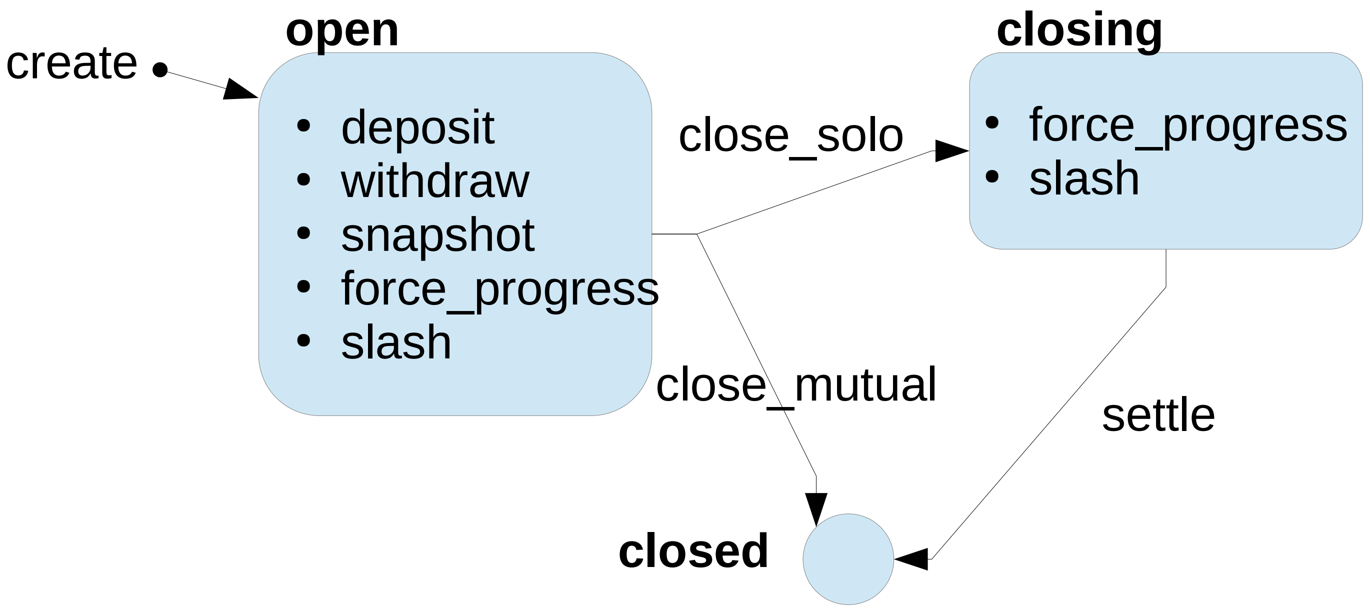
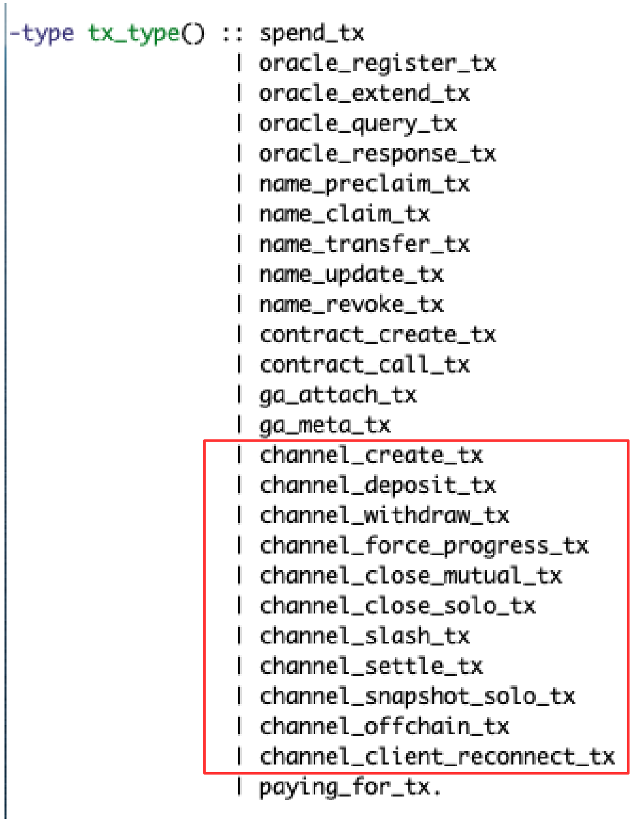
State Channels On-Chain

