A quick introduction to

OTP Applications

using a

Gen_server example

on Erlang/OTP R15B

Baed on the official online manuals and a gen_server example from http://www.rustyrazorblade.com/2009/04/erlang-understanding-gen_server/

files and source code

```
0 directories, 5 files
├── demo.script
├── demo_application.app
├── demo_application.erl
├── demo_supervisor.erl
└── demo_gen_server.erl
```

demo.script

```
 %% -*- erlang -*-
main(_Strings)->

  % compile all the .erl files to .beam files
  make:all(),

  % start demo_application
  application:start(demo_application),

  % run demo_gen_server:add/1
  Output = demo_gen_server:add(10),
  io:format(
    "\n\nInitial demo_gen_server state was 0;
adding 10; returned: ~p\n\n",
    [Output]
  ),

  halt().
```

demo_application.app

```
{ application,
  demo_application,
  [ { mod,
      { demo_application,
        []
      }
    },
    { applications,
      [ kernel,
        stdlib
      ]
    }
  ]
}.
```

demo_application.erl

```
-module(demo_application).
-behaviour(application).
-export([start/2, stop/1]).
-compile(native).

start(_Type,_Args) ->
  demo_supervisor:start_link().

stop(State) ->
  {ok,State}.
```

demo_supervisor.erl

```
-module(demo_supervisor).
-behaviour(supervisor).
-export([start_link/0, init/1]).
-compile(native).

start_link() ->
  supervisor:start_link(
    { local,
      demo_supervisor_instance1
    },
    demo_supervisor,
    []
  ).

init(_Args) ->
  { ok,
    { { one_for_one,
        1,
        60
      },
      [ { demo_gen_server_child_id,
          { demo_gen_server,
            start_link,
            []
          },
          permanent,
          brutal_kill,
          worker,
          [ demo_gen_server ]
        }
      ]
    }
  }.
```

demo_gen_server.erl

```
-module(demo_gen_server).
-export( [ start_link/0,
           add/1,
           subtract/1,
           init/1,
           handle_call/3
         ]
       ).
-compile(native).

start_link()->
  gen_server:start_link(
    { local,
      demo_gen_server_instance2
    },
    demo_gen_server,
    [],
    []
  ).

init(_Args) -> {ok, 0}.

add(Num) ->
  gen_server:call(
    demo_gen_server_instance1,
    {add, Num}
  ).

subtract(Num) ->
  gen_server:call(
    demo_gen_server_instance1,
    {subtract, Num}
  ).

handle_call({add, Num}, _From, State) ->
  {reply, State + Num, State + Num};
handle_call({subtract, Num}, _From, State) ->
  {reply, State - Num, State - Num}.
```

a quick demo

Method 1:
just run
demo.script.

```
PROMPT: escript demo.script
Recompile: demo_application
Recompile: demo_gen_server
Recompile: demo_supervisor


Initial demo_gen_server state was 0;
adding 10; returned: 10

PROMPT:
```

Method 2:
start the Erlang shell,
then manually run the commands from demo.script.

```
PROMPT: erl
Erlang R15B (erts-5.9) [source] [64-bit]
[smp:2:2] [async-threads:0] [hipe]
[kernel-poll:false]

Eshell V5.9  (abort with ^G)
1> make:all().
Recompile: demo_application
Recompile: demo_gen_server
Recompile: demo_supervisor
up_to_date
2> application:start(demo_application).
ok
3> demo_gen_server:add(10).
10
4> q().
ok
5>

PROMPT:
```

tear down

overview

```
0 directories, 5 files
├── demo.script
├── demo_application.app
├── demo_application.erl
├── demo_supervisor.erl
└── demo_gen_server.erl
```

