg

Categories and the installed Framework controls are explained in the next section.

3 Cryo HMI Framework Controls

3.1 Overview

Cryo HMI Framework Controls is a collection of TE2000 Framework Controls[2] created with web technologies (HTML, CSS, Typescript/Javascript) to support the development of Cryo HMI applications. They are created to be compatible with Web resposive design.

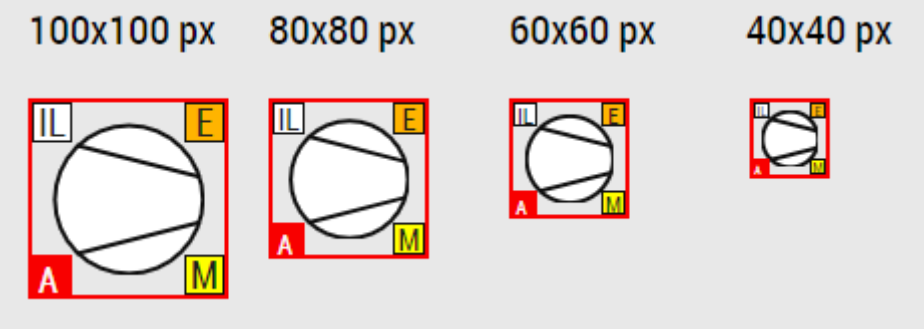


Fig. 1: Pump Controller with different pixel sizes.

Framework Controls are separated into four categories *Symbols*, *Components*, *Controls* and *Sequences*. Sequences are explained in their own section.

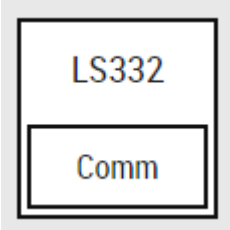
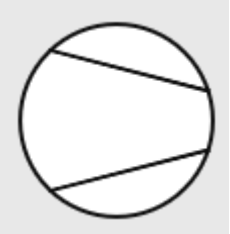
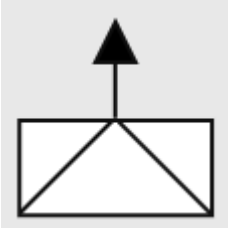
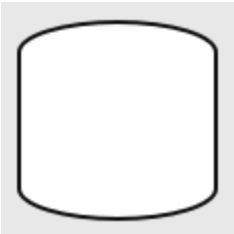
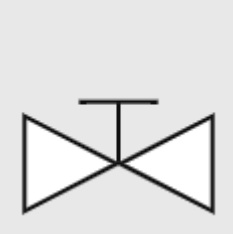
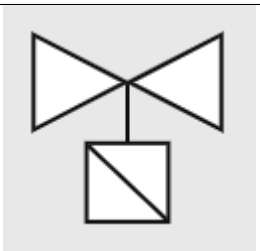
3.2 Symbols

Image representation of a Cryogenic device. They all share a common functionality with respect of the composition of the image and its attributes. There are some Symbols that require extra attributes (i.e. Lakeshore Controller Symbol), but in general terms, they share the same behavior.

There are 9 supported Symbols, and 11 considering alternatives:

[2] https://infosys.beckhoff.com/english.php?content=../content/1033/te2000_tc3_hmi_engineering/8971699467.html

Table 1: Symbols Table

Gauge Symbol	Lakeshore Controller Symbol	Pump (Prevacuum) Symbol	Pump (Turbomolecular) Symbol
			
Rupture Disc Symbol	Safety Overpressure Valve Symbol	Sorption Pump Symbol	Stirling Cooler Symbol
			
Tank Symbol	Valve (Manual) Symbol	Valve (Automatic) Symbol	
			

Visually, all Symbols can represent 4 states: Off, Starting, On and Stopping. Even when in some cases it doesn't make sense.

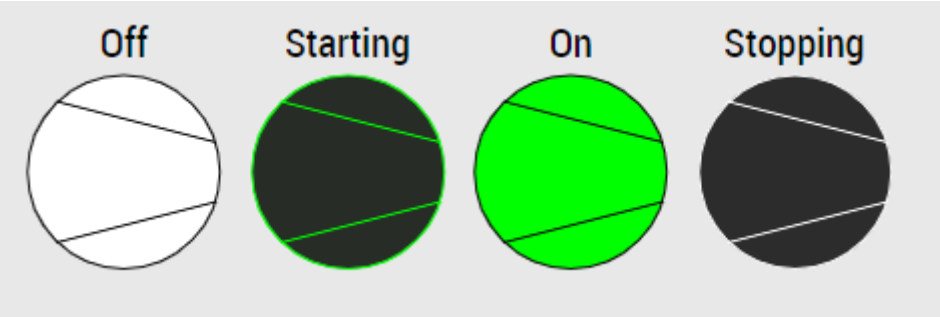


Fig. 2: Pump Symbol state colors and animations

Note: Starting is an animation between Starting/On colors, and Stopping is an animation between Stopping/Off colors

Most of the symbols can also rotate in steps of 90 degrees: Right, Up, Left and Down. With the exception being the Lakeshore Controller.

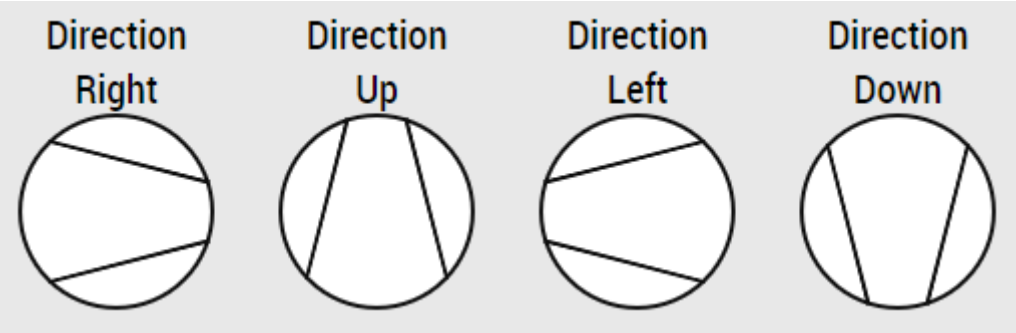


Fig. 3: Pump Symbol possible rotations

Interface

Table 2: Symbols interface Table

Name	Group	Description	Availability
StrokeThickness	Shape	Thickness of the Stroke of the symbol (default: 1)	All
State	Common	Defines the state of the symbol: Off, Starting, On, Stopping (default: Off)	All
Value	Common	Text shown on Lakeshore controller (default: Empty String)	Lakeshore Controller Symbol
SymbolType	Common	Alternative Symbol to show. Pump: Prevacuum or Turbomolecular; Valve: Auto or Manual.	Only with Pump and Valve Symbol
Direction	Common	Defines the direction the symbol will be drawn: Up, Down, Left, Right (default: Up)	Disabled in Gauge and Lakeshore
Drawing Stroke Color	Colors	Color of the Drawing Stroke (default: rgb(15, 15, 15), dark: rgb(92, 92, 92))	All
Drawing Transition Color	Colors	Transition color when state is Starting or Stopping (default: rgb(40, 40, 40), dark: rgb(53, 53, 53))	All
On Fill Color	Colors	Sets the color that represents On State (default: rgb(0, 255, 0), dark: rgb(1, 165, 1))	All
Off Fill Color	Colors	Sets the color that represents Off State (default: rgb(255, 255, 255), dark: rgb(255, 255, 255))	All
TextFontSize	Text	Size of the Font of the Value Attribute (default: 12)	Lakeshore Controller Symbol
TextCommFontSize	Text	Size of the Font for the text on “Comm” rectangle (default: 12)	Lakeshore Controller Symbol

Warning: As a rule of thumb, all attributes in the Color Group should not be changed. All Cryo HMI Framework Controls are already defined with the correct *ESO Color Scheme*. Changing them will also break Dark mode compatibility.

3.3 Components

Reusable framework controls that are available primarily to be used in a control composition, but that may also be used individually.

Data Label

Control that extends the functionality of TE2000 Textblock[3]. It has a flag **Data Valid** which makes the Control turn grey and adds a crossing yellow line to show the user that whatever data is being displayed it's not valid anymore.