Ulf Wiger
Dimitar Ivanov

Transaction types

- (channel_client_reconnect_tx is no longer used)

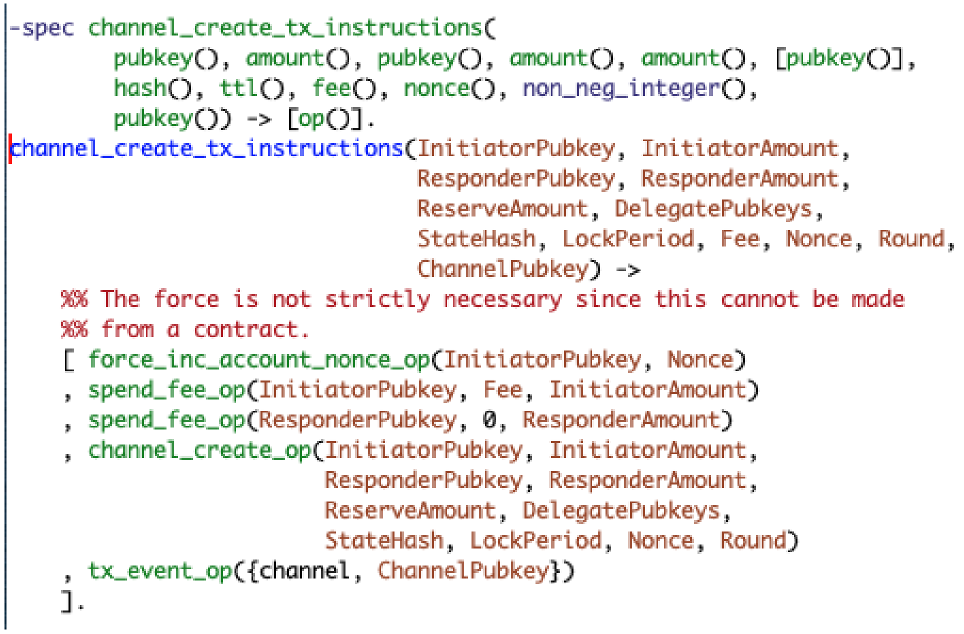




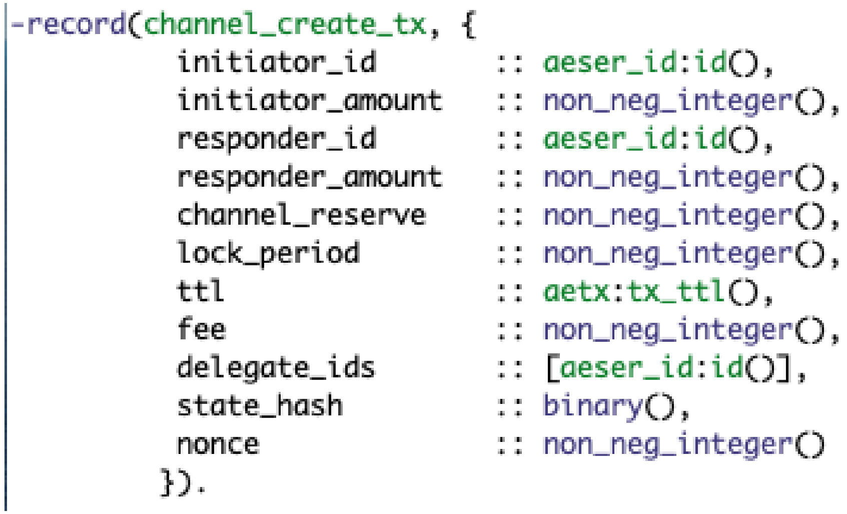
```
-type tx_type() :: spend_tx
                 | oracle_register_tx
                 | oracle_extend_tx
                 | oracle_query_tx
                 | oracle_response_tx
                 | name_preclaim_tx
                 | name_claim_tx
                 | name_transfer_tx
                 | name_update_tx
                 | name_revoke_tx
                 | contract_create_tx
                 | contract_call_tx
                 | ga_attach_tx
                 | ga_meta_tx
                 | channel_create_tx
                 | channel_deposit_tx
                 | channel_withdraw_tx
                 | channel_force_progress_tx
                 | channel_close_mutual_tx
                 | channel_close_solo_tx
                 | channel_slash_tx
                 | channel_settle_tx
                 | channel_snapshot_solo_tx
                 | channel_offchain_tx
                 | channel_client_reconnect_tx
                 | paying_for_tx.
```

