WPSIOT

Logic Designer
Manual de primeiro uso

Placa WPSIOT V3.0 2025

Firmware v1.23

Do zero ao primeiro programa rodando

02/08/2026

Este manual leva você do momento em que tira a placa da caixa até ter um programa em ladder escrito por você, gravado na placa e sendo acompanhado ao vivo na tela.

Ele assume o cenário mais comum: **a placa já vem com o firmware instalado de fábrica e ainda não conhece nenhuma rede WiFi**. Você não precisa instalar programa nenhum no computador — o editor mora dentro da própria placa.

Reserve cerca de 40 minutos para a primeira vez.

Como ler este manual

Capítulo	O que você faz
1	Monta a bancada mínima (o que ligar em cada borne)
2	Liga a placa pela primeira vez e abre o editor
3	Coloca a placa na sua rede WiFi
4	Cria um programa do zero: partida com selo e temporizador
5	Grava o programa na placa
6	Fica online e acompanha a lógica executando
7	Referência de todas as instruções
8	Cria a sua primeira AOI (instrução própria)
9	Power fail: a placa não perde nada na falta de energia
10	Quando algo dá errado
11	Referência rápida

> **Convenção:** `código` indica um texto que aparece na tela ou que você digita. Caminhos de menu aparecem como **CLP ▾ → ⬇ Serial**.

1. A bancada mínima

1.1 O que você precisa

- A placa **WPSIOT V3.0 2025** com o firmware já gravado
- Um **cabo USB** (para alimentar e ver as mensagens de diagnóstico)
- Uma **fonte de 5 V** para acionar as entradas
- **Dois botões** (ou dois fios que você encosta) para as entradas
- Um computador com **Chrome ou Edge** e WiFi
- Opcional: uma lâmpada ou contator ligado a um relé, para ver a saída atuar

1.2 As ligações

A placa tem 8 entradas digitais opto-acopladas de **5 V** (bornes `IN0` a `IN7`) e 8 saídas a relé de contato seco (bornes `OUT0` a `OUT7`).

Para o exemplo deste manual, ligue:

Borne	O que ligar	Para quê
IN0	botão momentâneo ligando 5 V ao borne	botão **Liga**
IN1	botão momentâneo ligando 5 V ao borne	botão **Desliga**
IN6	**5 V fixo** (fio direto, sem botão)	**energia OK** — veja o aviso abaixo
OUT0	nada, ou uma lâmpada	contator (relé 1)
OUT1	nada, ou uma lâmpada	sinaleiro (relé 2)

> **⚠ O borne IN6 é obrigatório neste exemplo**
>
> O projeto de demonstração que já vem na placa usa o recurso de **power fail** (capítulo 9), configurado para vigiar a **entrada 7 — borne IN6**.
>
> Enquanto esse borne **não estiver com 5 V**, a placa entende que a energia externa caiu: **para a CPU, desliga todos os relés e bloqueia as operações online**. Se você já estiver conectado pelo editor, verá uma faixa laranja de `POWER FAULT`; se ainda não conectou, o sintoma é simplesmente a placa não executar nada.
>
> Então: **ligue 5 V no IN6 antes de qualquer coisa** — ou desligue o recurso (o caminho está em 9.3).

Regra de polaridade da placa, boa de memorizar:

> **5 V no borne = a entrada está acionada = a tag vale 1. Tag de saída em 1 = o relé fecha o contato.**

1.3 O mapa de bornes

Esta é a correspondência de fábrica entre os bornes e os nomes das tags no editor. Todo projeto novo já nasce com essas tags criadas e mapeadas.

Entradas — opto-acopladas, 5 V:

Borne	Tag	GPIO	Observação
IN0	`I1`	36	só-entrada
IN1	`I2`	39	só-entrada
IN2	`I3`	34	só-entrada
IN3	`I4`	35	só-entrada
IN4	`I5`	32	
IN5	`I6`	33	
IN6	`I7`	25	usado como *energia OK* no demo
IN7	`I8`	26	

Saídas — relés de contato seco, comum único, 5 A por contato / 10 A no total:

Borne	Tag	GPIO	Observação
OUT0	`Rele1`	19	
OUT1	`Rele2`	18	
OUT2	`Rele3`	17	
OUT3	`Rele4`	16	
OUT4	`Rele5`	4	
OUT5	`Rele6`	27	
OUT6	`Rele7`	12	pino de strapping — veja abaixo
OUT7	`Rele8`	13	

Além disso: **I²C** em SDA 21 / SCL 22, reservado para expansão.

> **Nota técnica sobre o Rele7 (GPIO12):** esse pino é *strapping* do ESP32 — nível alto durante o boot altera a tensão da flash. O firmware só escreve nas saídas **depois** do boot, então o uso normal é seguro; apenas não instale pull-up externo nesse ponto.

Você pode consultar essa tabela a qualquer momento dentro do editor: na árvore do projeto, dê **duplo clique** em `WPSIOT V3.0 2025`.

---

2. Primeira energização e o editor

2.1 Ligue a placa

Conecte o cabo USB. Os relés ficam todos **desligados** no boot — isso é proposital: um controlador que acabou de ligar não pode energizar carga antes de saber o que está fazendo.

Se quiser acompanhar (opcional, mas útil da primeira vez), abra o **Monitor Serial** do Arduino IDE ou qualquer terminal a **115200 bps**. Você verá algo assim:

```
=== WPSIOT PLC v1.23 — WPSIOT V3.0 2025 ===
[core] ArduinoJson 7.0.4 (empacotado)
[plc] programa restaurado: WPSIOT_Demo (2759 bytes)
[mdns] http://wpsiot.local
[mbs] servidor Modbus TCP na porta 502 (FC3, so leitura — instrucao MBSRV)
[plc] I/O e vigia: core 1 prio 21 | Modbus: prio 10 | visualizacao: prio 1
[net] AP: WPSIOT-PLC (senha wpsiot123) -> http://192.168.4.1
[ser] comandos: info | wifi SSID SENHA | run | stop | stats
```

A linha que interessa agora é a do **AP**.

> **Feche o Monitor Serial antes de gravar pela USB.** Só um programa por vez pode usar a porta serial. Se esquecer, a gravação falha com `porta ocupada`.