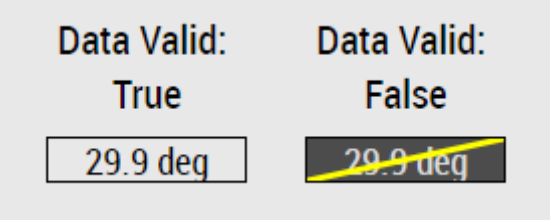


Fig. 4: Data Label with flag **Valid Data: True**, and with **Valid Data: False**. Added borders to show size.

Interface

The interface is the same as Textblock plus the following attributes.

Table 3: Data Label Interface Table

Name	Group	Description
Data Valid	Common	Flag for data validity
Data Invalid Text Color	Colors	Text Color when flag **Data Valid: False**
Data Invalid Stroke Color	Colors	Crossing line stroke color when flag **Data Valid: False**
Data Invalid Background Color	Colors	Background Color when flag **Data Valid: False**

[3] https://infosys.beckhoff.com/english.php?content=../content/1033/te2000_tc3_hmi_engineering/3845334283.html

Led Box

Multipurpose Box that acts as the bare minimum as a LED to show state On and Off with the user selected state colors.

This control also extends the functionality of Cryo HMI Framework Controls's *DataLabel* and support adding intermediate states by writing to the blink status flag.

When Data is not Valid, it overrides the State Color and Blinking, it's shown with the same characteristics as DataLabel control.

Fig. 5: All possible sates of LedBox Control with default colors

Note: On/Blink is an animation between On and On/Blink images of the figure, similarly, Off/Stopping is an animation between Off and Off/Blink images of the figure.

Interface

The interface is exactly the same as *DataLabel* plus the following attributes.

Table 4: LedBox Interface Table

Name	Group	Description
Data Valid	Common	Flag for data validity
State	Common	Flag for On/Off State
Blink	Common	Flag to show LedBox blinking. Colors will depend on **State** flag.
On Fill Color	Colors	Led Box color when **State: True** and **DataValid: True**.
Off Fill Color	Colors	Led Box color when **State: Off** and **DataValid: True**.

Device State

Shows the user a specific set of device states by adding information boxes at the corner of a device symbol. The states are fixed and are as follows:

- **Manual Mode**: Box on the lower right corner with a yellow color fill and the letter **M**.
- **Engineering Mode**: Box on the upper right corner with an orange color fill and the letter **E**.
- **Interlock Active**: Box on the upper left corner with a white color fill and the letters **IL**.

Under normal operations, these states should be not active and hidden from the user.

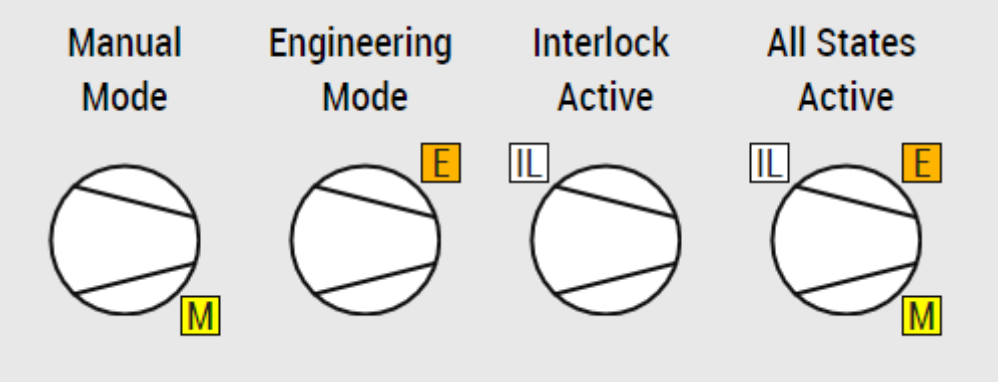


Fig. 6: Device States individually active (1, 2, 3) and All States active (4), the included Pump Symbol is for reference

Interface

Table 5: DeviceState interface Table

Name	Group	Description
Manual Mode	Common	Device Manual Mode On/Off
Engineering Mode	Common	Device Enginnering Mode On/Off
Interlock Active	Common	Interlock is currently Active
Box Size	Shape	Sets the size of the boxes in Pixels or by a percentage all 3 states: Manual, Engineering and Interlock Active (default: 20 %).

Event Box

Representation of the raised/confirmed events of the system. An event can be an Alarm and/or a Warning.

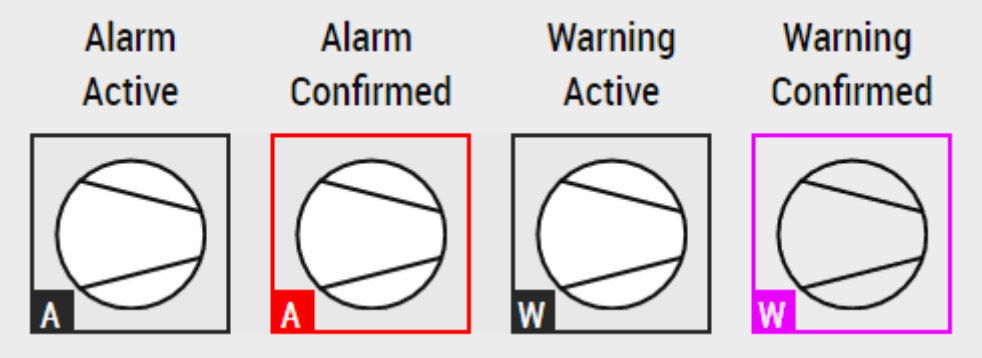


Fig. 7: Event Box Combination of States, the included Pump Symbol is for reference

Note: Alarm Active is an animation between Alarm Active and Confirmed, and Warning Active is an animation between Warning Active and Confirmed.

A device is enclosed with a colored rectangle that represents an event, it also contains a box with a letter **A** for Alarm and **W** for Warning. Alarms are represented in **Red** color, and Warnings are represented in **Purple** color. If no event is registered, the Framework Controls cannot be seen.

If an alarm or warning are active, the control will blink between the event color and black to alert the users of the new event. If the event is confirmed, then the control will stop blinking.

Note: If both events are active at the same time, an Alarm will always have priority over a warning.

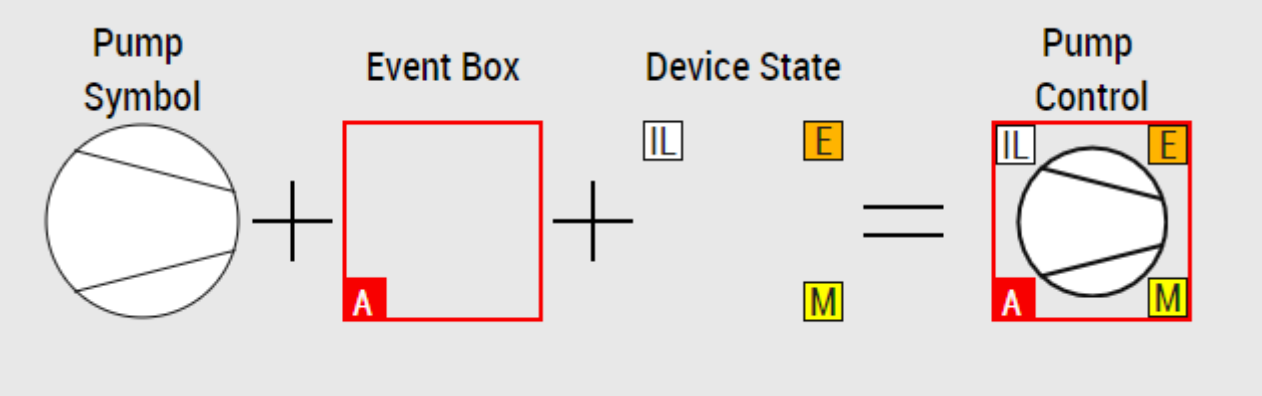
Interface

Table 6: Event Box Interface

Name	Group	Description
Alarm Raised	Common	Alarm Raised State (Active or Not)
Alarm Confirmed	Common	Alarm Confirmed State (Active or Not)
Warning Raised	Common	Warning Raised State (Active or Not)
Warning Confirmed	Common	Warning Confirmed State (Active or Not)
Text Box Size	Shape	Sets the size of the lower left box in Pixels or by a percentage of the Control minimum side length (default: 20 px).
Stroke Thickness	Shape	Thickness of the rectangle that shows Alarm/Warning state. (default: 2 px)

3.4 Controls

High level representation of a Cryogenic Device. Generated by a composition of other Cryo HMI Framework Controls: *Symbol*, *Event Box* and *Device State*.