channel_create_tx

- Creates the channel object
- Checks in aeprimop.erl

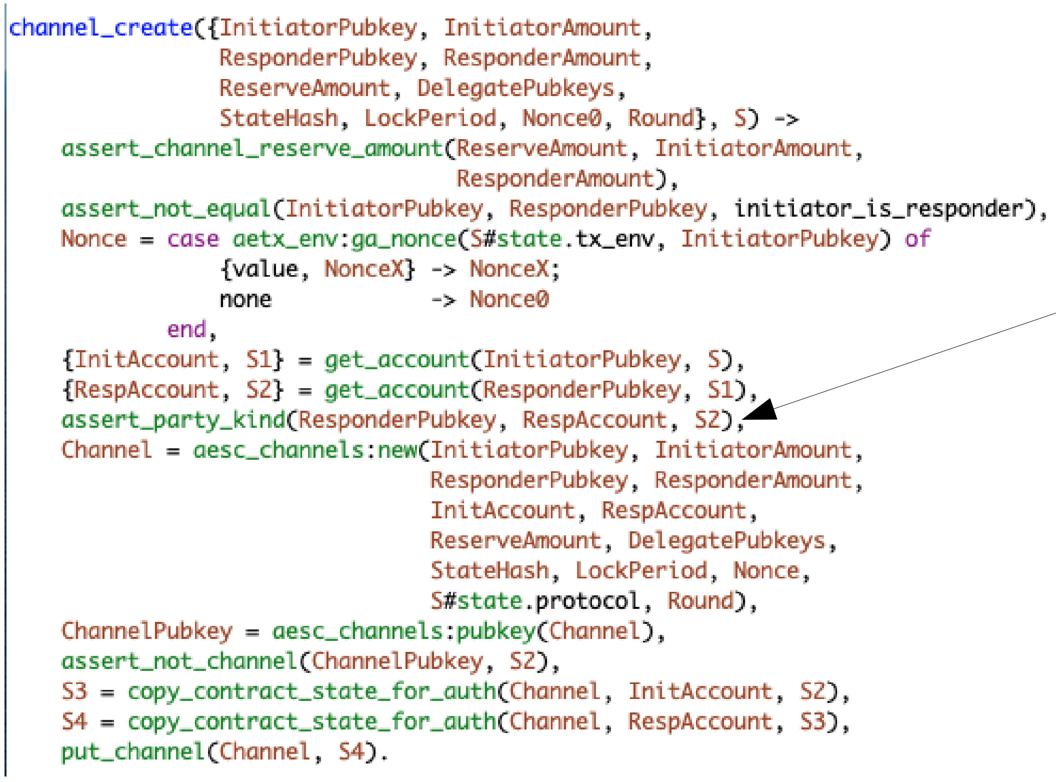


```
-spec channel_create_tx_instructions(
        pubkey(), amount(), pubkey(), amount(), amount(), [pubkey()],
        hash(), ttl(), fee(), nonce(), non_neg_integer(),
        pubkey()) -> [op()].
channel_create_tx_instructions(InitiatorPubkey, InitiatorAmount,
                               ResponderPubkey, ResponderAmount,
                               ReserveAmount, DelegatePubkeys,
                               StateHash, LockPeriod, Fee, Nonce, Round,
                               ChannelPubkey) ->
    %% The force is not strictly necessary since this cannot be made
    %% from a contract.
    [ force_inc_account_nonce_op(InitiatorPubkey, Nonce)
    , spend_fee_op(InitiatorPubkey, Fee, InitiatorAmount)
    , spend_fee_op(ResponderPubkey, 0, ResponderAmount)
    , channel_create_op(InitiatorPubkey, InitiatorAmount,
                        ResponderPubkey, ResponderAmount,
                        ReserveAmount, DelegatePubkeys,
                        StateHash, LockPeriod, Nonce, Round)
    , tx_event_op({channel, ChannelPubkey})
    ].
```



```
-record(channel_create_tx, {
          initiator_id        :: aeser_id:id(),
          initiator_amount    :: non_neg_integer(),
          responder_id        :: aeser_id:id(),
          responder_amount    :: non_neg_integer(),
          channel_reserve     :: non_neg_integer(),
          lock_period         :: non_neg_integer(),
          ttl                 :: aetx:tx_ttl(),
          fee                 :: non_neg_integer(),
          delegate_ids        :: [aeser_id:id()],
          state_hash          :: binary(),
          nonce               :: non_neg_integer()
         }).
```

