GenServer - a cheat sheet

— last version: `https://elixir-lang.org/getting-started/mix-otp/cheat-sheet.pdf`

— reference: `https://hexdocs.pm/elixir/GenServer.html`

initialization: .start → **`init/1`**

client

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

returns

```
{:ok, pid}
:ignore
{:error, term}
```

callback

```
def init(match_this) do
  # process input and compute result
  result
end
```

stop reasons

- `:normal` doesn't log, doesn't break links
- `:shutdown`, `{:shutdown, _}` doesn't log, breaks links
- anything else logs, breaks links.

applies globally

ˆreason =

```
:normal
:shutdown
{:shutdown, _}
_
```

ˆresult =

```
{:ok, state}
{:ok, state, then_what}
```

```
{:stop, reason}
:ignore
```

termination: .stop → **`terminate/2`**

client

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

returns

```
:ok
```

callback

```
def terminate(reason, state) do
  # perform cleanup
  # result will not be used
end
```

:stop

`terminate/2` is also called when `:stop` is returned and in case of errors, when `Process.flag(:trap_exit)` is true.

asynchronous operation: .cast → **`handle_cast/2`**

client

```
def your_api_async_op(pid, args) do
  GenServer.cast(pid, match_this)
end
```

returns

```
:ok
```

callback

```
def handle_cast(match_this, state) do
  # process input and compute result
  result
end
```

ˆthen_what =

applies globally

```
timeout_milliseconds
:hibernate
{:continue, match_this}
```

ˆresult =

```
{:noreply, state}
{:noreply, state, then_what}
```

```
{:stop, reason, state}
```

synchronous operation: .call → **handle_call/3**

client

```
def your_api_sync_op(pid, args) do
 GenServer.call(pid, match_this)
end
```

callback

```
def handle_call(match_this, from, state) do
 # process input and compute result
 result
end
```

^result =

```
{:reply, reply, state}
{:reply, reply, state, then_what}
```

```
{:noreply, state}
{:noreply, state, then_what}
```

```
{:stop, reason, reply, state}
```

returns

waits for callback, receives `reply` if result matches `{:reply, reply, ...}` or `{:stop, _, reply, _}`.

^reply =

user defined

handling messages: → **handle_info/2**

client

```
def handle_info(match_this, state) do
 # process input and compute result
 result
end
```

^result =

```
{:noreply, state}
{:noreply, state, then_what}
```

```
{:stop, reason, state}
```

^then_what = {:continue, match_this} → **handle_continue/2**

client

```
def handle_continue(match_this, state) do
 # process input and compute result
 result
end
```

^result =

```
{:noreply, state}
{:noreply, state, then_what}
```

```
{:stop, reason, state}
```

explain reason

the reasons (`:normal`, `:shutdown`, `{:shutdown, _}`) expected from a well behaved `GenServer` are not defined by `GenServer`, but by either Erlang (`:normal`) or the `Supervisor` module.

footnotes

— use `@impl true` before each definition to guarantee it matches the equivalent `GenServer` callback.

— callbacks not listed here are: `code_change/3` and `format_status/2`.

— source: `https://github.com/elixir-lang/elixir-lang.github.com`