Common

With the exception of *Analog Sensor*. All Controls share common traits. They all have a *Symbols* that go through 4 different states: **Off(0)**, **Starting(1)**, **On(2)** and **Stopping(3)**. They also show Alarms and Warning with an *Event Box*. And finally they all show their device state thanks to their *Device State* components.

Interface

Table 7: Common Controls Interface Table

Name	Group	Description	Availability
State	Common	INT value that represents the state of the device: Off(0), Starting(1), On(2) and Stopping(3)	All
Keep Aspect Ratio	Common	Maintains a square shape of the Control regardless of the actual size. Recommended value: True	All but Heater Control
Alarm Raised	*Event Box*	Alarm Raised State (Active or Not)	All
Alarm Confirmed	*Event Box*	Alarm Confirmed State (Active or Not)	All
Warning Raised	*Event Box*	Warning Raised State (Active or Not)	All
Warning Confirmed	*Event Box*	Warning Confirmed State (Active or Not)	All
Box Size	*Event Box*	Sets the size of the box in Pixels or by a percentage of the Control minimum side length (default: 20 %).	All
Stroke Thickness	*Event Box*	Thickness of the rectangle that shows Alarm/Warning state.	All
Manual Mode	*Device State*	Device Manual Mode On/Off	All
Engineering Mode	*Device State*	Device Engineering Mode On/Off	All but Lakeshore Controller
Interlock Active	*Device State*	Interlock is currently Active	All
Box Size	*Device State*	Sets the size of the boxes in Pixels or by a percentage all 3 states: Manual, Engineering and Interlock Active (default: 20 %).	All
Direction	*Symbol*	Selects Direction of the Symbol	All but Lakeshore Controller
Symbol Margin Size (%)	*Symbol*	Specifies the margin size of the Symbol with respect of the control. Values over 50% won't make sense	All
Stroke Thickness	*Symbol*	Symbol Stroke Thickness in pixels (px)	All

Heater

Heater Framework Control with all interfaces needed to be connected to PLC CryoLib.

The Heater Control is represented by a rectangle that supports adding a text inside of it. This text can be a string or a representation of a sensor value. The text has modifiers in the **Text** group attributes.

The heater can also be displayed as being disabled by using the DataValid flag.

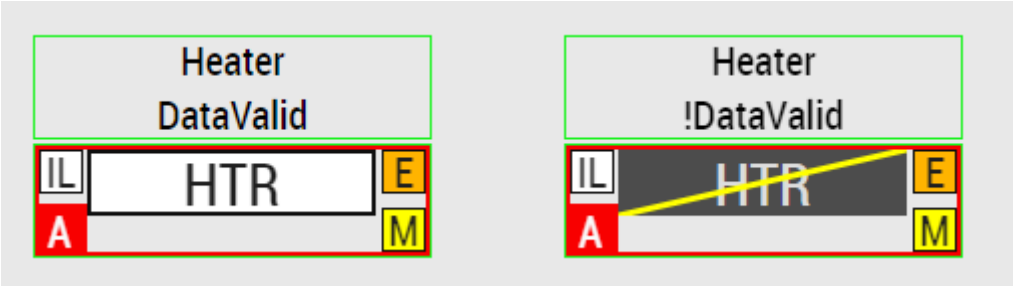


Fig. 8: Heater with DataValid Flag On and Off

Interface

This interface extends *Common Interface*.

Table 8: Heater interface Table

Name	Group	Description
Value	Common	Text or value to be displayed inside the Heater symbol
DataValid	Common	Flag for data validity. It works as *Data Label*
TextFontSize	Text	Size of the font for the text displayed by **Value** attribute
TextHorizontalAlignment	Text	Horizontal alignment inside the heater symbol for the text displayed by **Value** attribute
TextVerticalAlignment	Text	Vertical alignment inside the heater symbol for the text displayed by **Value** attribute

Lakeshore Controller

Lakeshore Controller Control with all interfaces needed to be connected to PLC CryoLib.

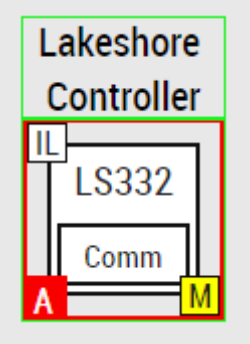


Fig. 9: Lakeshore Controller

The Lakeshore control symbol are 2 rectangles, the big one represents the state of the Lakekeshore controller, and the inner and smaller one, represents the state of the communication.

The user can add any text or value in the big rectangle by using the attribute **Value**, usually the text is the name of the controller.

The state of communication can be changed in the same way as the overall **State** of the system, if there is no feedback, then both attributes **State** and **CommState** should just reflect the state of the system.

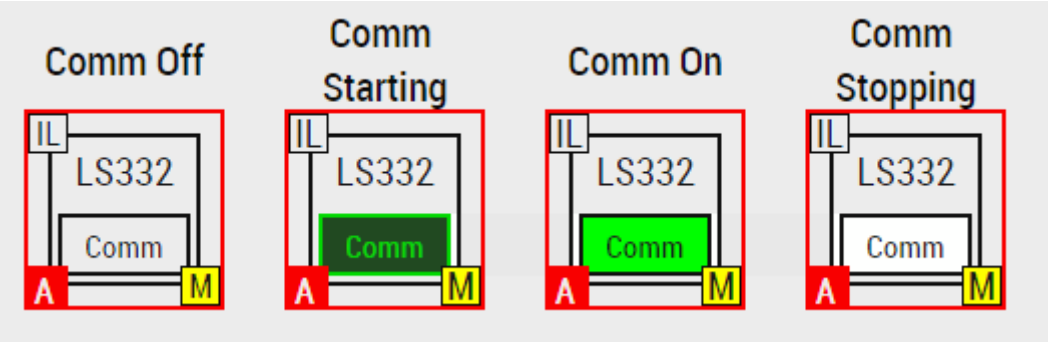


Fig. 10: Lakeshore Controller CommState: Off(0), Starting(1), On(2) and Stopping(3)

Note: Starting is an animation between Starting/On colors, and Stopping is an animation between Stopping/Off colors

Lakeshore Controller doesn't support changing the direction of the symbol, since it doesn't make sense. And it also doesn't support the **Engineering Mode** flag.

Interface

This interface extends *Common Interface*.

Table 9: Lakeshore controller interface Table

Name	Group	Description
Value	Common	Text or value to be displayed inside the Lakeshore Controller Symbol
CommState	Common	INT value that represents the state of the communication: Off(0), Starting(1), On(2) and Stopping(3)

Pump

Pump Framework Control with all interfaces needed to be connected to PLC CryoLib.

Pump Control complies completely with the *Common Controller* and it just adds the attribute **Symbol Type** to select between *Prevacuum* and *Turbomolecular* symbols.

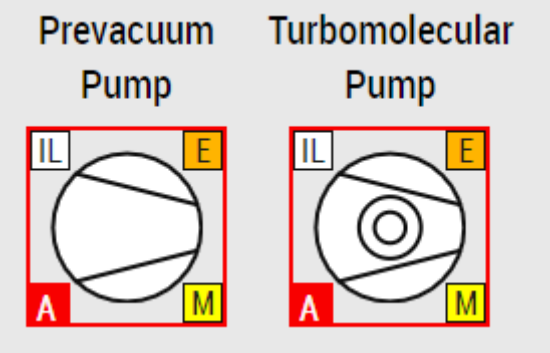


Fig. 11: Pump Control and its two varians: Prevacuum and Turbomolecular

Interface

This interface extends *Common Interface*.

Table 10: Pump interface Table

Name	Group	Description
Symbol Type	*Symbol*	Selects type of Pump to show: Prevacuum or Turbomolecular.

Stirling Cooler

Stirling Cooler Control with all interfaces needed to be connected to PLC CryoLib.

It complies completely with the *Common Controller*.

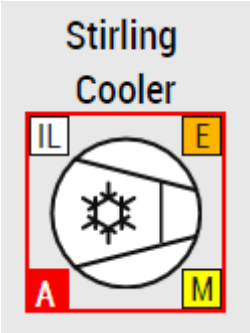


Fig. 12: Stirling Cooler Control

Interface

This interface is exactly as *Common Interface*.

Valve

Valve Framework Control with all interfaces needed to be connected to PLC CryoLib.

