

# GenServer - a cheat sheet

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

source: <https://github.com/elixir-lang/elixir-lang.github.com>; license: CC:BY-SA; Copyright: check the source.

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
```

**callback**

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

**`^result =`**

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

---

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

**returns**

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

**applies globally**

**`^reason =`**

```
:normal
:shutdown
{:shutdown, any}
any
```

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

**client**

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

**callback**

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

**returns**

```
:ok
```

**`: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
```

**callback**

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

**`^result =`**

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

---

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

**returns**

```
:ok
```

**applies globally**

**`^then_what =`**

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

### 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}
```

### `{: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}
```

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`.