

4. Graphic languages

The graphic languages defined in this standard are LD (Ladder Diagram) and FBD (Function Block Diagram). The sequential function chart (SFC) elements defined in 2.6 can be used in conjunction with either of these languages.

4.1 Common elements

The elements defined in this clause apply to both the graphic languages in the Standard, that is, LD (Ladder Diagram) and FBD (Function Block Diagram), and to the graphic representation of sequential function chart (SFC) elements.

4.1.1 Representation of lines and blocks

The graphic language elements defined in this clause are drawn with line elements using characters from the ISO/IEC 646 character set, or using graphic or semigraphic elements, as shown in table 57.

Lines can be extended by the use of *connectors* as shown in table 57. No storage of data or association with data elements shall be associated with the use of connectors; hence, to avoid ambiguity, it shall be an *error* if the identifier used as a connector label is the same as the name of another named element within the same program organization unit.

4.1.2 Direction of flow in networks

A *network* is defined as a maximal set of interconnected graphic elements, excluding the left and right rails in the case of networks in the LD language defined in 4.2. Provision shall be made to associate with each network or group of networks in a graphic language a *network label* delimited on the right by a colon (:). This label shall have the form of an identifier or an unsigned decimal integer as defined in clause 2 of this Part. The *scope* of a network and its label shall be *local* to the program organization unit in which the network is located. Examples of networks and network labels are shown in annex F.

Graphic languages are used to represent the flow of a conceptual quantity through one or more networks representing a control plan, that is:

- "Power flow", analogous to the flow of electric power in an electromechanical relay system, typically used in relay ladder diagrams;
- "Signal flow", analogous to the flow of signals between elements of a signal processing system, typically used in function block diagrams;
- "Activity flow", analogous to the flow of control between elements of an organization, or between the steps of an electromechanical sequencer, typically used in sequential function charts.

The appropriate conceptual quantity shall flow along lines between elements of a network according to the following rules:

- 1) Power flow in the LD language shall be from left to right.
- 2) Signal flow in the FBD language shall be from the output (right-hand) side of a function or function block to the input (left-hand) side of the function or function block(s) so connected.
- 3) Activity flow between the SFC elements defined in 2.6 shall be from the bottom of a step through the appropriate transition to the top of the corresponding successor step(s).

Table 57 - Representation of lines and blocks

No.	Feature	Example
1 2	Horizontal lines: ISO / IEC 646 "minus" character Graphic or semigraphic	-----
3 4	Vertical lines: ISO / IEC 646 "vertical line" character Graphic or semigraphic	
5 6	Horizontal/vertical connection: ISO / IEC 646 "plus" character Graphic or semigraphic	<pre> -+--+ </pre>
7 8	Line crossings without connection: ISO / IEC 646 characters Graphic or semigraphic	<pre> -+--+ </pre>
9 10	Connected and non-connected corners: ISO / IEC 646 characters Graphic or semigraphic	<pre> -+--+ +--- -+--+ +--- </pre>
11 12	Blocks with connecting lines: ISO / IEC 646 characters Graphic or semigraphic	<pre> +-----+ --- --- --- --- +-----+ </pre>
13 14	Connectors using ISO / IEC 646 characters: Connector Continuation of a connected line Graphic or semigraphic connectors	<pre> ----->OTTO> >OTTO>----- </pre>

4.1.3 Evaluation of networks

The order in which networks and their elements are evaluated is not necessarily the same as the order in which they are labeled or displayed. Similarly, it is not necessary that all networks be evaluated before the evaluation of a given network can be repeated. However, when the body of a program organization unit consists of one or more networks, the results of network evaluation within said body shall be functionally equivalent to the observance of the following rules:

- 1) No element of a network shall be evaluated until the states of all of its inputs have been evaluated.
- 2) The evaluation of a network element shall not be complete until the states of all of its outputs have been evaluated.
- 3) The evaluation of a network is not complete until the outputs of all of its elements have been evaluated, even if the network contains one of the execution control elements defined in 4.1.4.
- 4) The order in which networks are evaluated shall conform to the provisions of 4.2.6 for the LD language and 4.3.3 for the FBD language.