Valve Control complies completely with the *Common Controller* and it just adds the attribute **Symbol Type** to select between *Manual* and *Auto* symbols.

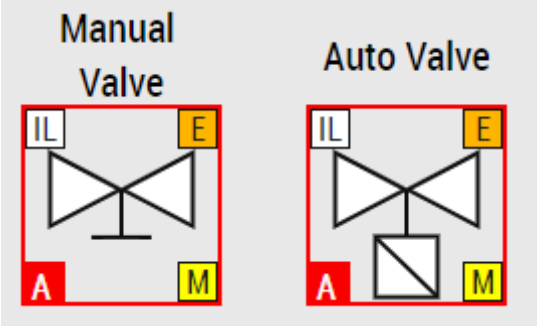


Fig. 13: Valve Control and its two varians: Manual and Auto.

Alternatively, Valve can show the Open/Close feedback by using the flag **Show Feedback**, which makes visible to Led Boxes with *Op* (for Opened) and *Cl* (for Closed). The boxes state can be changed by the attributes **Open State** and **Close State** respectively.

The last attribute **Valid Feedback** is used to show if the feedback values are valid or not.

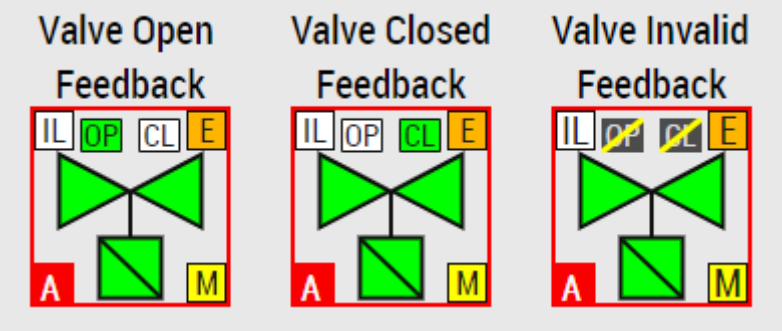


Fig. 14: Valve Control with feedback: Open, Close and !DataValid

Interface

This interface extends *Common Interface*.

Table 11: Valve interface Table

Name	Group	Description
Show Feedback	Common	Show Opened and Closed feedback boxes.
Open State	Common	Flag for Opened State.
Close State	Common	Flag for Closed State.
Valid Feedback	Common	Flag for data validity. It works as *Data Label*.
Symbol Type	*Symbol*	Selects type of Valve to show: Manual or Auto.

Analog Sensor

Analog sensor is a Control that does not conform to the *Common Controller* specification.

It is basically a super powered label that contains many extras for sensor reading data display. The control allows to select between Standard and Scientific notation and to include the sign in any case.

	not Signed	Signed
Standard	12345.67 deg	+1.23e+4 deg
Scientific	7.36e+10 deg	+1.23e+4 deg

Fig. 15: Analog Sensor Control: Notation and Signed Flags

It also supports *Event Box*.

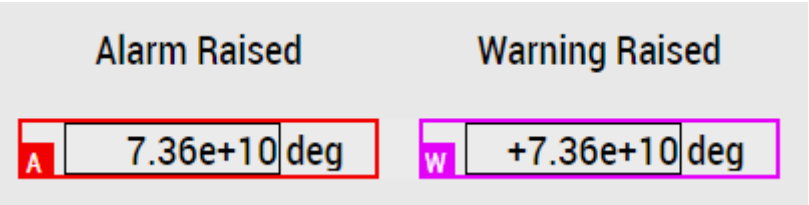


Fig. 16: As *Event Box* the events can be blinking when in active state

If the device is in Simulation State, it'll show a light blue color on the background. It also supports data validity with the same funtionality as *Data Label*.

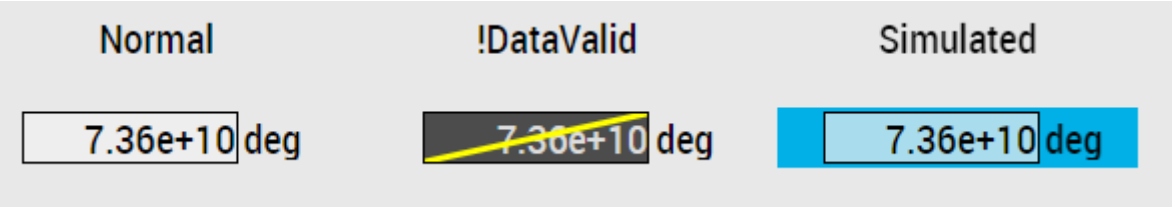


Fig. 17: Analog Sensor Control: Normal, !DataValid and Simulated visual information

Interface

Table 12: Analog Sensor Control Interface Table

Name	Group	Description
Value	Common	Sensor reading value to be displayed
Notation	Common	Shows numeric value **Normal** or wit **Scientific Notation**
Decimals	Common	Number of decimals to show in the value.
Signed	Common	Flag to enable plus (+) sign of the value to appear even when it's a positive value.
Simulated	Common	Flag show a light blue background on this controller to provide visual information that the sensor is in simulation.
DataValid	Common	Flag for data validity. It works as *Data Label*.
Unit	Common	Unit Text to be shown next to the value label.
Unit text box Width	Common	Percentage (%) of the horizontal space of the total control used to show the units of the value.
Alarm Raised	*Event Box*	Alarm Raised State (Active or Not)
Alarm Confirmed	*Event Box*	Alarm Confirmed State (Active or Not)
Warning Raised	*Event Box*	Warning Raised State (Active or Not)
Warning Confirmed	*Event Box*	Warning Confirmed State (Active or Not)
TextFontSize	Text	Size of the font for the text displayed by **Value** attribute
TextHorizontalAlign-ment	Text	Horizontal alignment inside the heater symbol for the text displayed by **Value** attribute
TextVerticalAlign-ment	Text	Vertical alignment inside the heater symbol for the text displayed by **Value** attribute

3.5 Sequences

Sequences are specialized controls that have a different implementation and use case. They are explained in detail at *Sequences*.

4 Sequences

Cryogenic application requires sequences for certain procedures (i.e. Evacuation). These sequences can be represented with a subset of flowchart *Blocks*: **Process**, **Decision** and **Terminator**. Cryo HMI Framework Controls implements this subset and joins them all together with a specialized *Sequence Container* that actually interfaces with the PLC code.

4.1 Blocks

Blocks are very simple controls with no important processing code inside of them. There are 3 blocks implemented, they all have the same interface, but their symbol and meaning are different:

- **Process**: Also called activity. Once finished it goes to the next step.
- **Decision**: Conditional operation that determines which one of the two paths the sequence will take.
- **Terminator**: Indicates the beginning and ending of a sequence.

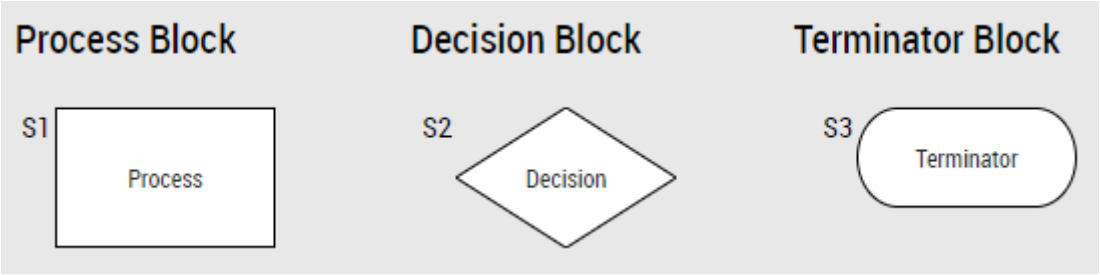


Fig. 1: The 3 types of Blocks

All of them can be in any of 3 different states: **Idle**, **Active** and **Finished**

- **Idle**: The sequence has not yet reached this step. The box has a white background.
- **Active**: The sequence is currently executing this step. The box background is animated between a white and a green background.
- **Finished**: The step has already been finished. The box has a green background.

For further information about Flowchart and its Blocks check its wiki page[4]

[4] https://en.wikipedia.org/wiki/Flowchart

Blocks Interface

Table 1: Common Blocks Interface Table

Name	Group	Description
Value	Common	String or Value that it's being shown inside the block symbol.
BlockId	Common	Unique number that represents the ID of this block, no other block should have the same value. The number is shown on the upper left side of the Framework Control as S<BlockId>.
State	Common	INT value that represents the state of the device: Idle (0), Active (1) and Finished (2).
On Fill Color	Colors	Symbol Fill Color when it's Finished
Off Fill Color	Colors	Symbol Fill Color when it's Idle

4.2 Sequence Container

As the name suggests, Sequence Container Framework Control inherits the TE2000 Container[5] capabilities.

