# Single Line Diagrams and Power Flow

Draw a single line diagram, connect and bind it, and see how the power flow engine energizes it at runtime.



---

Build a single line diagram
How the power flow engine works
The solution Power class

## Build a single line diagram

This walkthrough rebuilds the **SingleLine** display of the Electrical Template: a 230 kV line, a 230/69 kV transformer and one 69 kV feeder.

### 1. Create the equipment tags

In **Unified Namespace  Tags**, create one tag per piece of equipment and set its type to the matching UserType:

| Tag | Type | Equipment |
|---|---|---|
| `SL_LT1` | Generator | 230 kV line (the source) |
| `SL_89L1` | Switch | Line disconnector |
| `SL_52L1`, `SL_52T1H`, `SL_52T1L`, `SL_52F1` | Breaker | Line, transformer high side, transformer low side and feeder breakers |
| `SL_BUS230`, `SL_BUS69` | BusBar | 230 kV and 69 kV buses |
| `SL_TR1` | Transformer | 230/69 kV transformer |

A line coming from the grid is modeled as a **Generator** drawn with the `Generator_Transmission` symbol: it is the source that energizes the diagram.

### 2. Place the symbols

1. Create a Canvas display and open it in the Draw editor.
2. From the Symbols panel (**In Solution**), drag the electrical symbols onto the canvas: Generator_Transmission, Switch, Breaker, BusBar, Breaker, Transformer, Breaker, BusBar, Breaker, Standard_Load.
3. Select each symbol and set its properties:
    - The equipment link (`Generator`, `Switch`, `Breaker`, `BusBar`, `Transformer`) to its tag, for example `@Tag.SL_52L1`.
    - On the source: `BaseVoltage = @Enum.Voltage.kV_230`.
    - On the transformer: `BaseVoltage1 = @Enum.Voltage.kV_230`, `BaseVoltage2 = @Enum.Voltage.kV_69`, and the winding connections `TypeConnector1` and `TypeConnector2`.

### 3. Connect the symbols

1. Select the **Connector** tool.
2. Click a terminal of the first symbol, then a terminal of the next one. The connector attaches to the terminal; a red marker shows the terminal it will attach to.
3. Repeat for every link: source, disconnector, line breaker, 230 kV bus, transformer high side breaker, transformer, transformer low side breaker, 69 kV bus, feeder breaker, load.

On a **BusBar**, feed it on a top node (odd numbers) and take the outgoing circuits from a bottom node (even numbers). Use the end nodes 97 and 98 for bus couplers.

> A line that only looks connected does not carry energization. Always end a connector on a terminal.

### 4. Save

Saving the display stores the topology. With the Electrical Pack in the license, the save also gives every connector its own color channel, so each line shows the voltage of its own section, and writes the equipment identification to the bound tags. Save again after you change any connection.

### 5. Run

Start the runtime and open the display. Within a few seconds the engine colors the network: the source and everything it reaches through closed devices take the energized color of their voltage level; sections behind an open device take the de-energized color.

| What you see | Why |
|---|---|
| The 230 kV side and the 69 kV side in different colors | Each voltage level has its own energized color in `TensionColors`; the transformer changes the level |
| Breaker body in the closed color | Closed, good quality, from `SwitchColors` |
| Breaker body in the open color and the line below dark | Open; the downstream section is de-energized |

### 6. Try it with Simulation

Without field devices, turn on **SIMULATION** in the header. The engine then makes each equipment follow its own commands: a breaker goes to the commanded position, a generator to the commanded state.

1. Click the feeder breaker. Its dialog opens.
2. Click **Open** and confirm with **Yes**.
3. The breaker opens and the feeder is de-energized.

Simulation is a client setting: it acts only while that client has the check box on.

### Troubleshooting

| Symptom | Check |
|---|---|
| Nothing is colored | The license includes the Electrical Pack; the symbols are electrical symbols (`IsElectrical`); the display was saved after connecting. |
| One symbol stays uncolored | Its equipment link points to a tag of the same UserType as the symbol: a Breaker symbol needs a Breaker tag. |
| A section stays dark although its breakers are closed | A connector does not end on a terminal, or a busbar is fed from a bottom node. |
| The generator does not energize | `DgState` is `1` and `BaseVoltage` is set on the symbol. |

## How the power flow engine works

### Lifecycle

1. **Display opens.** With the Electrical Pack in the license and connected electrical symbols on the display, the client creates one engine instance for that display.
2. **Register.** The client registers every electrical symbol and connector on the canvas as an element, with its type (the symbol name without the `_H`, `_V` or `_Type2` suffix) and its bound equipment tag, then registers each saved connection between two terminals.
3. **Cycle.** While the display is open, the engine runs on every refresh cycle of the display.