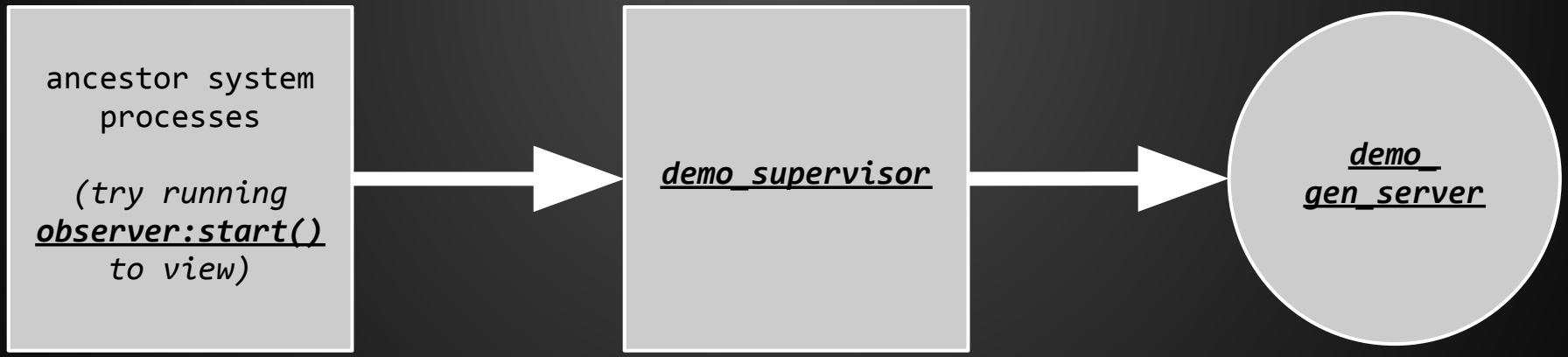
```
make:all().
```

This searches the current working directory, compiles any .erl files to .beam files if the latter do not already exist, and recompiles any .erl files which have changed since their existing .beam files were compiled.

```
application:start(demo_application).
```

This reads ***demo_application.app***, which is an application configuration file, then attempts to compile, load, and start the application.

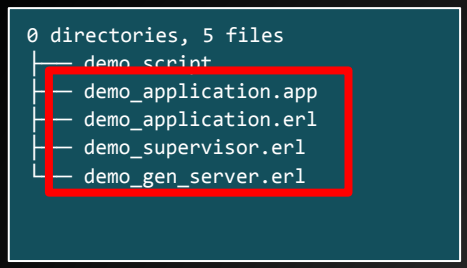
The internal events are expanded on the next slide.




```
demo_gen_server:add(10).
```

The "callback" module ***demo_gen_server*** implements the stock OTP "behaviour" module ***gen_server***, and is now instantiated as a process in a supervision tree.

The internal events are expanded on the next slide.

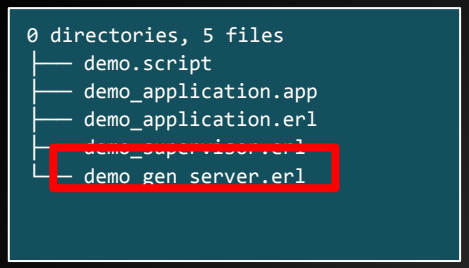


loading & starting

(1)	calling:	(behaviour module:function)	***application:start(demo_application)***
(2)	(1) calls:	behaviour module:function)	***application:load(demo_application)***
(3)	(2) reads:	(configuration file)	***demo_application.app***
(4)	(3) points to:	(callback module)	***demo_application***
(5)	(1) calls:	(callback module:function)	***demo_application:start/2***
(6)	(5) calls:	(callback module:function)	***demo_supervisor:start_link/0***
(7)	(6) calls:	(behaviour module:function)	***supervisor:start_link/2,3***
(8)	(7) calls:	(callback module:function)	***demo_supervisor:init/1***
(9)	(8) points to:	(callback module:function)	***demo_gen_server:start_link/0***
(10)	(7) calls:	(callback module:function)	***demo_gen_server:start_link/0***
(11)	(10) calls:	(behaviour module:function)	***gen_server:start_link/3,4***
(12)	(11) calls:	(callback module:function)	***demo_gen_server:init/1***

... and the application is started!

The process is probably even more complicated, further under the hood, but this should suffice for introductions.



using/calling/doing

(1) calling: (callback module:function) ***demo_gen_server:add(10)***

(2) (1) calls: (behaviour module:function) ***gen_server:call/2,3***
(3) (2) calls: (callback module:function) ***demo_gen_server:handle_call/3***

... then work is done, and a result is returned!