As a plain container, it can group any child control. All child controls added are defined relative to the position of the container.

Sequence Containers also allow the user to interface with the PLC sequence by using a custom json-schema named **Sequence Object** . Check *Sequence Container Interface* for more information.

Note: There is a function block in CryoLib FB_CRYO_HMI_SEQUENCE that can interface directly with Sequence Container, but this is not covered in this manual.

The container will accept any type of control inside of it, but will only interact with **Process**, **Decision** and **Terminator** and only if they are direct childs of the container. Please check the section *Example* for more information.

Interface

Sequence Container has a specially defined Object as interface called *Sequence*. The interface is defined by a json-schema:

```
{
  "$schema": "http://json-schema.org/draft-04/schema",
```

(continues on next page)

[5] https://infosys.beckhoff.com/english.php?content=../content/1033/te2000_tc3_hmi_engineering/3845319947.html

(continued from previous page)

```
  "definitions": {
    "TcHmi.Controls.CryoFramework.SequenceContainer": {
      "type": "object",
      "frameworkInstanceOf": "TcHmi.Controls.System.TcHmiControl",
      "frameworkControlType": "SequenceContainer",
      "frameworkControlNamespace": "TcHmi.Controls.CryoFramework"
    },
    "TcHmi.Controls.CryoFramework.Sequence": {
      "properties": {
        "arSequence": {
          "items": {
            "type": "integer"
          },
          "type": "array"
        },
        "bIsFinished": {
          "type": "boolean"
        }
      },
      "type": "object",
      "required":[
        "arSequence",
        "bIsFinished"
      ]
    }
  }
}
```

In summary, the Object *Sequence* is a structure that contains two attributes:

- arSequence: Array of integer (BlockIds) values.
- bIsFinished: Flag that signals if the sequence is still running.

The Sequence Object isthe only interface Object that needs to be linked with the PLC.

Table 2: Sequence Container Interface Table

Name	Group	Description
Sequence.arSequence	Common	Array of numbers indicating Finished blocks. The last value of the array is the active block.
Sequence.bFinished	Common	If this value is true, the current active block is marked as finished.

4.3 Blocks and Container Logic

Every time the container gets an update of arSequence and bIsFinished, the following procedure is called:

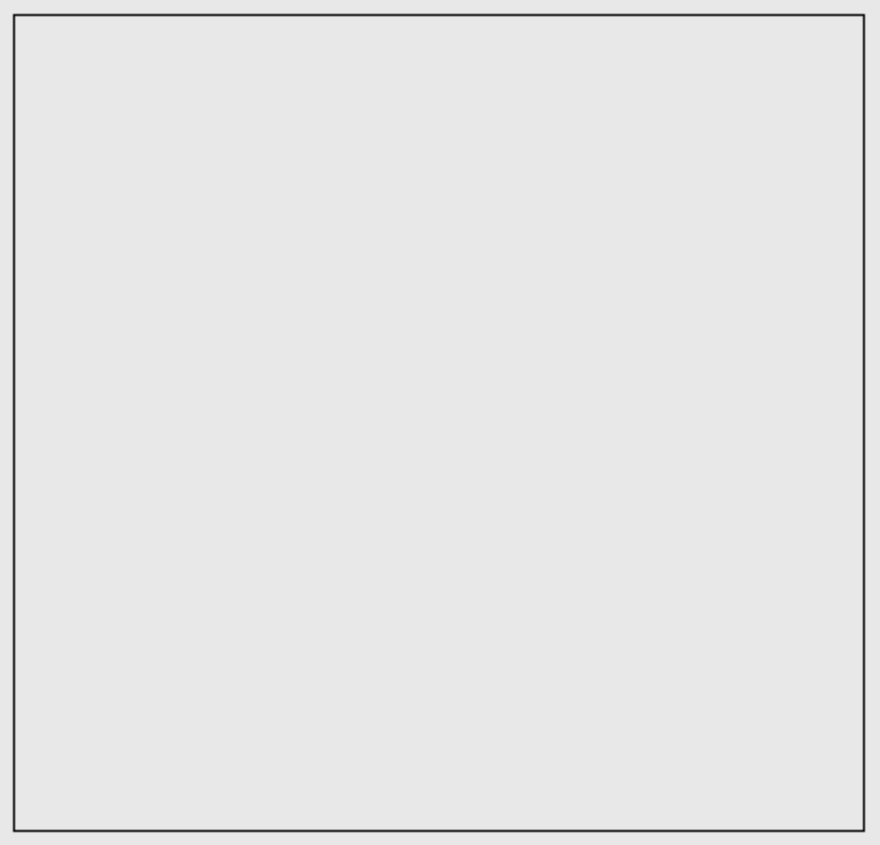
```
## WARNING This is just a small representation of the actual code.

## sequenceBlocks is a dictionary of blocks with block_id as key

def SequenceUpdated(self, ar_sequence: List, is_finished: bool):
    ## All elements of sequence_blocks are put to Idle state
    for block_id in self.sequence_blocks:
        self.sequence_blocks[block_id].setState(State.IDLE)

    ## Take last value, which corresponds to the current processed block
    last_value = ar_sequence.pop()
    if is_finished:
        self.sequence_blocks[last_value].setState(State.FINISHED)
    else:
        self.sequence_blocks[last_value].setState(State.ACTIVE)

    ## Put all other valid blocks from the sequence array as Finished
    for block_id in ar_sequence:
        if block_id in self.sequence_blocks:
            self.sequence_blocks[block_id].setState(State.FINISHED)
        elif block_id > 0:
            ## Error logged to browser console
            error("SequenceID " + block_id + "not present in HMI sequence!")
```

4.4 Example

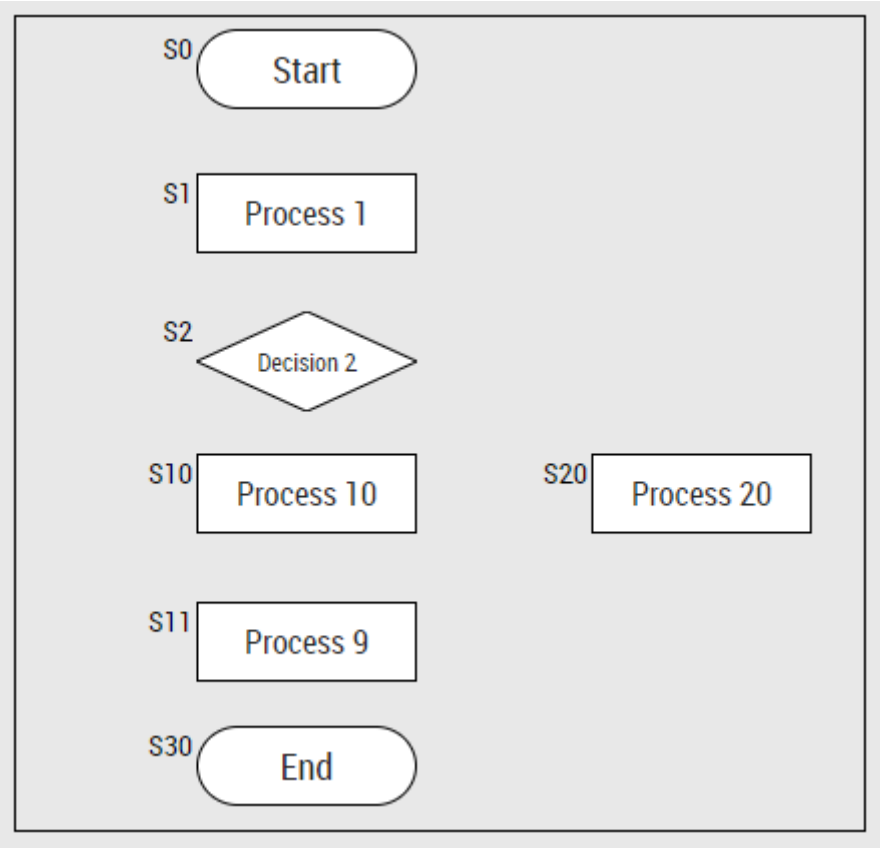
Let's create a simple sequence for testing and understanding Sequences.

