initialization

client

```
def start_link(opts \\ []) do
 GenServer.start_link(__MODULE__, :ok, opts)
end
```

immediate

```
{:ok, pid}
```

callback

```
def init(:ok) do
 state = init_state()
 return_value
end
```

^return_value =

```
{:ok, state}
{:ok, state, 5_000}
{:ok, state, :hibernate}
{:stop, reason * }
:ignore!
```

termination

client

```
def stop(pid, reason \\ :normal,
              timeout \\ :infinity) do
 GenServer.stop(pid, reason, timeout)
end
```

immediate

```
:ok
```

callback

```
def terminate(reason, state) do
# Perform cleanup here
# ...
end
```

^reason =

```
:normal
:shutdown
{:shutdown, term}
term
```

synchronous operation

client

```
def sync_op(pid, args) do
 GenServer.call(pid, {:sync_op, args})
end
```

immediate

waits for callback and returns `reply` if returned value from callback matches {`:reply, reply, _`}

callback

```
def handle_call({:sync_op, args}, from, state) do
 new_state = f(state, args)
 return_value
end
```

^return_value =

```
{:reply, reply, new_state}
{:reply, reply, new_state, 5_000}
{:reply, reply, new_state, :hibernate}

{:noreply, new_state}
{:noreply, new_state, 5_000}
{:noreply, new_state, :hibernate}

{:stop, reason * , reply, new_state}
{:stop, reason * , new_state}
```

asynchronous operation

client

```
def sync_op(pid, args) do
 GenServer.cast(pid, {:async_op, args})
end
```

immediate

```
:ok
```

callback

```
def handle_cast({:sync_op, args}, state) do
 new_state = f(state, args)
 return_value
end
```

^return_value =

```
{:noreply, new_state}
{:noreply, new_state, 5_000}
{:noreply, new_state, :hibernate}

{:stop, reason * , new_state}
```

handling information messages

```
def handle_info(msg, state) do
new_state = f(state, msg)
{:noreply, new_state}
end

{:noreply, new_state}
{:noreply, new_state, 5_000}
{:noreply, new_state, :hibernate}
{:stop, reason * , new_state}
```