2.2 A placa cria a própria rede WiFi

Mesmo sem conhecer nenhuma rede, a placa **sempre** publica um ponto de acesso próprio:

Rede (SSID)	`WPSIOT-PLC`
Senha	`wpsiot123`
Endereço	`http://192.168.4.1`

Esse ponto de acesso **não desaparece** depois que você configurar a sua rede — ele é a porta dos fundos que sempre funciona.

2.3 Abra o editor

No computador, conecte-se à rede `WPSIOT-PLC` e abra o navegador em:

```
http://192.168.4.1
```

O editor **WPSIOT Logic Designer** carrega. Ele está inteiro dentro da placa — não há nada para instalar, e funciona sem internet.

Você verá:

- No alto, a marca **WPSIOT Logic Designer** com a versão do firmware embaixo

- A barra de menus: **Arquivo ▾**, **CLP ▾**, **Opções ▾**, os botões `Assemble` e `Cancelar edições`, e o indicador de modo à direita (`OFFLINE`)
- À esquerda, o **Controller Organizer** (a árvore do projeto)
- No centro, a área de lógica em ladder
- Acima dela, a **toolbox** com as instruções, dividida em abas

> **Dois jeitos de abrir o editor — e a diferença que importa**
>
> | Como você abriu | Dá para gravar pela USB? | Dá para ficar online? |
> |---|---|---|
> | `http://192.168.4.1` (servido pela placa) | **Não** | Sim |
> | Arquivo `wpsiot-logic-designer.html` no seu PC | Sim | Sim |
>
> A gravação pela USB usa a tecnologia *Web Serial*, que os navegadores só liberam para páginas locais ou seguras — nunca para uma página `http://` comum como a servida pela placa. Se você for gravar pela USB, abra o arquivo do editor que veio no kit (`editor\wpsiot-logic-designer.html`), com duplo clique, no Chrome ou Edge.
>
> **Neste manual vamos pela rede**, que é o caminho mais confortável.

---

3. Colocando a placa na sua rede

Enquanto a placa só tem o próprio ponto de acesso, você precisa se desconectar da sua rede para falar com ela. Vale a pena resolver isso já.

3.1 Conecte-se à placa primeiro

A configuração de WiFi é feita **através da placa** — então o editor precisa estar falando com ela. Ainda conectado ao `WPSIOT-PLC` e com o editor aberto em `http://192.168.4.1`:

CLP ▾ → □ Conectar na placa

O diálogo **Conectar na placa** já sugere o endereço certo (`192.168.4.1`). Clique **OK**. O indicador no alto muda para `ONLINE`.

> Se aparecer o diálogo `Tela ≠ CLP — programas diferentes`, escolha **Só monitorar** por enquanto — vamos gravar a nossa lógica no capítulo 5.

3.2 Peça a lista de redes

Agora sim: **CLP ▾ → □ WiFi**.

A placa varre o ambiente e abre o diálogo **Redes WiFi**, com as redes que *ela* enxergou — não as que o seu computador enxerga. Isso é de propósito: o que importa é o alcance da antena da placa.

Cada linha mostra a força do sinal, o nome e se é protegida:

```
▂▄▆  MinhaRede        ·  -52 dBm  □
▂▄   Vizinho          ·  -63 dBm  □
— outra rede (digitar o nome) —
```

> **A placa só enxerga redes de 2,4 GHz.** Se a sua rede é 5 GHz (ou o roteador está em modo exclusivo 5 GHz), ela não vai aparecer na lista. Use a banda de 2,4 GHz do roteador.

3.3 Conecte

Escolha a sua rede, digite a senha e clique em **Conectar**.

A senha vai direto para a memória permanente da placa (a NVS) — **ela não fica guardada no arquivo do projeto**, então você pode compartilhar o projeto sem expor a senha da sua rede.

Em alguns segundos aparece o diálogo **Placa na sua rede ✔**, com os dois endereços que passam a valer:

```
http://192.168.0.110      (o IP que o seu roteador deu)
http://wpsiot.local       (o nome, que funciona mesmo se o IP mudar)
```

Anote o IP. A partir de agora você pode **voltar para a sua rede normal** no computador e abrir o editor por qualquer um desses dois endereços — sem perder a internet.

3.4 Se der errado

O que apareceu	O que fazer
`WiFi falhou: nao conectou em 15 s`	Senha errada, ou rede em 5 GHz. Tente de novo.
A rede não aparece na lista	Clique em **Buscar de novo**; se persistir, use `— outra rede (digitar o nome) —`
`Sem resposta da placa em 25 s...`	Normal: a placa trocou de rede e derrubou a conexão junto. Abra `http://wpsiot.local`

A partir daqui a placa reconecta sozinha nessa rede toda vez que ligar.

4. Seu primeiro programa, do zero

Vamos escrever a lógica mais clássica de todas — a **partida com selo** — e acrescentar um temporizador.

O que ela faz:

1. Apertar o botão **Liga** (IN0) energiza o **contator** (Rele1)
2. O contator **se mantém ligado** depois que você solta o botão (o *selo*)
3. Apertar o botão **Desliga** (IN1) derruba tudo
4. Três segundos depois de ligar, acende um **sinaleiro** (Rele2)

4.1 Comece um projeto novo

Arquivo ▾ → Novo. O diálogo **Novo projeto** pede o **Nome do controller** (sugestão: `Meu_Primeiro`). Clique em **Criar**.

> O projeto novo já vem com **as 16 tags de I/O da placa criadas e mapeadas** (`I1`–`I8` nas entradas, `Rele1`–`Rele8` nas saídas) e uma MainRoutine vazia. É por isso que vamos usar essas tags diretamente: elas já apontam para os bornes certos.

A árvore fica assim:

```
▯ Controller Meu_Primeiro
  ▯ Controller Tags
  ▯ Tasks (1/4)
   ⟳ MainTask (contínua)
        ▯ MainProgram
           ▣ MainRoutine  [1]
  ▯ Data Types (UDT)
  ⚙ Add-On Instructions
     ⚙ Partida  [3]          ← a AOI de exemplo, que vem de fábrica
  ▯ I/O Configuration
     ▤ WPSIOT V3.0 2025
```

Clique em **MainRoutine** para abrir a área de lógica.

> **Entendendo a árvore em 30 segundos.** Uma **task** decide *quando* a lógica roda (a contínua roda o tempo todo; uma periódica roda a cada N milissegundos). Dentro dela há um **programa**, e dentro do programa as **rotinas**. A rotina marcada com ▣ é a *Main* — a única que executa sozinha; as outras (▢) só rodam quando chamadas por uma instrução `JSR`.