The Setup

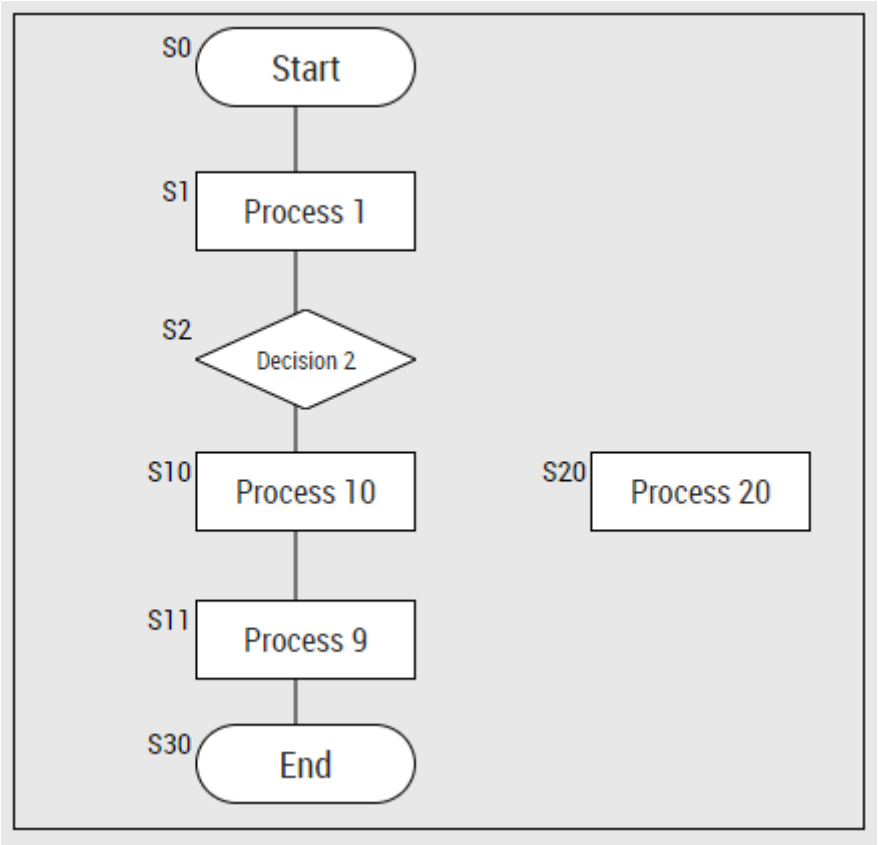
1. Add a Sequence Container, in this example, we also add borders so it's easy to see its limits.



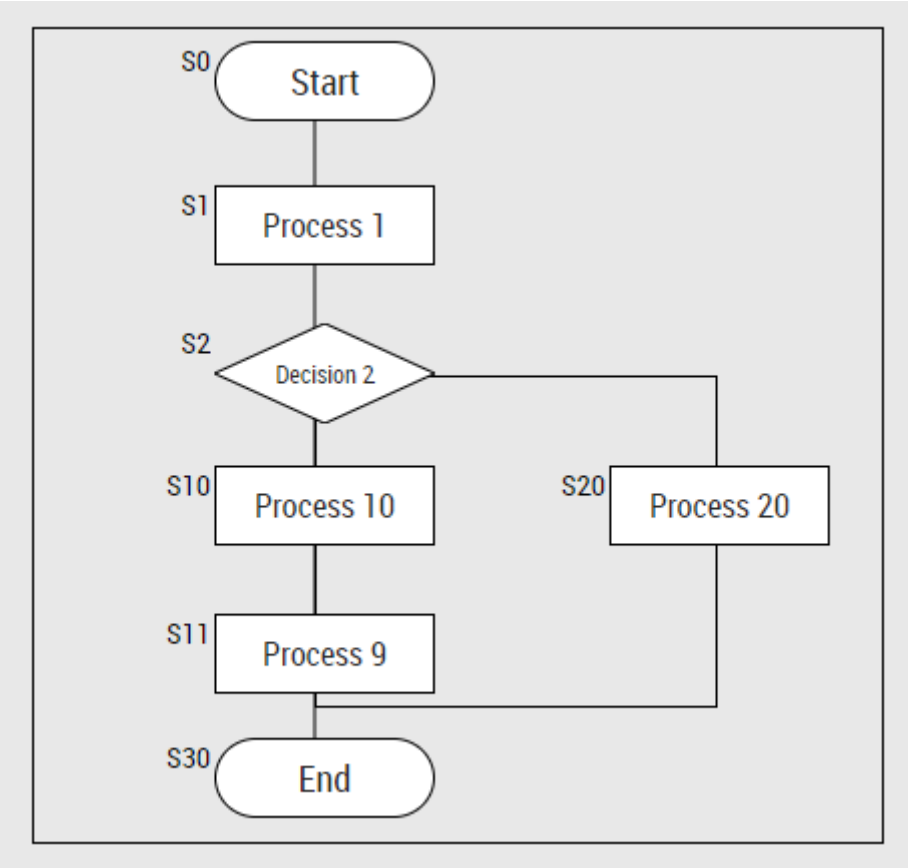
2. Add a Sequence Blocks and give them unique **BlockID**, also add some text on their **Value**



3. Use a simple line to connect all the vertical Blocks $[0,\ 1,\ 2,\ 10,\ 11,\ 30]$.



4. Add a rectangle to join Block S20 to the Rest of the diagram.



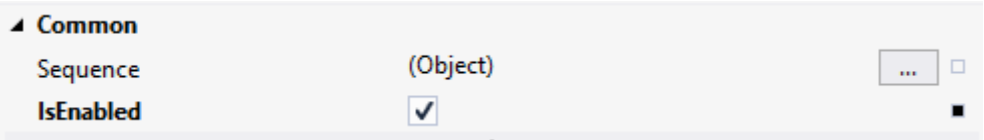
After following these steps, the setup is done.

Note: Steps 3 and 4 are only for cosmetic reasons, the line and the rectangle will not affect in any way the sequence behavior.

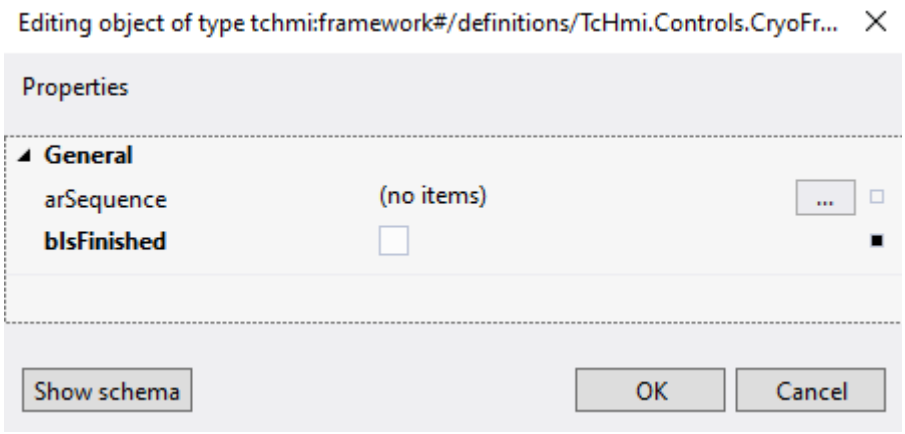
Testing a correct Sequence

Now that the sequence done, we will modify its **Sequence** attribute to see how it affects it.