A *feedback path* is said to exist in a network when the output of a function or function block is used as the input to a function or function block which precedes it in the network; the associated variable is called a *feedback variable*. For instance, the Boolean variable RUN is the feedback variable in the example shown in figure 23. A feedback variable can also be an output element of a function block data structure as defined in 2.5.2.

Feedback paths can be utilized in the graphic languages defined in 4.2 and 4.3, subject to the following rules:

- 1) Explicit loops such as the one shown in 23a shall only appear in the FBD language defined in 4.3.
- 2) It shall be possible for the user to define the order of execution of the elements in an explicit loop, for instance by selection of feedback variables to form an implicit loop as shown in figure 23b.
- 3) Feedback variables shall be initialized by one of the mechanisms defined in clause 2. The initial value shall be used during the first evaluation of the network.
- 4) Once the element with a feedback variable as output has been evaluated, the new value of the feedback variable shall be used until the next evaluation of the element.

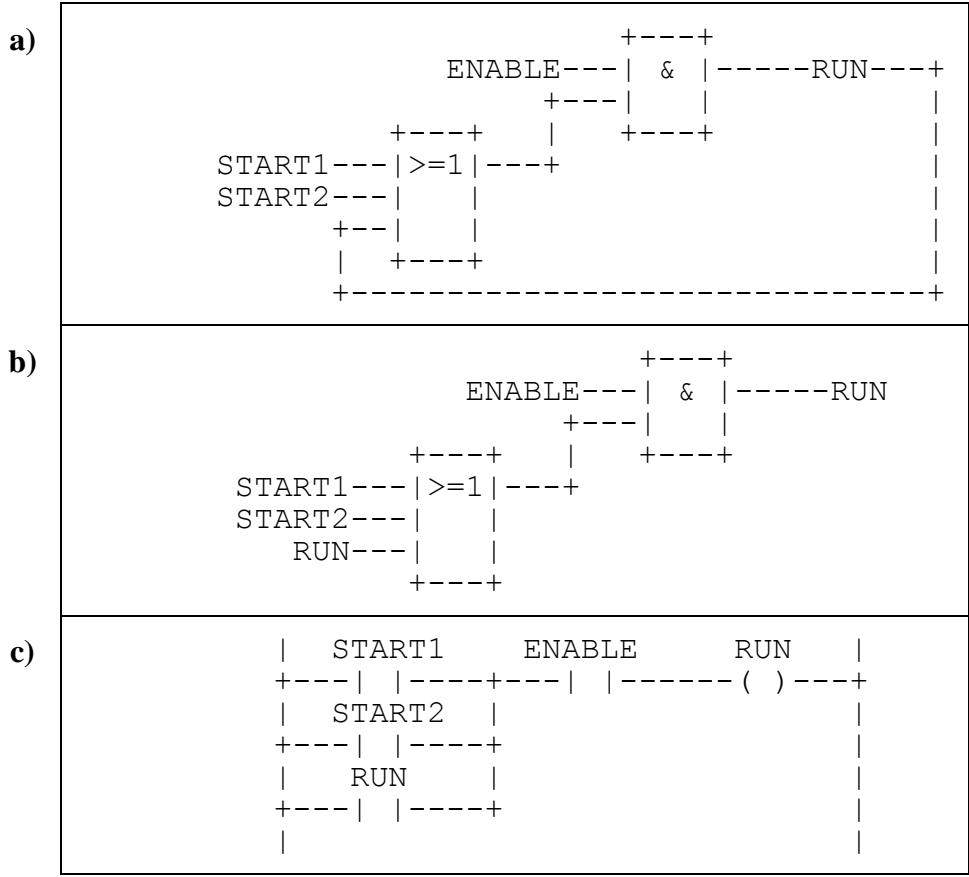


Figure 23 - Feedback path example
a) Explicit loop
b) Implicit loop
c) LD language equivalent

4.1.4 Execution control elements

Transfer of program control in the LD and FBD languages shall be represented by the graphical elements shown in table 58.

Jumps shall be shown by a Boolean signal line terminated in a double arrowhead.