### What one cycle does

1. Reads the Simulation setting of the client.
2. Copies the tag values into the network: generator state and base voltage, breaker, disconnector and fuse positions and quality, transformer winding voltages.
3. Orders the generators by voltage, highest first, and walks the network from each running generator through the connections:
    - closed breakers, disconnectors and fuses pass power; open ones stop it;
    - transformers change the voltage to the base voltage of each winding;
    - bars and busbars join every connected line;
    - connectors carry the voltage of their section.
4. Writes the result back to the tags: colors, states, and terminal voltages, plus the color channel of each connector.
5. When Simulation is on, each equipment also follows its commands.

A closed loop in the network is handled: an element is visited once per walk, so a ring does not repeat. A connection to an element that is no longer on the display is skipped, and the runtime trace names the element that holds it.

### Colors

- **Lines and equipment**: the `TensionColors` enumeration. Each voltage level has an energized color (ActiveColor) and a de-energized color (InactiveColor).
- **Switching devices**: the `SwitchColors` enumeration, one color per state (intermediate, closed, open, inconsistent); good quality uses ActiveColor and bad quality uses InactiveColor.

Change a color once in the enumeration and every diagram follows it.

### What each element reads and writes

| Element | Reads | Writes | With Simulation on |
|---|---|---|---|
| Generator | `DgState`, `TBaseVoltage` | `TCurrentColor` | State follows `CmdState`; emergency stop stops it |
| Breaker | `InBreakerPosition1`, and `InBreakerPosition2` when enabled, with quality | Colors, state, terminal voltages and states | Position follows `CmdBreakerPosition`; releases follow the state |
| Switch | `InSwitchPosition1`, `InSwitchPosition2` | As Breaker | Position follows `CmdSwitchPosition` |
| Fuse | `DgFusePosition` when enabled | Colors and state | |
| Transformer, Transformer3 | Winding base voltages | Colors and voltages per winding | Paralleling, forced air and tap mode follow their commands |
| Reactor, Capacitor, Rectifier | | Colors and states of both terminals | |
| Bar, BusBar, Standard, Jumper, Connector | | Color of the element | |

## The solution Power class

The power flow engine is built into the product. A solution can replace it with its own: a Script Class named **`Power`**, domain **Client**, content **Namespace**, declaring the namespace `T.MyPower`. When that class exists, the client loads `T.MyPower.Power` (or `T.MyPower.PowerBase`); otherwise it uses the built in engine.

The Electrical Template ships its own `Power` class: a full copy of the engine that adds the directional **BusBar** element. Displays that use the BusBar symbol need this class. The class is compiled for both the desktop clients and the HTML5 client, so keep any change compatible with both.

### Customizing the engine

Edit **Scripts  Classes  Power**. Common changes:

- **A new equipment type**: add a class derived from `BaseElement`, named exactly as the symbol (without suffix), and override the steps you need.
- **Different energization rules**: override the execution steps of an existing element, for example to treat a breaker in test position as open.
- **Different simulation**: override `Simulation()` of an element.

```
public class MySwitchgear : BaseElement
{
    public Terminal Terminal1 { get; set; } = new Terminal();
    public Terminal Terminal2 { get; set; } = new Terminal();

    public MySwitchgear(string objectNameOrUid, Guid guid, PowerBase parent)
        : base(objectNameOrUid, guid, parent) { }

    public override void CopyTagToNet()  { /* read your tag members */ }
    public override void CopyNetToTag()  { /* write colors and states */ }
    public override Terminal GetTerminal(int terminal)
        => terminal == 1 ? this.Terminal1 : this.Terminal2;
}
```

Build the solution after editing. If the class does not build, the display shows no energization and the client trace reports that the Power class could not be found.

### Debugging

With Debug tracing on, the engine writes the network topology to the trace when the display opens (which element connects to which), and reports any cycle that takes longer than 50 ms.

---

#### In this section...