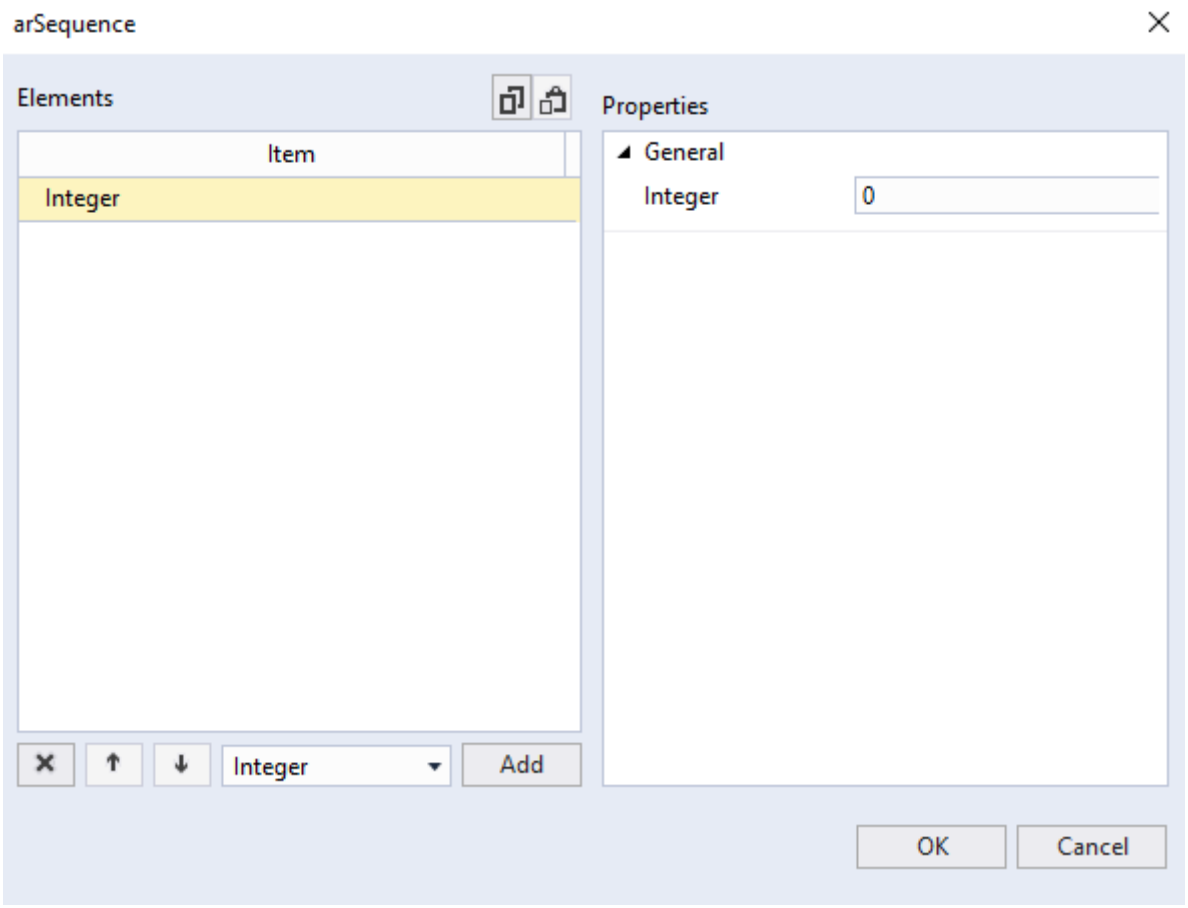
1. With the sequence container selected, click the attribute **Common** → **Sequence** three doted box [...].



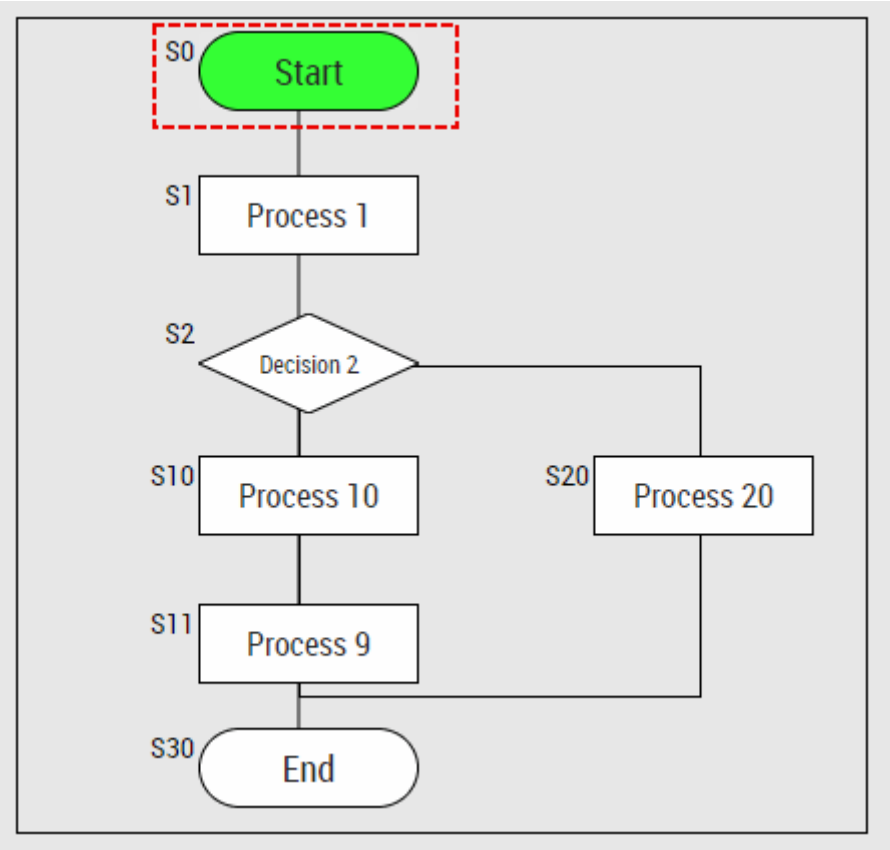
2. A new box will appear, select bIsFinished and then deselect it (this is to avoid an error)



3. Then press on the three doted box [...] on arSequence and a new menu will appear.
4. This menu is to edit arSequence array. Press Add button and a new value will appear, make sure the value is 0.

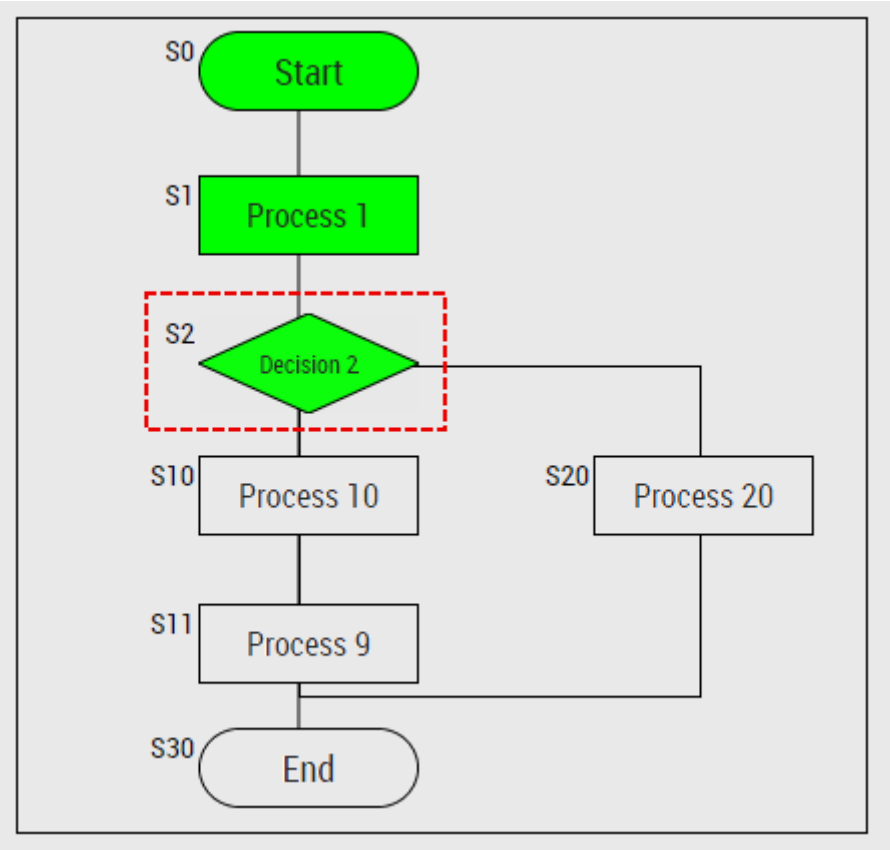


5. Close both modal windows and you should see how it affects the block with BlockId = 0. The block will blink between white and green colors signaling the user that this is the current step in the sequence.