4.2 Descreva as tags que vai usar e crie o timer

Clique em **Controller Tags** na árvore. A tabela abre com estas colunas:

`Tag` · `Tipo` · `Valor (online)` · `Alias` · `Preset` · `Dim` · `Descrição`

As quatro tags de I/O que precisamos já existem. Preencha só a **Descrição** delas, para o programa ficar legível:

Tag	Alias (já vem)	Descrição a escrever
`I1`	`IN0`	Botão liga
`I2`	`IN1`	Botão desliga
`Rele1`	`OUT0`	Contator do motor
`Rele2`	`OUT1`	Sinaleiro de marcha

Falta só o temporizador. Clique em **+ Nova tag**: nasce uma linha chamada `NovaTag<n>`, do tipo BOOL. Edite-a:

Tag	Tipo	Alias	Preset	Descrição
`T_Sinaleiro`	TIMER	*(vazio)*	`3000`	Atraso do sinaleiro

Clique em **Aplicar**. O editor responde `Definições de tags aplicadas.`

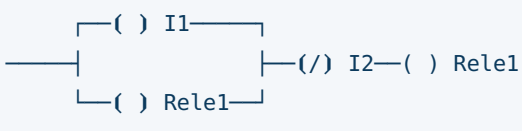
> **O que é o Alias.** É o que amarra a tag a um borne físico. `I1` com alias `IN0` significa: "esta tag vale 1 quando houver 5 V no borne IN0". Sem alias, a tag é apenas uma memória interna. É o mesmo conceito de alias do Logix 5000 — e a vantagem é que, se um dia você trocar de placa, muda o alias e a lógica continua idêntica. **Cada borne deve ter uma única tag com aquele alias.**
>
> **O que é o Preset.** Só vale para TIMER e COUNTER: é o valor que a contagem precisa alcançar. `3000` num TIMER significa 3000 milissegundos = 3 segundos.

4.3 O primeiro rung: a partida com selo

Volte para a aba **MainRoutine**. Você verá um rung vazio, numerado `0`.

> **Os pontos de inserção são invisíveis até você aproximar o mouse.** Passe o cursor sobre a linha do rung: onde uma instrução pode entrar aparece um retângulo cinza. Clicado, ele fica **verde** — é ali que a próxima instrução vai nascer.

Vamos montar isto:



Em palavras: "*se o botão de liga estiver apertado ou o contator já estiver ligado, e o botão de desliga não estiver apertado, então mantenha o contator ligado.*"

Primeiro, a linha em série:

1. **Clique no ponto de inserção** do rung 0 (o retângulo que aparece ao passar o mouse). Ele fica verde. *(Se você clicar numa instrução sem escolher o ponto antes, o editor avisa: `Selecione um ponto de inserção (quadradinho cinza) primeiro.`)*

2. Na toolbox, aba **Bit**, clique em **( ) XIC** . Um contato normalmente aberto aparece no rung, com o operando `?`.
3. **Clique no** `?` dentro do contato e digite `I1`. Pressione **Enter**. *(Também dá para arrastar a instrução da toolbox direto para o ponto.)*
4. Clique no ponto **à direita** do contato e insira um **(/) XIO** (contato normalmente fechado). Operando: `I2`.
5. Clique no ponto seguinte e insira uma **( ) OTE** (bobina). Operando: `Rele1`.

Agora o selo — o ramo em paralelo. No editor, como no Logix, você não "agrupa" instruções num branch: você **insere o branch e depois arrasta o rail direito** para incluir o que deve ficar dentro dele.

1. Clique no ponto de inserção **à esquerda do contato** `I1` (o primeiro do rung) e clique em **Branch** na toolbox. Nasce um branch vazio, com duas legs, **antes** do contato.
2. **Clique no rail direito do branch** (a barra vertical do lado direito dele) para selecioná-lo. O editor confirma: `Branch selecionado — arraste o rail direito (ou use ◀ Rail / Rail ▶) para incluir instruções.`
3. Clique em **Rail ▶** (ou pressione **Alt+→**). O contato `I1` é engolido para dentro do branch: `1 elemento(s) incluído(s) no branch.`
4. Clique no ponto vazio da **segunda leg** do branch e insira um **( ) XIC** com o operando `Rele1`.

Pronto: o rung 0 está completo. Em *neutral text* — a notação textual do Logix, que o editor aceita na barra do rodapé — ele é:

```
[XIC(I1),XIC(Rele1)]XIO(I2)OTE(Rele1);
```

> **Atalho que economiza cliques:** em vez de montar o rung à mão, digite essa linha na barra **Neutral text**, no rodapé, e clique em **Aplicar ao rung**. Vale para qualquer rung deste manual.
>
> **Por que o selo funciona.** No primeiro instante, `I1` fecha o caminho e a bobina liga `Rele1`. No scan seguinte, mesmo que você solte o botão, o contato `XIC(Rele1)` — que agora vale 1 — mantém o caminho. É a própria saída se segurando: por isso "selo".

4.4 O segundo e o terceiro rung: o temporizador

Adicione um rung novo: selecione o rung 0 e pressione **Ctrl+R**, ou use o botão **Rung** da toolbox.

Rung 1 — dispara a contagem enquanto o motor estiver ligado:

```
———( ) Rele1———[TON  T_Sinaleiro  PRE 3000]
```

1. Insira um **XIC** com operando `Rele1`
2. Na toolbox, aba **Timer/Counter**, insira um **TON**
3. No operando do TON, digite `T_Sinaleiro`

O bloco mostra `PRE 3000` e `ACC 0`, além das lâmpadas `EN`, `TT` e `DN`.

Rung 2 — usa o resultado do timer:

```
———( ) T_Sinaleiro.DN———( ) Rele2
```

1. Novo rung, um **XIC** com operando `T_Sinaleiro.DN` (*`.DN` é o bit "terminou" do timer — veja o capítulo 7)*
2. Uma **OTE** com operando `Rele2`

Em neutral text, os três rungs do programa:

```
[XIC(I1),XIC(Rele1)]XIO(I2)OTE(Rele1);
XIC(Rele1)TON(T_Sinaleiro);
XIC(T_Sinaleiro.DN)OTE(Rele2);
```

4.5 Comente o que você fez

Comentários não custam nada — eles nem viajam para a placa — e salvam você daqui a seis meses.