channel_create_tx



```
channel_create({InitiatorPubkey, InitiatorAmount,
                ResponderPubkey, ResponderAmount,
                ReserveAmount, DelegatePubkeys,
                StateHash, LockPeriod, Nonce0, Round}, S) ->
    assert_channel_reserve_amount(ReserveAmount, InitiatorAmount,
                                  ResponderAmount),
    assert_not_equal(InitiatorPubkey, ResponderPubkey, initiator_is_responder),
    Nonce = case aetx_env:ga_nonce(S#state.tx_env, InitiatorPubkey) of
                {value, NonceX} -> NonceX;
                none            -> Nonce0
            end,
    {InitAccount, S1} = get_account(InitiatorPubkey, S),
    {RespAccount, S2} = get_account(ResponderPubkey, S1),
    assert_party_kind(ResponderPubkey, RespAccount, S2),
    Channel = aesc_channels:new(InitiatorPubkey, InitiatorAmount,
                                ResponderPubkey, ResponderAmount,
                                InitAccount, RespAccount,
                                ReserveAmount, DelegatePubkeys,
                                StateHash, LockPeriod, Nonce,
                                S#state.protocol, Round),
    ChannelPubkey = aesc_channels:pubkey(Channel),
    assert_not_channel(ChannelPubkey, S2),
    S3 = copy_contract_state_for_auth(Channel, InitAccount, S2),
    S4 = copy_contract_state_for_auth(Channel, RespAccount, S3),
    put_channel(Channel, S4).
```

Sanity check on account types

Note: Only type check is done on delegate accounts
No check is done e.g. to verify that they are actual accounts

