Factory Contracts

Factory contracts are contracts that create other contracts. They are useful for:

- Dividing your app into parts, each with different responsibilities.
- Extending the scope of your app by dynamically adding a piece of functionality.
- Managing different parts of the app from a single location, etc.
-

TheFactory Contract Template is a good starting point for setting up your own factory contracts and experimenting with them. Here are some examples of active projects that use factory contracts:

- SecretSwap
- , where a Factory contractcreates
- a new Child contract for every new pair supported on the exchange. (Cosmwasm v0.10).
- SecretJack
- , where a Parent "Bank" Contract which manages the funds creates
- a single "Game" Contract which manages the game logic. (Cosmwasm v0.10).
-

How to create Factory contracts

After the child contract is stored on chain, you can instantiate child contracts from the factory in the following manner:

Cosmwasm v1 Cosmwasm v0.10 Assume the child contract expects the following instantiate message:

```

Copy // example of an Instantiate Msg

# [derive(serde::Serialize, serde::Deserialize)]

pubstructOffspringInstantiateMsg{ pubexample_field:i32, }

```

To instantiate the child contract, you can send asubmessage from the parent with the child's instantiate message:

```

Copy usecosmwasm_std::{SubMsg,Response}; usesecret_toolkit::utils::{InitCallback};

```

```

Copy constOFFSPRING_INSTANTIATE_REPLY_ID:u64=1;

// Implement InitCallback on the OffsprintInit message // to be able to convert it to cosmosMsg
implInitCallbackforOffspringInstantiateMsg{ constBLOCK_SIZE:usize=BLOCK_SIZE; }

// populate the message letinit_msg=OffspringInstantiateMsg{ example_field:1, };

// build a submessage from the offspring init message letinit_submsg=SubMsg::reply_always( initmsg.to_cosmos_msg( "example_label".to_string(),// label for offspring contract 1,// offspring code id "d519793af2...".to_string(),// offspring code hash None,// funds amount )?, OFFSPRING_INSTANTIATE_REPLY_ID, );

// then build the response with the submessage Ok(Response::new().add_submessage(init_submsg))

```

If you don't care about the reply, you can send a regular message instead:

```

Copy letinit_cosmos_msg=initmsg.to_cosmos_msg( "example_label".to_string(),// label for offspring contract 1,// offspring code id "d519793af2...".to_string(),// offspring code hash None,// funds amount )?;

Ok(Response::new().add_message(init_cosmos_msg))
```

``` Assume the child contract expects the following instantiate message:

```

Copy // example of an Instantiate Msg

[derive(serde::Serialize, serde::Deserialize)]

pubstructOffspringInstantiateMsg{ pubexample_field:i32, }

```

To instantiate the child contract, send a message on the parent's response:

```

Copy usecosmwasm_std::{CosmosMsg,WasmMsg}; usesecret_toolkit::utils::{InitCallback};

```

```

Copy // Implement InitCallback on the OffsprintInit message // to be able to convert it to cosmosMsg implInitCallbackforOffspringInstantiateMsg{ constBLOCK_SIZE:usize=BLOCK_SIZE; }

letmessages=vec![initmsg.to_cosmos_msg("example_label".to_string(),// label for offspring contract 1,// offspring code id "d519793af2...".to_string(),// offspring code hash None,// funds amount)?,]);

// Then return the message from an entry point, for example on init: Ok(InitResponse{ messages, log:vec![], })

```

Last updated1 year ago On this page Was this helpful?Edit on GitHub Export as PDF
```