Clique no espaço logo acima do rung (com o rung selecionado aparece um `// comentário…` cinza) e digite. **Enter** confirma, **Esc** cancela.

Sugestão:

- Rung 0: `Partida com selo – I1 sela em Rele1, I2 derruba`
- Rung 1: `Conta 3 s a partir da marcha`
- Rung 2: `Sinaleiro acende quando o timer termina`

> Comentários podem ser editados **também com a placa online e rodando** — como eles não fazem parte do programa compilado, editá-los não grava nada na placa nem perturba a máquina.

4.6 Teste na tela, antes da placa

Você não precisa de placa nenhuma para ver isso funcionar.

CLP ▾ → ▶ Simular. O indicador muda para `SIM • RUN` e a lógica passa a ser executada pelo próprio navegador — com a mesma semântica do firmware.

Para simular o aperto do botão: **clique com o botão direito** sobre o contato `I1` (ou selecione-o e aperte a tecla **T**). Como essa tag é alias de uma entrada física, o editor alterna o **borne**, e responde:

```
Borne IN0 (I1) → 5 V
```

Você verá:

- O caminho de energia ficar **verde** através do rung 0
- A bobina `Rele1` acender
- O `ACC` do timer subir até 3000, e o `DN` acender
- O `Rele2` acender três segundos depois

Alterne o borne de novo para "soltar" o botão e confirme que o selo segura. Depois teste o `I2`.

Quando terminar, **CLP ▾ → ■ Parar**.

4.7 Salve o projeto

Arquivo ▾ → Salvar. O arquivo vai para a pasta de downloads do navegador.

> **Salvar guarda tudo:** a lógica, as tags, os comentários, as descrições, as cores que você escolheu e até os valores das tags. É o seu projeto; a placa recebe só a parte compilada.

---

5. Gravando o programa na placa

5.1 Grave pela rede

Com a placa na sua rede (capítulo 3), abra o editor pelo endereço dela (`http://wpsiot.local` ou o IP) e conecte:

CLP ▾ → □ Conectar na placa → OK

O indicador de modo muda para `ONLINE` e três pontinhos passam a correr ao lado — é o fluxo de dados entre a tela e a placa. Se os pontinhos congelarem em vermelho, a comunicação morreu (mais em 10.4).

> **Neste momento aparece o diálogo `Tela ≠ CLP — programas diferentes`.** É esperado: a placa ainda está com o programa de demonstração de fábrica, e você acabou de escrever outro. Escolha **⬇ Gravar no CLP** — a placa passa a executar a sua lógica.
>
> As outras opções existem para o dia em que você chegar numa máquina em operação: **⬆ Trazer do CLP para a tela** substitui o que está na tela pelo que a máquina está executando, e **Só monitorar** deixa os dois como estão.

Se o diálogo não apareceu, grave você mesmo: **CLP ▾ → ⬇ Gravar na placa**.

A confirmação é explícita:

```
✔ Gravado no CLP — a lógica da tela é a que está executando.
```

Aperte os botões da bancada: os relés devem responder exatamente como responderam na simulação.

5.2 Ou grave pela USB

Alternativa, útil quando a placa ainda não está na rede.

1. Feche as abas do editor servidas pela placa e o Monitor Serial
2. Abra o **arquivo local** do editor (`editor\wpsiot-logic-designer.html`) com duplo clique, no Chrome ou Edge
3. **CLP ▾ → ⬇ Serial**
4. Escolha a porta COM da placa na janela do navegador

Ao final aparece o diálogo **Download concluído ✔**, que já mostra os endereços para você ficar online depois.

5.3 O que exatamente foi gravado

Vai para a placa	Fica só no seu projeto
A lógica compilada (rungs e instruções)	Comentários de rung
As tags, tipos, alias e presets	Descrições das tags
Os valores das tags (no download completo)	Nomes de rotinas e das AOIs
A configuração de MSG, MBSRV e power fail	Cores, larguras, layout

Isso não é uma limitação — é economia de memória num microcontrolador, e é por isso que comentário pode ser editado online sem perturbar nada.

> **O programa fica gravado na flash.** Pode desligar a placa da tomada: ao voltar, ela restaura o programa **e o modo em que estava** (rodando ou parado) sozinha, sem você gravar de novo.

6. Online: acompanhando a lógica ao vivo

Com o `ONLINE` no alto da tela, o que você vê **é** o que a placa está executando: os caminhos verdes, os valores, o `ACC` do timer subindo.

6.1 Os comandos do controlador

No menu **CLP ▾** aparecem, só quando online:

- ▶ **RUN do CLP** — a placa passa a varrer a lógica
- ⏹ **STOP do CLP** — a varredura para e **as saídas caem**

O STOP pede confirmação, com o aviso na cara:

> **A varredura vai PARAR e as saídas caem.** Os relés desligam imediatamente. Se a placa estiver ligada em alguma coisa que se move, confira antes.

Com o CLP parado, uma faixa aparece no alto do ladder:

> ⏹ **CLP em PROG** — a varredura está parada e as **saídas estão DESENERGIZADAS no ferro**. Os valores na tela são o **último estado da lógica**, não o que está nos relés.

Essa distinção é importante: **em PROG uma bobina pode aparecer verde na tela e o relé estar desligado na placa**. A tela mostra o último estado calculado; o relé obedece à regra de segurança, que é cair.

6.2 Forçando valores para testar

Como as suas entradas físicas estão nos bornes, o jeito honesto de testar é apertar os botões de verdade. Mas dá para escrever valores pela tela:

- **Botão direito num contato ou bobina booleana** → alterna na hora
- **Duplo clique** numa instrução com valor numérico → abre o campo para digitar
- Na tabela de tags, **clique na coluna** `Valor (online)` e digite

Se a tag for alias de uma entrada, o editor escreve no **borne simulado** e avisa `Borne IN0 (I1) → 5 V`. Se a lógica escrever nessa tag todo scan (uma OTE, por exemplo), o valor que você forçou é sobrescrito no próximo scan — e o editor avisa isso também.

6.3 Editando com a máquina rodando

Você pode alterar a lógica sem parar o processo — como no Logix 5000:

1. **Duplo clique no início da linha** (na margem esquerda) do rung que quer mudar. Nasce uma **cópia editável**, marcada com `i` e fundo azul; o original fica marcado com `r` e **continua executando**.
2. Edite a cópia à vontade — ela não executa.
3. Clique em **Assemble**. A cópia vira o rung definitivo, o original some, e a alteração é gravada na placa automaticamente.

