Modeling sCO_2 Power Cycles with Twine: A Composable, Open-Source Framework in Rust

John Dyreby
Greg Troszak
isentropic development

Twine is a Rust framework for building models as compositions of pure, strongly typed functions

Safe

- Pure functions make behavior predictable and easy to test
- Strong typing guarantees component connections are valid
- Compile-time unit checking enforces dimensional consistency and handles conversions automatically

Fast

- Compiled Rust binaries run efficiently without interpreter overhead
- Fast model evaluations support optimization and parametric studies
- Pure, stateless components enable safe parallel execution

Portable

- Models can be compiled for use across diverse workflows
- Compiled code runs in desktop, cloud, or embedded environments
- Rust's ecosystem provides libraries that enable integration with Python, MATLAB, and other languages

Batteries Included

- Built-in component models, such as turbomachinery and heat exchangers, provide ready-to-use building blocks
- Shared solvers and integrators work with any system model
- Multiple thermodynamic property libraries are supported

Twine is open-source software available at https://github.com/isentropic-dev/twine

Modeling the Recompression Cycle

```rust
fn compressor<Fluid>(
    inlet: State<Fluid>,
    p_out: Pressure,
    eta: IsentropicEfficiency,
    thermo: &impl ThermoModel<Fluid>,
) -> CompressorResult<Fluid> {
    // pure, stateless model
}

struct CompressorResult<Fluid> {
    outlet: State<Fluid>,
    work: SpecificEnthalpy,
}
```

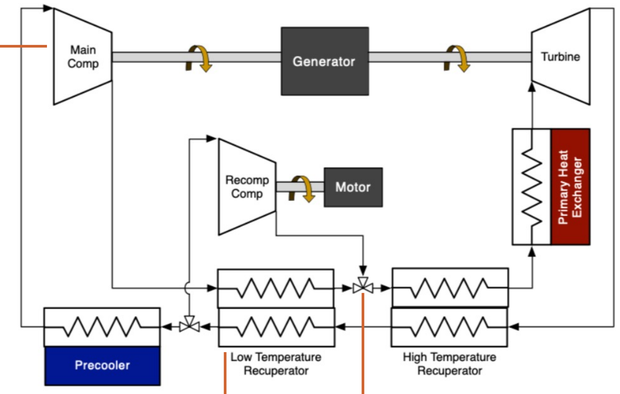
Inputs and outputs are strongly typed physical quantities

```rust
fn recuperator<Fluid>(
    cold_stream: Stream<Fluid>,
    hot_stream: Stream<Fluid>,
    cold_dp: PressureDrop,
    hot_dp: PressureDrop,
    ua: ThermalConductance,
    thermo: &impl ThermoModel<Fluid>,
) -> RecuperatorResult<Fluid> {
    // pure, stateless model
}

struct Stream<Fluid> {
    m_dot: MassRate,
    state: State<Fluid>,
}

enum PressureDrop {
    None,
    Absolute(Pressure),
    Fraction(Ratio)
}
```

All components use the same typed, pure function pattern




```rust
// Call the low temperature recuperator and recompressor
let lt_recup = recuperator(/* … */)?;
let recomp = compressor(/* … */)?;

// Build the mixing valve input from upstream outputs
let from_lt_recup = Stream {
    m_dot: m_dot_comp,
    state: lt_recup.top.outlet,
};
let from_recomp = Stream {
    m_dot: m_dot_recomp,
    state: recomp.outlet,
};
let mix_input = MixingValveInput {
    streams: [from_lt_recup, from_recomp]
};

// Call the mixing valve
let mix = mixing_valve(mix_input, &thermo)?;
```

Component inputs are built from upstream outputs, and only valid typed connections will compile

- Design-point and off-design cycle models are assembled from component models
- These assembled models are solved using shared numerical methods to support:
  - **design-point optimization**, including component sizing and tradespace exploration
  - **off-design operation**, enabling performance evaluation and control strategy design
  - **annual simulation**, running the off-design model under time-varying operating conditions

Full source code and an interactive demo are available at https://github.com/isentropic-dev/brayton

The 9th International Supercritical CO_2 Energy Technologies Symposium • March 2 – 5, 2026 • Pittsburgh, PA, USA