The signal line for a jump condition shall originate at a Boolean variable, at a Boolean output of a function or function block, or on the power flow line of a ladder diagram.

A transfer of program control to the designated network label shall occur when the Boolean value of the signal line is 1 (TRUE); thus, the unconditional jump is a special case of the conditional jump.

The target of a jump shall be a network label within the program organization unit within which the jump occurs. If the jump occurs within an ACTION...END_ACTION construct, the target of the jump shall be within the same construct. Conditional returns shall be implemented using a RETURN construction as shown in table 58. Program execution shall be transferred back to the invoking entity when the Boolean input is 1 (TRUE), and shall continue in the normal fashion when the Boolean input is 0 (FALSE). Unconditional returns shall be provided by the physical end of the function or function block, or by a RETURN element connected to the left rail in the LD language, as shown in table 58.

Table 58 - Graphic execution control elements

No.	Symbol/Example	Explanation
1	<pre> 1----->>LABELA </pre>	<p>Unconditional Jump: FBD Language</p>
2	<pre> +----->>LABELA </pre>	<p>LD Language</p>
3	<pre> X----->>LABELB +-----+ %IX20--- & ---->>NEXT %MX50--- +-----+ NEXT: +-----+ %IX25--- >=1 ----%QX100 %MX60--- +-----+ </pre>	<p>Conditional Jump (FBD Language)</p> <p>Example: Jump Condition</p> <p>Jump Target</p>

4	<pre> X +- ----->>LABELB %IX20 %MX50 +--- ----- ---->>NEXT NEXT: %IX25 %QX100 +--- -----+----- ()----+ %MX60 +--- -----+ </pre>	<p>Conditional Jump (LD Language)</p> <p>Example: Jump Condition</p> <p>Jump Target</p>
5	<pre> X +-- ----<RETURN> </pre>	<p>Conditional Return: LD Language</p>
6	<pre> X---<RETURN> </pre>	<p>FBD Language</p>
7	<pre> END_FUNCTION END_FUNCTION_BLOCK +---<RETURN> </pre>	<p>Unconditional Return: from FUNCTION from FUNCTION_BLOCK Alternative representation in LD language</p>
8		

4.2 Language LD (Ladder Diagram)

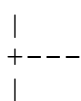
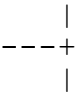
This subclause defines the LD language for ladder diagram programming of programmable controllers.

A LD program enables the programmable controller to test and modify data by means of standardized graphic symbols. These symbols are laid out in networks in a manner similar to a "rung" of a relay ladder logic diagram. LD networks are bounded on the left and right by *power rails*.

4.2.1 Power rails

As shown in table 59, LD network shall be delimited on the left by a vertical line known as the *left power rail*, and on the right by a vertical line known as the *right power rail*. The right power rail may be explicit or implied.

Table 59 - Power rails

No.	Symbol	Description
1		Left power rail (with attached horizontal link)
2		Right power rail (with attached horizontal link)

4.2.2 Link elements and states

As shown in table 60, link elements may be horizontal or vertical. The state of the link element shall be denoted "ON" or "OFF", corresponding to the literal Boolean values 1 or 0, respectively. The term *link state* shall be synonymous with the term *power flow*.

The state of the left rail shall be considered ON at all times.. No state is defined for the right rail.

A horizontal link element shall be indicated by a horizontal line. A horizontal link element transmits the state of the element on its immediate left to the element on its immediate right.

The vertical link element shall consist of a vertical line intersecting with one or more horizontal link elements on each side. The state of the vertical link shall represent the inclusive OR of the ON states of the horizontal links on its left side, that is, the state of the vertical link shall be:

- OFF if the states of all the attached horizontal links to its left are OFF;
- ON if the state of one or more of the attached horizontal links to its left is ON.

The state of the vertical link shall be copied to all of the attached horizontal links on its right. The state of the vertical link shall not be copied to any of the attached horizontal links on its left.