Se mudar de ideia antes, **Cancelar edições** descarta a cópia.

Marcador	Significa
`i` (fundo azul)	cópia editável, **não executa e não é gravada** enquanto não houver Assemble
`r` (fundo cinza)	o original, executando
`d` (fundo rosa)	marcado para excluir no próximo Assemble

> **A armadilha clássica:** editar a cópia `i` e ir embora sem Assemble. O rung pendente **não é enviado para a placa** — e a sua alteração parece ter sumido quando você olha o programa online. O editor avisa disso ao gravar; leia o aviso.

6.4 Download × edição online

São duas coisas diferentes, de propósito:

	O que faz
⬇ Gravar na placa (download)	Leva a lógica **e os valores das tags** da tela. É "a placa passa a ser exatamente o que está na tela".
Assemble (edição online)	Leva só a lógica alterada. **Não toca nos valores** — timers continuam contando, contadores mantêm a contagem, o processo não reinicia.

Por isso, com a máquina rodando, prefira sempre a edição online.

7. Todas as instruções

A toolbox agrupa as instruções em abas. **Favoritos** reúne as dez mais usadas; as demais abas seguem a categoria.

Em qualquer instrução: clique num ponto de inserção e clique na instrução, ou arraste-a até o rung. Para preencher um operando, clique sobre ele e digite — **Enter** confirma, **Esc** cancela. Se a tag não existir, o editor oferece criá-la.

7.1 Bit — a base de tudo

Estas seis são 90% de qualquer lógica de campo.

( ) XIC — *Examine If Closed*

`XIC(tag)` — deixa a energia passar **quando o bit vale 1**.

É o "contato normalmente aberto". Note que ele examina o **valor da tag**, não o estado de um botão físico: se a tag for alias de uma entrada ligada a um contato NF (normalmente fechado) do painel, o comportamento se inverte na prática — cuidado clássico de chão de fábrica.

(/) XIO — *Examine If Open*

`XIO(tag)` — deixa a energia passar **quando o bit vale 0**.

O "contato normalmente fechado". Usado para paradas, intertravamentos e condições de bloqueio. No exemplo do capítulo 4 é o `I2`: enquanto ele **não** está apertado, o caminho segue.

( ) OTE — *Output Energize*

`OTE(tag)` — a tag recebe o estado do rung, **todo scan**.

Rung verdadeiro → tag em 1. Rung falso → tag em **0**. Ele espelha, não memoriza.

> **Regra de ouro:** só uma OTE por tag no programa inteiro. Se duas OTE escrevem a mesma tag, a de baixo vence e a de cima não faz nada — e isso quase sempre é bug. Para conferir quem escreve numa tag, use o □ (cross reference) na linha dela, na tabela de tags: as ocorrências que escrevem aparecem marcadas.

(L) OTL — *Output Latch*

`OTL(tag)` — **só liga**. Rung verdadeiro liga a tag; rung falso **não faz nada**. A tag fica em 1 até alguém desligá-la.

(U) OTU — *Output Unlatch*

`OTU(tag)` — **só desliga**. É o par obrigatório do OTL.

> **A boa prática que evita sustos:** trave um **bit interno** com OTL/OTU e leve esse bit à saída com uma OTE. Assim, quando o controlador for para PROG e voltar para RUN, a saída obedece ao programa em vez de reassumir sozinha.

[↑] ONS — *One Shot*

`ONS(tag)` — deixa a energia passar por **um único scan**, na borda de subida do rung.

Use quando algo deve acontecer *uma vez* por acionamento: incrementar um contador, disparar uma mensagem, registrar um evento.

> O operando é apenas o **nome de referência** dessa instância do ONS: o armazenamento do estado anterior é interno e individual de cada ONS — a tag em si não é lida nem escrita pela instrução.

7.2 Timer/Counter — tempo e contagem

Timers contam **tempo de verdade** (milissegundos), não scans. Todos têm:

Membro	O que é
`.PRE`	o valor a atingir, em ms (o *preset*)
`.ACC`	o valor acumulado agora
`.EN`	*Enable* — o rung está verdadeiro
`.TT`	*Timing* — está contando neste momento
`.DN`	*Done* — chegou no preset

Você endereça esses bits com ponto: `T_Sinaleiro.DN`, `T_Sinaleiro.ACC`.

TON — *Timer On Delay*

`TON(timer)` — começa a contar quando o rung fica verdadeiro. Ao chegar no preset, liga `.DN`. **Se o rung ficar falso, zera tudo.**

É o atraso para ligar: "3 segundos depois de partir, acende o sinaleiro".

TOF — *Timer Off Delay*

`TOF(timer)` — o inverso. Com o rung verdadeiro, `.DN` já está ligado. Quando o rung **cai**, começa a contar; ao chegar no preset, desliga `.DN`.

É o atraso para desligar: "o exaustor continua 30 segundos depois de o motor parar".

RTO — *Retentive Timer On*

`RTO(timer)` — conta como o TON, mas **não zera** quando o rung cai: guarda o acumulado e retoma de onde parou. Só zera com um `RES`.

É o horímetro: "somar o tempo total em que a máquina esteve em marcha".

CTU — *Count Up*

`CTU(counter)` — soma 1 no `.ACC` a **cada borda de subida** do rung. `.DN` liga quando `.ACC` alcança `.PRE`.

CTD — *Count Down*

`CTD(counter)` — subtrai 1 a cada borda de subida.

RES — *Reset*

`RES(tag)` — zera o `.ACC` e os bits de status do timer ou contador informado. É a única forma de zerar um RTO ou um contador.

7.3 Compare — comparações

Todas são condições (ficam à esquerda do rung) e aceitam **tag ou número** nos operandos.

Instrução	Operandos	Verdadeira quando
`EQU`	a, b	a = b
`NEQ`	a, b	a ≠ b
`GRT`	a, b	a > b
`GEQ`	a, b	a ≥ b
`LES`	a, b	a < b
`LEQ`	a, b	a ≤ b
`LIM`	low, test, high	*test* está entre *low* e *high*

O `LIM` tem uma sutileza fiel ao Logix: se você informar `low` **maior** que `high`, a faixa se inverte — passa a ser verdadeira **fora** do intervalo. Útil para faixas que cruzam o zero.

7.4 Compute/Math — contas

Todas são saídas (ficam à direita) e só agem com o rung verdadeiro.