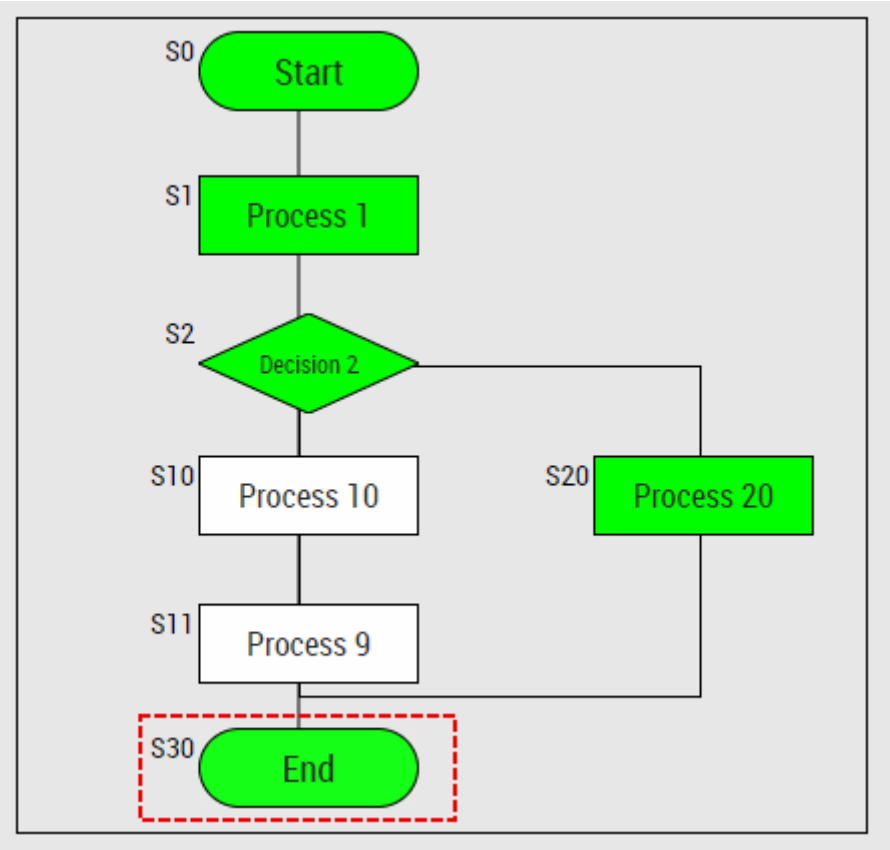
Note: The red box with dashed outline indicates that the block background is blinking between white and green.

6. Repeat step 3, 4 and 6. But this time add $[1, 2]$ to the array. Make sure 2 is the last value of the array (at the end).



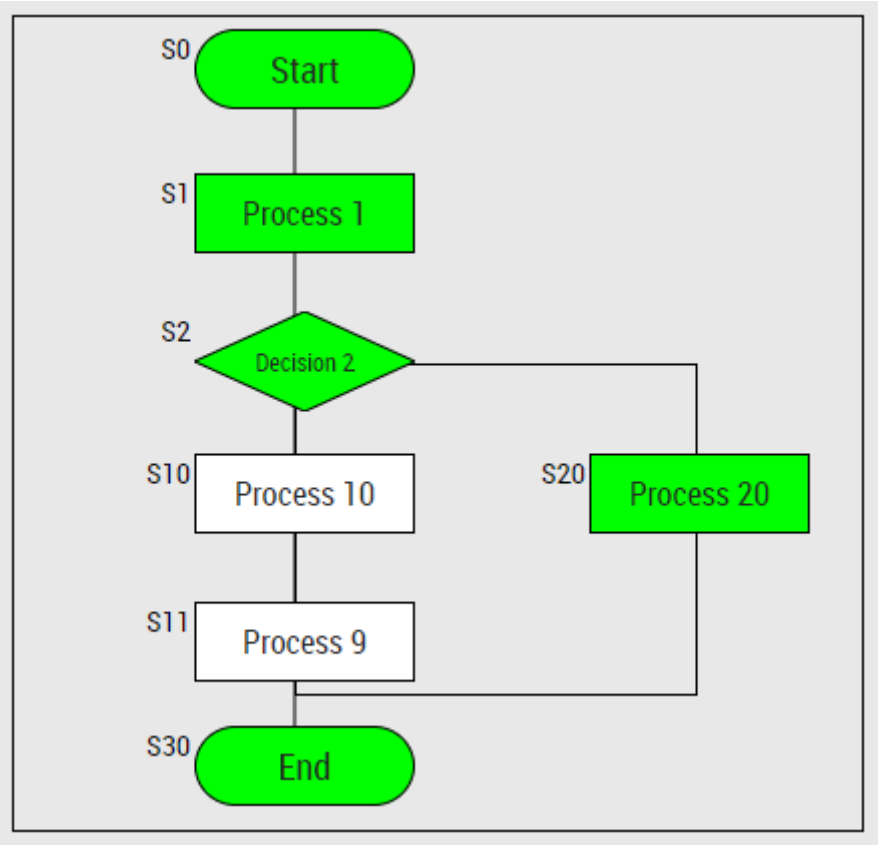
Note: The red box with dashed outline indicates that the block background is blinking between white and green.

7. Repeat step 3, 4 and 6. But this time add $[10, 30]$ to the array. Make sure 30 is the last value of the array (at the end). Notice how the blinking block is always the last one on the list, in this case, Block 30



Note: The red box with dashed outline indicates that the block background is blinking between white and green.

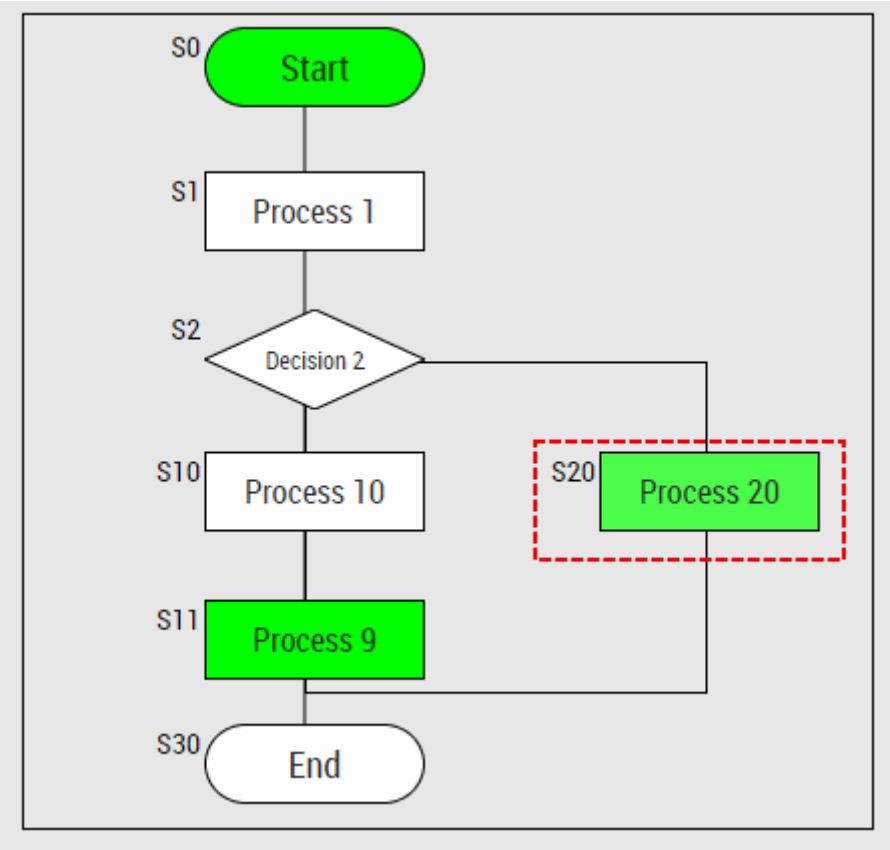
8. Open again the Sequence Object (as Step 1), this time, only check the value bIsFinished. Now Block S30 will stop blinking



This concludes the example, this should clear more or less how Sequences work.

Testing an incorrect Sequence

1. Follow the steps of *Testing a correct sequence* until step 4.
2. Add into arSequence the values $[9, 20]$.



Note: The red box with dashed outline indicates that the block background is blinking between white and green.

3. Notice how Block S9 is green and S20 is blinking.

Warning: As shown in the example, Sequence Blocks and Container are only a meant to show the state of a Sequence of a PLC. No logic is present to make sure that the data is consistent, so if an error like this appear could mean that the PLC sequence is wrong, or the BlockId between the PLC and the HMI seuquence are not correctly aligned.