Toggly Ruby

Core client, Rails requests, definition snapshots and telemetry

Install

GEMFILE

```
gem 'toggly'       # Plain Ruby
# Or gem 'toggly-rails' for Rails (includes core)
gem 'toggly-cache' # Optional Redis snapshots
# Optional transports:
gem 'websocket-client-simple', '~> 0.8'
gem 'grpc'
gem 'google-protobuf'
```

SHELL

```
bundle install
bundle exec rails generate toggly:install # Rails
```

Use a Ruby version supported by your Rails version. Let Bundler resolve dependencies and commit Gemfile.lock; gem versions need not match.

Core startup and cleanup

CHECKOUT.RB

```
require 'toggly'
client = Toggly::Client.new(
  app_key: ENV['TOGGLY_APP_KEY'],
  environment: 'Production',
  defaults: { 'ExpressCheckout' => false }
)
begin
  # Build context before the corresponding check.
  context = Toggly::Context.new(
    identity: 'user-123', groups: ['premium'],
    claims: { 'role' => 'customer' },
    request: Toggly::RequestContext.new(
      country: 'US', accept_language: 'en-US',
      user_agent: 'Mozilla/5.0 (Macintosh)'
    ),
    entity: Toggly::EntityContext.new(
      kind: 'Order', key: 'order-42',
      attributes: {
        'Vip' => true, 'Total' => 149.95 }
    )
  )
  puts client.enabled?('ExpressCheckout',
    context: context)
ensure
  client.close
end
```

Definitions and decisions

The constructor loads app/environment definitions. User and Order context is supplied to local evaluations later; changing context does not fetch definitions. Reuse one client per process. Services close it at shutdown, not after each request.

Without a key, nonempty defaults select offline mode. Missing both key and defaults raises ConfigError. Disabling background refresh alone still permits the initial fetch.

Boolean diagnostics

INSIDE THE BEGIN BLOCK ABOVE

```
client.disabled?('ExpressCheckout',
  context: context)
client.enabled?('unknown', context: context,
  default: false)
result = client.evaluate('ExpressCheckout',
  context: context)
puts result.to_h
```

Result fields: feature_key, enabled, reason, matched_rule, matched_rule_index. evaluate does not apply unknown defaults or record a check. It is not variant allocation.

Variants (catalog-local)

AMBIENT IDENTITY FIRST; CONTEXT: IS OVERRIDE

```
client = Toggly::Client.new(
  app_key: ENV['TOGGLY_APP_KEY'],
  environment: 'Production',
  identity: 'user-123')
client.set_identity('user-123')

v = client.get_variant('checkout-flow')
puts v&.name # e.g. "treatment", or nil
client.get_variant_value('checkout-flow')

# Rails: feature_variant(:checkout_flow)
# uses toggly_context
```

No `enable_variants` toggle. The `variant:` kwarg on record_usage/record_view is a telemetry label, not this assignment.

Rails: one initializer

CONFIG/INITIALIZERS/TOGGLY.RB

```
Toggly::Rails.configure do |config|
  config.app_key = Rails.application.credentials.dig(:toggly, :app_key)
  config.environment = Rails.env.production? ? 'Production' : 'Staging'
  config.defaults = { 'ExpressCheckout' => false }
  config.context_builder = ->(request, user) do
    # role/role_names are your authenticated user's application methods.
    Toggly::Context.new(
      identity: user&.id&.to_s,
      groups: user ? user.role_names : [],
      claims: user ? { 'role' => user.role } : {},
      request: Toggly::HttpRequestMapper.from_http_headers(
        'user-agent' => request.user_agent,
        'accept-language' => request.headers['Accept-Language'],
        'cf-ipcountry' => request.headers['CF-IPCountry']
      ),
      entity: request.env['toggly.entity']
    )
  end
end
```

Claims come from verified application state. Trust proxy headers only through your configured proxy. Default Rails mapping has identity/groups/traits/entity; use this builder for claims and structured request fields.

Entity before first request check

APP/CONTROLLERS/ORDERS_CONTROLLER.RB

```
class OrdersController < ApplicationController
  prepend_before_action :load_toggly_order
  def show
    render json: { express: feature_enabled?('ExpressCheckout') }
  end
  private
  def load_toggly_order
    # Authenticate before this callback; scope lookup to authorized Orders.
    order = current_user.orders.find(params[:id])
    request.env['toggly.entity'] = Toggly::EntityContext.new(
      kind: 'Order', key: order.id.to_s,
      attributes: { 'Vip' => order.vip?, 'Total' => order.total.to_f }
    )
  end
end
```

The SDK before-action caches context. Later env changes do not rebuild it. For another authorized Order, create another EntityContext and pass toggly_context.with_entity(other_entity) as context: to the next check. User identity stays the same.

