```
html"""
<style>
    main {
        margin: 0 auto;
        max-width: 2000px;
        padding-left: max(160px, 10%);
        padding-right: max(160px, 10%);
    }
</style>
"""
```

```
begin
    using StanSample ✓
    using DataFrames ✓
    import StanSample: BS
end
```

```
"/Users/rob/.julia/dev/Stan/Example_Notebooks/E
```

```
pwd()
```

Setup BridgeStan if necessary.

If `bridgestan` is installed in the same directory as `cmdstan`, StanSample includes setup. See INSTALLING_CMDSTAN.md in StanSample.jl

Run the Stan Language program

```
bernoulli = "
data {
  int<lower=1> N;
  int<lower=0,upper=1> y[N];
}
parameters {
  real<lower=0,upper=1> theta;
}
model {
  theta ~ beta(1,1);
  y ~ bernoulli(theta);
}
";
```

```
data =
▶Dict("N" ⇒ 10, "y" ⇒ [0, 1, 0, 1, 0, 0, 0, 0
```

```
data = Dict("N" => 10, "y" => [0, 1, 0, 1,
0, 0, 0, 0, 0, 1])
```

```
begin
    sm = SampleModel("bernoulli",
    bernoulli)
    rc = stan_sample(sm; data,
    save_warmup=true)
end;
```

```
/var/folders/l7/pr04h0650q5dvqttnvs8s
2c00000gn/T/jl_Mx0KN7/bernoulli.stan
updated.
```

Creade the BridgeStan model library

```
▶Dict(:suffix ⇒ ["1", "2", "3", "4"], :chain =
```

```
available_chains(sm)
```

```
begin
    chain_id = 2
    smb = BS.StanModel(stan_file =
    sm.output_base * ".stan", data =
    sm.output_base *
    "_data_$(chain_id).json")
end;
```

Model name:

```
"bernoulli_model"
```

```julia
BS.name(smb)
```

Number of model parameters:

```
1
```

```julia
BS.param_num(smb)
```

Compute log_density and gradient at a random observation

```
▶ (log_density = -6.19127, gradient = [-0.60880
```

```julia
let
    x = rand(BS.param_unc_num(smb))
    q = @. log(x / (1 - x)); #
    unconstrained scale
    ld, grad =
    BS.log_density_gradient(smb, q,
    jacobian = false)
    (log_density=ld, gradient=grad)
end
```

Or a range of densities

	x	q	log_density	gradient
1	0.1	-2.19722	-7.64528	▶ [2.0]
2	0.108081	-2.1105	-7.47529	▶ [1.9191
3	0.116162	-2.02929	-7.32269	▶ [1.8383
4	0.124242	-1.95285	-7.18522	▶ [1.7575
5	0.132323	-1.88057	-7.06108	▶ [1.6767
6	0.140404	-1.81194	-6.94874	▶ [1.5959
7	0.148485	-1.74653	-6.84698	▶ [1.5151
8	0.156566	-1.68401	-6.75475	▶ [1.4343
9	0.164646	-1.62405	-6.67117	▶ [1.3535
10	0.172727	-1.56642	-6.59547	▶ [1.2727
⋮ more				
100	0.9	2.19722	-16.4342	▶ [-6.0]

```
if typeof(smb) == BS.StanModel
    x = rand(BS.param_unc_num(smb))
    q = @. log(x / (1 - x))        #
unconstrained scale

    function sim(smb::BS.StanModel,
x=LinRange(0.1, 0.9, 100))
        q = zeros(length(x))
        ld = zeros(length(x))
        g = Vector{Vector{Float64}}(undef,
length(x))
        for (i, p) in enumerate(x)
            q[i] = @. log(p / (1 - p)) #
unconstrained scale
            ld[i], g[i] =
BS.log_density_gradient(smb, q[i:i],
                jacobian = 0)
        end
        return DataFrame(x=x, q=q,
log_density=ld, gradient=g)
    end

  sim(smb)
end
```

Check the BridgeStan model library has been created in the tmpdir

```
▶ ["bernoulli", "bernoulli.hpp", "bernoulli.sta
```

```
readdir(sm.tmpdir)
```