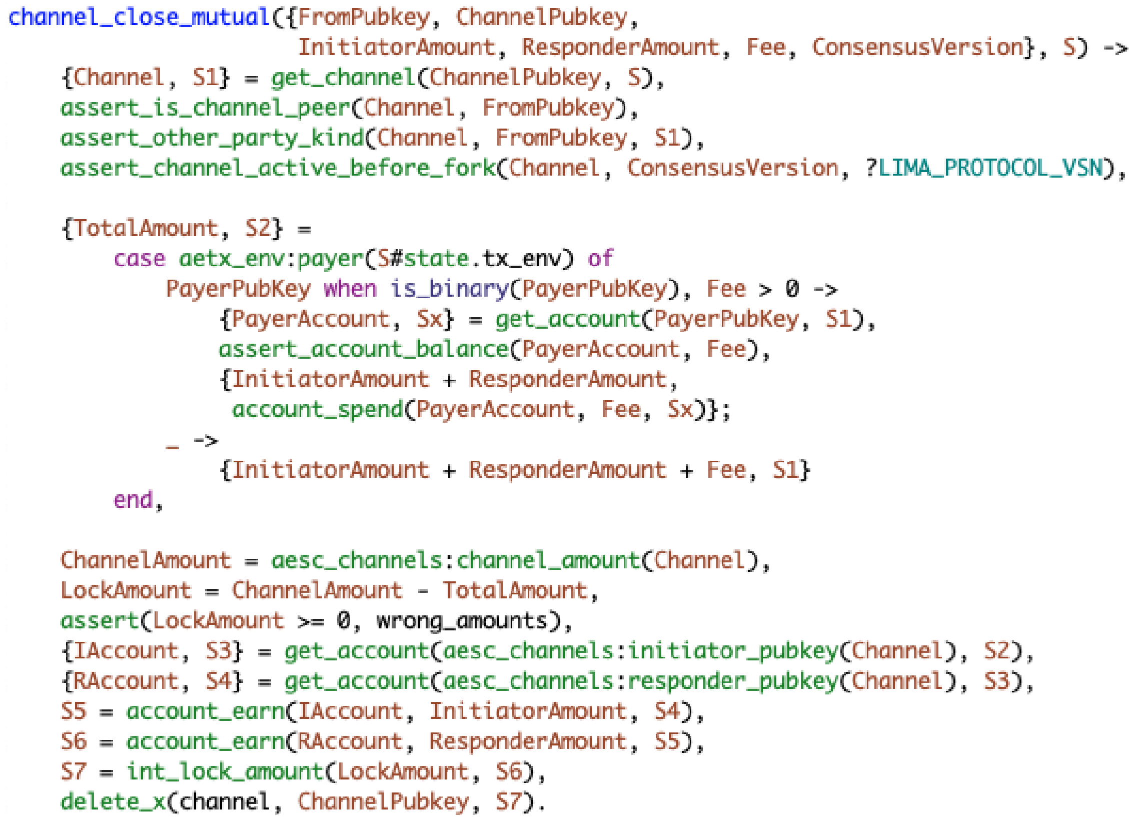
channel_deposit_tx

```
-spec channel_deposit_tx_instructions(pubkey(), pubkey(), amount(), hash(),
                                      non_neg_integer(), fee(), nonce()
                                     ) -> [op()].
channel_deposit_tx_instructions(FromPubkey, ChannelPubkey, Amount, StateHash,
                                Round, Fee, Nonce) ->
    [ inc_account_nonce_op(FromPubkey, Nonce)
    , spend_fee_op(FromPubkey, Fee, Amount)
    , channel_deposit_op(FromPubkey, ChannelPubkey, Amount, StateHash, Round)
    , tx_event_op({channel, ChannelPubkey})
    ].
```

```
channel_deposit_op(FromPubkey, ChannelPubkey, Amount, StateHash, Round) ->
    {channel_deposit, {FromPubkey, ChannelPubkey, Amount, StateHash, Round}}.

channel_deposit({FromPubkey, ChannelPubkey, Amount, StateHash, Round}, S) ->
    {Channel, S1} = get_channel(ChannelPubkey, S),
    assert_channel_active(Channel),
    assert_is_channel_peer(Channel, FromPubkey),
    assert_other_party_kind(Channel, FromPubkey, S1),
    assert_channel_round(Channel, Round, deposit),
    Channel1 = aesc_channels:deposit(Channel, Amount, Round, StateHash),
    put_channel(Channel1, S1).
```