Rails views and gates

APP/VIEWS/ORDERS/SHOW.HTML.ERB

```
<%= feature('ExpressCheckout') do %>
  <p>Express checkout</p>
<% end %>
<%= feature('ExpressCheckout', negate: true) do %>
  <p>Standard checkout</p>
<% end %>
```

INSIDE A CONTROLLER

```
before_action do
  require_feature!('ExpressCheckout')
end
```

The gate returns 404 when disabled; HTML needs public/404.html. Keep normal authentication and authorization.

Definition snapshots

ALTERNATIVE CORE STARTUP

```
require 'toggly'
provider = Toggly::SnapshotProviders::File.new(
  path: './tmp/shop-Production-toggly.json')
client = Toggly::Client.new(
  app_key: ENV['TOGGLY_APP_KEY'],
  environment: 'Production',
  defaults: { 'ExpressCheckout' => false },
  snapshot_provider: provider)
# Use this process client; close at shutdown.
client.close
```

Snapshots persist definitions, not user decisions. Startup loads the snapshot and then attempts a fetch. Refresh can change the next evaluation.

Redis snapshot provider

CHOOSE INSTEAD OF THE FILE PROVIDER ABOVE

```
require 'toggly-cache'
require 'redis'
redis = Redis.new(url: ENV['REDIS_URL'])
provider = Toggly::Cache::RedisSnapshotProvider.new(
  redis: redis,
  key_prefix: 'shop:Production:toggly',
  ttl: 3600)
provider.exists?
provider.remaining_ttl # Seconds or nil
provider.touch         # Uses configured TTL
# provider.clear       # Remove persisted snapshot
```

Choose a separate prefix for each app/environment. Rails: set use_rails_cache and cache_key_prefix in the existing initializer. The default prefix does not isolate apps.

Native Rails lifecycle

The Railtie installs middleware and controller/view helpers, and closes the client at exit. Middleware clears Toggly context/user entries on normal and error requests; application entity data is request-owned. Configure only once.

Rails uses Rails.logger automatically. Its Configuration has no logger, enable_live_updates, or telemetry setters. See the core-options startup recipe in the live-updates guide.

Context rules

identity/groups describe the user; claims feed UserClaims; request feeds HTTP filters; entity feeds ContextProperty. Traits do not substitute for claims or request. Context data is not deeply frozen; create request-owned contexts.

cache_key excludes entity and definition revision, so it is not sufficient for caching evaluation results. Register optional entity mappers/schema before constructing the client.

Refresh and signing

enable_live_updates defaults true; it needs websocket-client-simple. disable_background_refresh prevents polling and WebSocket startup. use_signed_definitions switches endpoints; Ruby does not verify signatures locally, and allowed_key_ids is unused.

Offline Rails tests

USE IN THE SINGLE TEST INITIALIZER

```
Toggly::Rails.configure do |config|
  config.app_key = nil
  config.defaults = { 'my_feature' ⇒ false }
  config.disable_background_refresh = true
end
```

Initialize before using test helpers. Overrides change global definitions; use process isolation for parallel tests.

RSpec

AFTER RAILS/TEST INITIALIZER LOADS

```
require 'rspec'
require 'rspec/expectations'
require 'toggly/rails/testing'
RSpec.configure do |config|
  config.include(
    Toggly::Rails::Testing::RSpecHelpers)
end
RSpec.describe 'flags' do
  it 'restores after the block' do
    with_feature(:my_feature, enabled: true) do
      expect(:my_feature).to be_feature_enabled
    end
    expect(:my_feature).not_to be_feature_enabled
  end
end
```

Minitest

AFTER RAILS/TEST INITIALIZER LOADS

```
require 'minitest/autorun'
require 'toggly/rails/testing'
class FlagsTest < Minitest::Test
  include Toggly::Rails::Testing::MinitestHelpers
  def test_override
    with_feature(:my_feature, enabled: true) do
      assert_feature_enabled(:my_feature)
    end
    assert_feature_disabled(:my_feature)
  end
end
```

Usage and metrics

WITH SHARED CLIENT AND REQUEST CONTEXT

```
client.record_view('ExpressCheckout',
  identity: context.identity)
client.record_usage('ExpressCheckout',
  identity: context.identity)
client.measure('checkout_total', 149.95,
  feature: 'ExpressCheckout', variant: 'enabled')
client.increment_counter('orders', 1,
  feature: 'ExpressCheckout', variant: 'enabled')
client.observe('checkout_ms', 85,
  feature: 'ExpressCheckout', variant: 'enabled')
client.flush_telemetry
```

Record views at display and usage at interaction. measure sums; counters add; observe records individual values. variant is a telemetry label.

Telemetry setup and resources

Core hash construction with an app key defaults telemetry on. Block/Rails configuration does not recalculate defaults when assigning a key: enable_usage_tracking and enable_metrics must be set on the core Config. Default senders need grpc/google-protobuf. TOGGLY_DISABLE_TELEMETRY=1 disables all telemetry.

Ruby guides · Testing and WebMock setup · Rails core-options recipe · Samples catalog