Instrução	Operandos	O que faz
`MOV`	source, dest	`dest = source`
`CPY`	source, dest	idêntico ao MOV
`CLR`	dest	`dest = 0`
`ADD`	a, b, dest	`dest = a + b`
`SUB`	a, b, dest	`dest = a − b`
`MUL`	a, b, dest	`dest = a × b`
`DIV`	a, b, dest	`dest = a ÷ b` (divisão por zero é ignorada)
`SHL`	dest, casas	desloca os bits de `dest` para cima
`SHR`	dest, casas	desloca os bits de `dest` para baixo

Inteiro e real: quem manda é o destino

Existem dois tipos numéricos: **DINT** (inteiro) e **REAL** (com casas decimais). Você pode misturá-los à vontade nas contas — **o resultado obedece ao tipo da tag de destino**:

Conta	Destino REAL	Destino DINT
`2,5 + 1`	`3,5`	`3`
`7 ÷ 2`	`3,5`	`3`
`−2,5`	`−2,5`	`−2`

O corte é **truncamento** (joga fora a fração), não arredondamento — e sempre em direção ao zero. Se você precisa da fração, o destino tem de ser REAL.

Nos deslocamentos (`SHL` / `SHR`) entra sempre zero, e o bit que sai pela ponta se perde — não é rotação. Você pode endereçar cada bit de um DINT com ponto: `Palavra.0` até `Palavra.31`.

7.5 Program Control

JSR — *Jump to Subroutine*

`JSR(rotina)` — executa a sub-rotina informada quando o rung é verdadeiro.

Serve para organizar: separe intertravamentos, alarmes e sequências em rotinas próprias, e chame cada uma da MainRoutine.

Cuidado que vale a pena entender. Se você condicionar o JSR e ele deixar de ser chamado, as bobinas **dentro** da sub-rotina deixam de ser varridas — e ficam **congeladas no último estado**, ligadas. Para desligar algo, a lógica que desliga precisa continuar rodando. A regra prática é: chame a sub-rotina sempre, e coloque a condição **dentro** dela.

O aninhamento é limitado a 8 níveis.

7.6 Comunicação

MSG — *Message* (a placa é cliente Modbus TCP)

`MSG(tag_message)` — dispara uma transação Modbus TCP para **outro** equipamento quando o rung fica verdadeiro.

O operando precisa ser uma tag do tipo **MESSAGE**. **Duplo clique no bloco** abre a configuração: IP de destino, porta, unit id, função (FC), endereço inicial, quantidade e a tag de destino dos dados.

A instrução é **assíncrona**: ela apenas enfileira o pedido e o rung segue na hora. O andamento aparece nos bits `.EN` (disparada), `.DN` (concluída) e `.ER` (erro). A comunicação **não rouba tempo da varredura** — é feita por uma tarefa separada, de prioridade menor que a da lógica.

Use um **ONS** antes da MSG se quiser exatamente um disparo por acionamento.

MBSRV — servidor Modbus (a placa é servidor)

`MBSRV(array, endereço)` — publica um array do seu programa como *holding registers*, para um supervisório ou outro CLP **ler** da placa.

- Antes de usar, crie uma tag **DINT com** `Dim` — por exemplo `MB` com Dim 20. É ela que vira os registradores.
- **Duplo clique no bloco** abre a configuração: qual array e o endereço inicial.
- Enquanto o rung for verdadeiro, cada posição do array é publicada como um registrador de 16 bits, a partir do endereço escolhido. Até 20 posições.
- A placa atende na **porta 502**, **function code 3**, **somente leitura**: tentativas de escrita de fora recebem exceção. Ninguém de fora altera a memória do seu CLP.
- Com o rung falso, os registradores mantêm o último valor publicado.

Valores maiores que 16 bits são truncados; negativos aparecem em complemento de dois (−1 vira 65535), como é padrão em registrador de CLP.

7.7 Add-On — as suas instruções

A aba **Add-On** lista as instruções que **você** criou. É o assunto do próximo capítulo.

---

8. AOI — criando a sua própria instrução

Uma **Add-On Instruction** é um bloco de lógica que você escreve uma vez e usa quantas vezes quiser, com nomes e parâmetros próprios. Se você tem cinco motores iguais, escreve a lógica de *um* motor e instancia cinco vezes — e uma correção feita na AOI vale para todos.

Vamos criar a AOI `Partida_Motor`: exatamente a lógica do capítulo 4, empacotada.

> **Por que não chamá-la de `Partida`?** Porque o projeto já traz uma AOI de exemplo com esse nome, e o editor recusa nomes repetidos (`"Partida" já existe (instrução ou AOI).`). Se preferir o nome curto, exclua antes a AOI de exemplo: botão direito nela → **Excluir AOI**.

8.1 Crie a AOI

Na árvore, **botão direito** em `Add-On Instructions` → **Nova AOI...**

Digite o nome `Partida_Motor` e confirme. O editor responde:

```
AOI "Partida_Motor" criada — defina os parâmetros na barra e a lógica nos rungs.
Ela aparece na toolbox Add-On.
```

Uma aba nova abre, com uma **barra de parâmetros** no alto e uma área de ladder embaixo — que funciona exatamente como qualquer rotina.

8.2 Defina os parâmetros

Os parâmetros são a "cara" da sua instrução: o que entra, o que sai e o que ela guarda por dentro. A barra tem as colunas:

`Nome` · `Direção` · `Tipo` · `Preset` · `Descrição`

Direções possíveis:

Direção	Significado
In	entra por argumento na chamada (uma condição, um setpoint)
Out	sai para a tag informada na chamada (um comando, um estado)
Internal	vive só dentro da instância — o usuário da AOI nem vê

Tipos possíveis: `BOOL`, `DINT`, `REAL`, `TIMER`, `COUNTER`.

Para a `Partida_Motor`, use **+ Parâmetro** e monte:

Nome	Direção	Tipo	Preset	Descrição
`Liga`	In	BOOL		Botão de partida
`Para`	In	BOOL		Botão de parada
`Contator`	Out	BOOL		Saída para o contator
`Sinaleiro`	Out	BOOL		Sinaleiro de marcha
`T_Sel`	Internal	TIMER	`3000`	Atraso do sinaleiro

Clique em **Aplicar**: `Parâmetros da AOI aplicados — instâncias sincronizadas.`

> O `T_Sel` é **Internal** de propósito: cada motor precisa do *seu* timer. Se ele fosse uma tag global, os cinco motores dividiriam a mesma contagem — e nada funcionaria.