Similar flow for withdraw

channel_close_mutual_tx

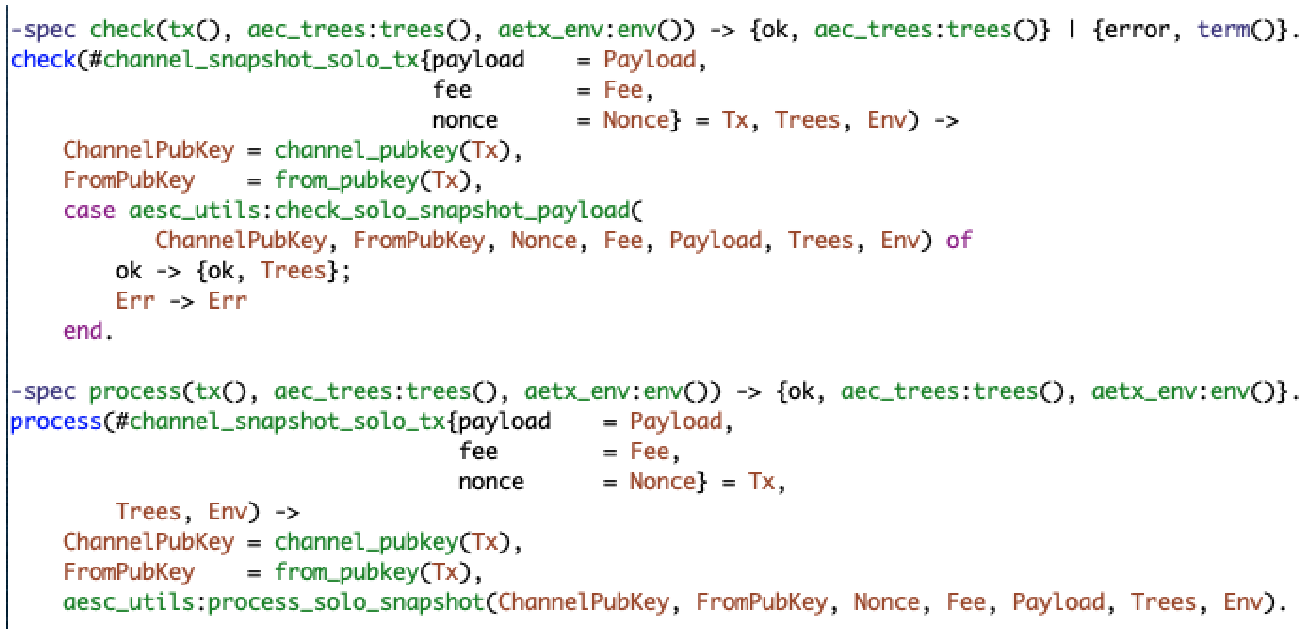


```
channel_close_mutual({FromPubkey, ChannelPubkey,
                      InitiatorAmount, ResponderAmount, Fee, ConsensusVersion}, S) ->
    {Channel, S1} = get_channel(ChannelPubkey, S),
    assert_is_channel_peer(Channel, FromPubkey),
    assert_other_party_kind(Channel, FromPubkey, S1),
    assert_channel_active_before_fork(Channel, ConsensusVersion, ?LIMA_PROTOCOL_VSN),

    {TotalAmount, S2} =
        case aetx_env:payer(S#state.tx_env) of
            PayerPubKey when is_binary(PayerPubKey), Fee > 0 ->
                {PayerAccount, Sx} = get_account(PayerPubKey, S1),
                assert_account_balance(PayerAccount, Fee),
                {InitiatorAmount + ResponderAmount,
                 account_spend(PayerAccount, Fee, Sx)};
            _ ->
                {InitiatorAmount + ResponderAmount + Fee, S1}
        end,

    ChannelAmount = aesc_channels:channel_amount(Channel),
    LockAmount = ChannelAmount - TotalAmount,
    assert(LockAmount >= 0, wrong_amounts),
    {IAccount, S3} = get_account(aesc_channels:initiator_pubkey(Channel), S2),
    {RAccount, S4} = get_account(aesc_channels:responder_pubkey(Channel), S3),
    S5 = account_earn(IAccount, InitiatorAmount, S4),
    S6 = account_earn(RAccount, ResponderAmount, S5),
    S7 = int_lock_amount(LockAmount, S6),
    delete_x(channel, ChannelPubkey, S7).
```