Table 60 - Link elements

No.	Symbol	Description
1	-----	Horizontal link
2	<pre> ---+--- ---+ +---- </pre>	<p>Vertical link (with attached horizontal links)</p>

4.2.3 Contacts

A *contact* is an element which imparts a state to the horizontal link on its right side which is equal to the Boolean AND of the state of the horizontal link at its left side with an appropriate function of an associated Boolean input, output, or memory variable. A contact does not modify the value of the associated Boolean variable. Standard contact symbols are given in table 61.

4.2.4 Coils

A *coil* copies the state of the link on its left to the link on its right without modification, and stores an appropriate function of the state or transition of the left link into the associated Boolean variable. Standard coil symbols are given in table 62.

4.2.5 Functions and function blocks

The representation of functions and function blocks in the LD language shall be as defined in clause 2 of this Part, with the following exceptions:

- 1) Actual parameter connections may optionally be shown by writing the appropriate data or variable outside the block adjacent to the formal parameter name on the inside.
- 2) At least one Boolean input and one Boolean output shall be shown on each block to allow for power flow through the block.

4.2.6 Order of network evaluation

Within a program organization unit written in LD, networks shall be evaluated in top to bottom order as they appear in the ladder diagram, except as this order is modified by the execution control elements defined in 4.1.4.

Table 61 - Contacts

Static contacts		
No	Symbol	Description
1	*** -- --	<i>Normally open contact</i> The state of the left link is copied to the right link if the state of the associated Boolean variable (indicated by "***") is ON. Otherwise, the state of the right link is OFF.
2	or *** -- ! ! --	
3	*** -- / --	<i>Normally closed contact</i> The state of the left link is copied to the right link if the state of the associated Boolean variable is OFF. Otherwise, the state of the right link is OFF.
4	or *** -- ! / ! --	
Transition-sensing contacts		
5	*** -- P --	<i>Positive transition-sensing contact</i> The state of the right link is ON from one evaluation of this element to the next when a transition of the associated variable from OFF to ON is sensed at the same time that the state of the left link is ON. The state of the right link shall be OFF at all other times.
6	or *** -- ! P ! --	
7	*** -- N --	<i>Negative transition-sensing contact</i> The state of the right link is ON from one evaluation of this element to the next when a transition of the associated variable from ON to OFF is sensed at the same time that the state of the left link is ON. The state of the right link shall be OFF at all other times.
8	or *** -- ! N ! --	
NOTE: As specified in 2.1.1, the exclamation mark "!" shall be used when a national character set does not support the vertical bar " ".		

Table 62 - Coils

Momentary coils		
No.	Symbol	Description
1	*** -- () --	<i>Coil</i> The state of the left link is copied to the associated Boolean variable and to the right link.
2	*** -- (/) --	<i>Negated coil</i> The state of the left link is copied to the right link. The inverse of the state of the left link is copied to the associated Boolean variable, that is, if the state of the left link is OFF, then the state of the associated variable is ON, and vice versa.
Latched Coils		
3	*** -- (S) --	<i>SET (latch) coil</i> The associated Boolean variable is set to the ON state when the left link is in the ON state, and remains set until reset by a RESET coil.
4	*** -- (R) --	<i>RESET (unlatch) coil</i> The associated Boolean variable is reset to the OFF state when the left link is in the ON state, and remains reset until set by a SET coil.
Retentive coils (see Note)		
5	*** ---- (M) ----	Retentive (Memory) coil
6	*** ---- (SM) ----	SET retentive (Memory) coil
7	*** ---- (RM) ----	RESET retentive (Memory) coil
Transition-sensing coils		
8	*** -- (P) --	<i>Positive transition-sensing coil</i> The state of the associated Boolean variable is ON from one evaluation of this element to the next when a transition of the left link from OFF to ON is sensed. The state of the left link is always copied to the right link.
9	*** -- (N) --	<i>Negative transition-sensing coil</i> The state of the associated Boolean variable is ON from one evaluation of this element to the next when a transition of the left link from ON to OFF is sensed. The state of the left link is always copied to the right link.

NOTE - The action of coils 5, 6, and 7 is identical to that of coils 1, 3, and 4, respectively, except that the associated Boolean variable is automatically declared to be in retentive memory without the explicit use of the VAR RETAIN declaration defined in 2.4.2.