8.3 Escreva a lógica interna

Na área de ladder da aba da AOI, monte os mesmos três rungs — só que usando os **nomes dos parâmetros** no lugar das tags. Pela barra de neutral text:

```
[XIC(Liga),XIC(Contator)]XIO(Para)OTE(Contator);
XIC(Contator)TON(T_Sel);
XIC(T_Sel.DN)OTE(Sinaleiro);
```

É ladder normal: branches, timers, tudo funciona igual. A diferença é que os operandos são parâmetros, e cada instância terá a sua cópia deles.

8.4 Crie uma instância

Uma instância é uma **tag do tipo da AOI**.

Vá em **Controller Tags** → **+ Nova tag**. A linha nasce como `NovaTag<n>`: renomeie para `Motor1` e, no campo `Tipo`, escolha `Partida_Motor` (o nome da sua AOI aparece na lista, junto com BOOL, DINT e os demais). Clique em **Aplicar** — `Definições de tags aplicadas.`

Clique no ▸ ao lado de `Motor1` para ver `Motor1.Liga`, `Motor1.Contator`, `Motor1.T_Sel` e os demais.

8.5 Use no ladder

Abra a MainRoutine, crie um rung e **arraste a instrução** `Partida_Motor` da toolbox (aba **Add-On**) ou da própria árvore para dentro dele.

O bloco aparece com um pino por parâmetro:

```
┌──────────────────────┐
│   Partida_Motor      │
│  Motor1              │
│  Liga      ◂  ?      │
│  Para      ◂  ?      │
│  Contator  ▸  ?      │
│  Sinaleiro ▸  ?      │
└──────────────────────┘
```

Clique em cada pino e digite a tag que ele deve usar:

- `Liga` → `I1`
- `Para` → `I2`
- `Contator` → `Rele1`
- `Sinaleiro` → `Rele2`

Pinos deixados em branco apontam automaticamente para o membro da própria instância (`Motor1.Liga`), o que é útil quando o valor vem de outra parte do programa em vez de um borne.

Para o segundo motor: crie a tag `Motor2` do tipo `Partida_Motor`, insira outra `Partida_Motor` num rung novo e ligue nos bornes do motor 2. **A lógica é a mesma; os dados são independentes.**

8.6 O que acontece quando você grava

O microcontrolador não conhece AOIs — quem resolve isso é o compilador do editor, que **expande** a chamada em lógica comum na hora de gravar:

1. A chamada vira uma bobina interna de habilitação
2. Entram cópias dos parâmetros **In** (o argumento → o membro da instância)
3. Entram os rungs da AOI, cada um condicionado à habilitação, com os parâmetros renomeados para membros da instância (`Liga` → `Motor1.Liga`)
4. Entram cópias dos parâmetros **Out** (o membro → a tag do argumento)

Consequências práticas que vale saber:

- **A AOI não pode estar dentro de um branch.** O compilador avisa se estiver.
- Ao trazer o programa **do** CLP para a tela (upload), a lógica volta **expandida**: os nomes das AOIs e das rotinas não viajam para a placa. Ela continua correta e fiel byte a byte, só não vem mais agrupada.
- Por isso: **guarde sempre o arquivo do projeto** (Arquivo ▾ → Salvar). Ele é a única cópia que preserva AOIs, nomes e comentários.

---

9. Power fail: a placa não perde nada

Um CLP industrial guarda os valores na falta de energia porque tem memória com bateria. Esta placa resolve o mesmo problema de outro jeito, mais simples e sem bateria para trocar.

9.1 Como funciona

Uma das entradas é usada para **vigiar a energia externa**: enquanto houver 5 V nela, está tudo bem. Se ela cair, o controlador entende que a alimentação acabou e, dentro da folga que os capacitores da fonte garantem (cerca de 1 segundo):

1. **Para a CPU** — o que derruba todos os relés imediatamente (e ainda alivia a fonte, esticando a folga)
2. **Salva os valores das tags** na memória permanente
3. Guarda também se estava rodando ou parado

Quando a energia volta:

1. Os valores são **restaurados**
2. A placa **volta a rodar sozinha — se estava rodando** quando a energia caiu. Se estava parada em PROG, continua parada; e se houver falha maior pendente, também não retoma.

E se você ligar a placa com a energia externa ainda caída, ela restaura tudo mas **segura em PROG** até a entrada acionar — não se parte uma máquina com a rede elétrica oscilando.

> **O que exatamente é salvo:** as tags internas do programa. **Não** viajam as tags mapeadas em bornes (a entrada é o borne quem manda, e uma saída reenergizada por um scan seria um relé piscando) nem as tags de MESSAGE. De timers e contadores é salvo o **acumulado**; os bits de status (`EN`, `TT`, `DN`) são reconstruídos na partida, como manda a semântica de pré-scan.

9.2 O que você vê no editor

Enquanto a energia estiver caída e você estiver online, uma faixa laranja ocupa o alto do ladder:

> ⚡ **POWER FAULT** — a alimentação externa caiu (entrada de energia IN6 desacionada). A CPU está **parada**, os valores foram **salvos na flash** e as operações online estão **bloqueadas**. Energia de volta: o CLP retoma RUN sozinho.

Nesse estado, qualquer comando que dependa da placa é recusado com uma explicação — RUN, STOP, gravar, escrever valor, upload. Isso é proposital: não se manda comando para um controlador que está no escuro.

9.3 Configurando

Na árvore, **duplo clique** em `WPSIOT V3.0 2025` (ou botão direito → ⚡ **Entrada de energia (powerfail)…**).

No campo ⚡ **Entrada de energia (powerfail)** escolha o borne — ou `— desligado —` para não usar o recurso. Clique em **Aplicar**.

O projeto de demonstração vem configurado em **IN6**. Se você não tem esse borne alimentado na bancada, ou desligue o recurso, ou ligue 5 V nele.

> **Como ligar de verdade numa aplicação.** A entrada de energia deve ser alimentada por um ponto que **caia junto com a rede** (antes da fonte retificada da placa) — por exemplo, uma fonte auxiliar de 5 V sem capacitância significativa, ou o secundário de um relé de falta de fase. O que a lógica precisa é enxergar a queda **antes** que a placa perca alimentação.