Note/TODO: The GA auth contract copies are not deleted

channel_snapshot_solo_tx

aesc_snapshot_solo_tx.erl



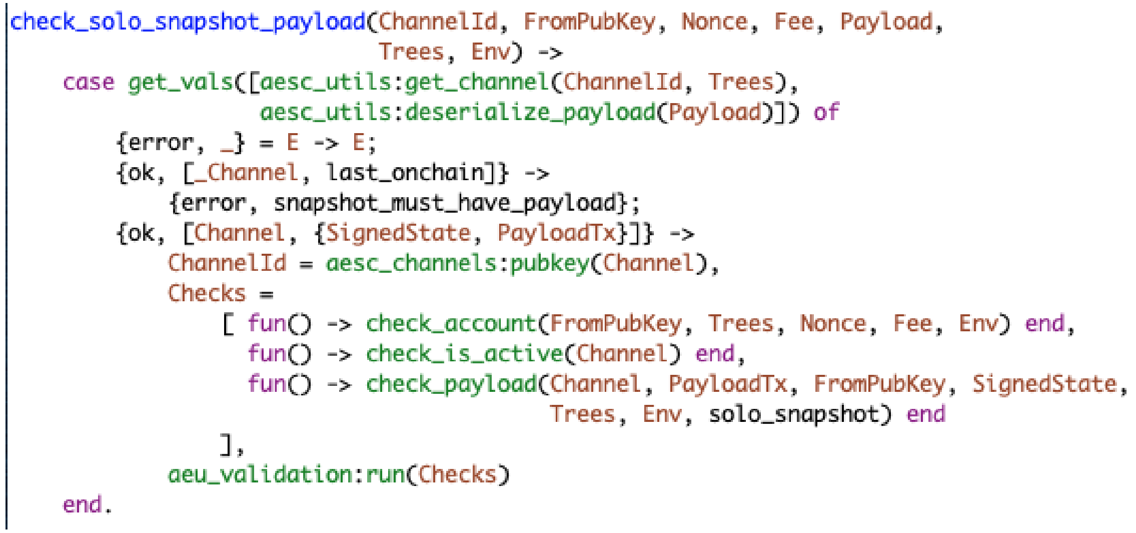
```
-spec check(tx(), aec_trees:trees(), aetx_env:env()) -> {ok, aec_trees:trees()} | {error, term()}.
check(#channel_snapshot_solo_tx{payload    = Payload,
                                fee        = Fee,
                                nonce      = Nonce} = Tx, Trees, Env) ->
    ChannelPubKey = channel_pubkey(Tx),
    FromPubKey    = from_pubkey(Tx),
    case aesc_utils:check_solo_snapshot_payload(
           ChannelPubKey, FromPubKey, Nonce, Fee, Payload, Trees, Env) of
        ok -> {ok, Trees};
        Err -> Err
    end.

-spec process(tx(), aec_trees:trees(), aetx_env:env()) -> {ok, aec_trees:trees(), aetx_env:env()}.
process(#channel_snapshot_solo_tx{payload    = Payload,
                                  fee        = Fee,
                                  nonce      = Nonce} = Tx,
        Trees, Env) ->
    ChannelPubKey = channel_pubkey(Tx),
    FromPubKey    = from_pubkey(Tx),
    aesc_utils:process_solo_snapshot(ChannelPubKey, FromPubKey, Nonce, Fee, Payload, Trees, Env).
```

Other txs processed by aesc_utils:

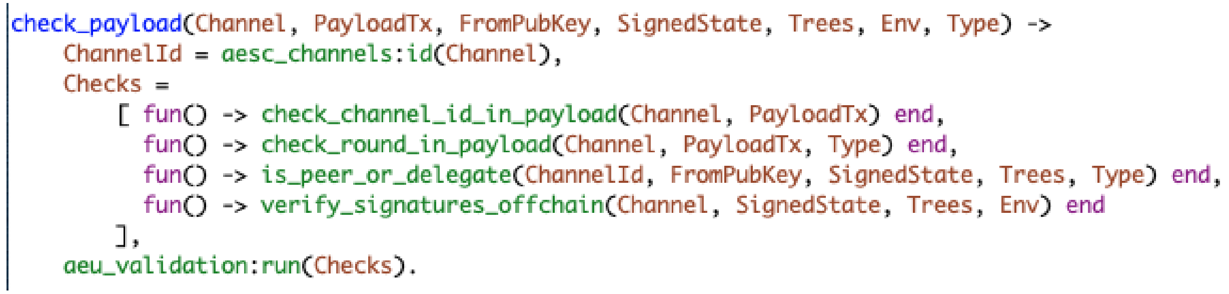
- close_solo
- force_progress
- slash

channel_snapshot_solo_tx

aesc_utils.erl



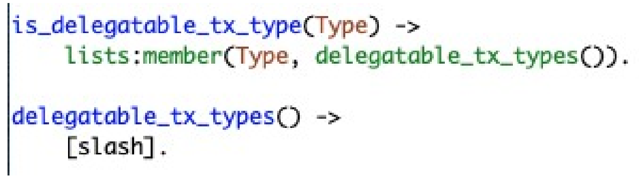
```
check_solo_snapshot_payload(ChannelId, FromPubKey, Nonce, Fee, Payload,
                            Trees, Env) ->
    case get_vals([aesc_utils:get_channel(ChannelId, Trees),
                   aesc_utils:deserialize_payload(Payload)]) of
        {error, _} = E -> E;
        {ok, [_Channel, last_onchain]} ->
            {error, snapshot_must_have_payload};
        {ok, [Channel, {SignedState, PayloadTx}]} ->
            ChannelId = aesc_channels:pubkey(Channel),
            Checks =
                [ fun() -> check_account(FromPubKey, Trees, Nonce, Fee, Env) end,
                  fun() -> check_is_active(Channel) end,
                  fun() -> check_payload(Channel, PayloadTx, FromPubKey, SignedState,
                                         Trees, Env, solo_snapshot) end
                ],
            aeu_validation:run(Checks)
    end.
```



```
check_payload(Channel, PayloadTx, FromPubKey, SignedState, Trees, Env, Type) ->
    ChannelId = aesc_channels:id(Channel),
    Checks =
        [ fun() -> check_channel_id_in_payload(Channel, PayloadTx) end,
          fun() -> check_round_in_payload(Channel, PayloadTx, Type) end,
          fun() -> is_peer_or_delegate(ChannelId, FromPubKey, SignedState, Trees, Type) end,
          fun() -> verify_signatures_offchain(Channel, SignedState, Trees, Env) end
        ],
    aeu_validation:run(Checks).
```