---

10. Quando algo dá errado

10.1 Os relés não acionam

Verifique	Como
A placa está em RUN?	Se houver faixa `CLP em PROG`, clique nela para voltar a RUN
Há power fault?	Faixa laranja: alimente o borne de energia (9.3)
Há falha maior?	Faixa vermelha: veja 10.2
O rung tem caminho verde?	Se o caminho para na metade, a condição que falta está antes do ponto onde o verde acaba
A tag tem alias?	Sem alias, ela é só memória e não chega em borne nenhum
Duas tags com o mesmo alias?	Só a primeira recebe o borne — deixe um alias por ponto

10.2 Faixa vermelha: FALHA MAIOR

> **FALHA MAIOR do controlador** em *task* — *causa*. A varredura **parou** e as **saídas estão DESENERGIZADAS**.

As causas possíveis:

Causa	O que significa
passou do WATCHDOG de execução	a lógica demorou mais do que o tempo máximo configurado para a task
OVERLAP: a task periódica perdeu um período inteiro	a task não conseguiu rodar na cadência pedida
VIGIA: a task parou de executar	a task deixou de dar sinal de vida

Corrija a causa (quase sempre uma lógica pesada demais para o período escolhido, ou um período curto demais) e clique na própria faixa para dar RUN e limpar a falha.

10.3 A alteração que eu fiz não foi para a placa

Sintoma clássico: você editou, gravou, mas a placa continua fazendo o de antes.

Olhe a margem esquerda dos rungs: se houver um marcador `i` (fundo azul), a alteração está numa **cópia pendente** — e rung pendente não é compilado nem enviado. Clique em **Assemble**.

O editor avisa isso ao gravar, no diálogo **O que NÃO vai para a placa**. Se o Assemble for recusado, é porque há operando vazio ou `?` em algum rung: o diálogo **Assemble bloqueado** lista exatamente quais.

10.4 O editor mostra valores, mas eles estão parados

Olhe os três pontinhos ao lado do `ONLINE`: se estiverem **congelados em vermelho**, a comunicação caiu e o que está na tela é o último valor recebido — não o atual. Reconecte (**CLP ▾ → □**).

10.5 Perdi o endereço da placa

Três saídas, nesta ordem:

1. Tente `http://wpsiot.local`
2. Conecte-se à rede `WPSIOT-PLC` (senha `wpsiot123`) e abra `http://192.168.4.1` — esse caminho **sempre** funciona
3. Ligue o cabo USB, abra um terminal a 115200 e digite `info`

10.6 A gravação pela USB não aparece / falha

- O editor precisa estar aberto como **arquivo local**, não pela página da placa
- Use **Chrome ou Edge** no desktop, em aba normal
- Feche o **Monitor Serial** e qualquer outra aba do editor: só um programa por vez pode usar a porta

10.7 Diálogo "Tela ≠ CLP" ao conectar

Não é erro: é o editor avisando que o programa aberto na tela **não é** o que a placa está executando — e que monitorar assim mostraria uma lógica que não é a real. Escolha conscientemente entre gravar a tela no CLP, trazer o do CLP para a tela, ou só monitorar.

10.8 O selo não segura (o motor desliga ao soltar o botão)

Quase sempre o branch ficou **vazio** — ele nasce depois do ponto onde você clicou, sem envolver nada, e uma leg vazia é um fio: sempre verdadeira.

Selecione o rail direito do branch e use **Rail ▶** para incluir o contato dentro dele (item 4.3, passos 6 a 9). Confira o resultado na barra de neutral text: o correto é `[XIC(I1),XIC(Rele1)]…`, não `XIC(I1)[,XIC(Rele1)]…`.

---

11. Referência rápida

11.1 Endereços e acessos

Rede própria da placa	`WPSIOT-PLC` / senha `wpsiot123`
Endereço nessa rede	`http://192.168.4.1`
Nome na sua rede	`http://wpsiot.local`
Serial	115200 bps
Comandos seriais	`info` · `wifi SSID SENHA` · `run` · `stop` · `stats`
Modbus (servidor, se usar MBSRV)	porta 502, FC3, só leitura

11.2 Atalhos de teclado

Atalho	O que faz
`Ctrl+S`	salvar o projeto
`Ctrl+Z` / `Ctrl+Y`	desfazer / refazer
`Ctrl+C` `Ctrl+X` `Ctrl+V`	copiar, recortar, colar
`Ctrl+R`	novo rung
`T` (ou `Ctrl+T`)	alternar o bit selecionado
`Alt+→` / `Alt+←`	incluir / excluir elemento no branch selecionado
`F3` / `Shift+F3`	próxima / anterior referência da tag
`Enter` / `Esc`	confirmar / cancelar a edição em curso

11.3 Os menus

Arquivo ▾ — `Novo` · `Abrir…` · `Salvar` · `Importar…` · `Exportar…`

CLP ▾ — `▶ Simular` · `▶ RUN do CLP` · `□ STOP do CLP` · `⬇ Serial` · `□ WiFi` · `□ Conectar na placa` · `□ Log`

Opções ▾ — `□ Descrições no ladder` · `□ Cores e exibição…`

Na barra, fora dos menus: `Assemble` e `Cancelar edições`.

11.4 As instruções, em uma tabela

	Instrução	Operandos
Bit	XIC, XIO, OTE, OTL, OTU, ONS	tag
Timer	TON, TOF, RTO	timer
Contador	CTU, CTD	counter
	RES	tag
Compare	EQU, NEQ, GRT, GEQ, LES, LEQ	a, b
	LIM	low, test, high
Math	MOV, CPY	source, dest
	CLR	dest
	ADD, SUB, MUL, DIV	a, b, dest
	SHL, SHR	dest, casas
Controle	JSR	rotina
Comunicação	MSG	tag MESSAGE
	MBSRV	array, endereço

11.5 Sequência típica de trabalho

```
Arquivo ▾ Novo
      ↓
Controller Tags → conferir alias, criar as tags que faltam
      ↓
MainRoutine → montar os rungs
      ↓
CLP ▾ ▶ Simular → testar na tela        ← volte aqui sempre que mudar algo
      ↓
Arquivo ▾ Salvar
      ↓
CLP ▾ □ Conectar na placa
      ↓
CLP ▾ ⬇ Gravar na placa
      ↓
acompanhar online · editar com duplo clique na margem + Assemble
```

WPSIOT — Manual de primeiro uso do WPSIOT Logic Designer, firmware v1.23.