Delegates are an embryo to ”watch towers”.

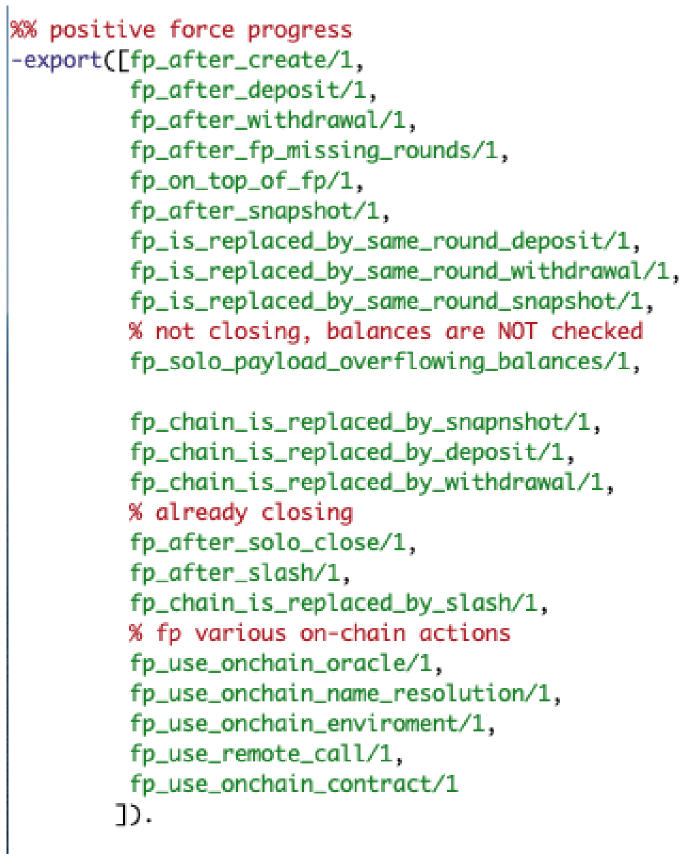
One could argue that snapshots should be delegatable.



```
is_delegatable_tx_type(Type) ->
    lists:member(Type, delegatable_tx_types()).

delegatable_tx_types() ->
    [slash].
```

Forced progress

- A whole talk in itself
- Can be used while open, if peer rejects a valid contract call
- Can also be used while closing, and can then be slashed
- FSM responds to FP events from chain, and must also roll forward if there are FPs after the state for reestablish.



```
%% positive force progress
-export([fp_after_create/1,
         fp_after_deposit/1,
         fp_after_withdrawal/1,
         fp_after_fp_missing_rounds/1,
         fp_on_top_of_fp/1,
         fp_after_snapshot/1,
         fp_is_replaced_by_same_round_deposit/1,
         fp_is_replaced_by_same_round_withdrawal/1,
         fp_is_replaced_by_same_round_snapshot/1,
         % not closing, balances are NOT checked
         fp_solo_payload_overflowing_balances/1,

         fp_chain_is_replaced_by_snapnshot/1,
         fp_chain_is_replaced_by_deposit/1,
         fp_chain_is_replaced_by_withdrawal/1,
         % already closing
         fp_after_solo_close/1,
         fp_after_slash/1,
         fp_chain_is_replaced_by_slash/1,
         % fp various on-chain actions
         fp_use_onchain_oracle/1,
         fp_use_onchain_name_resolution/1,
         fp_use_onchain_enviroment/1,
         fp_use_remote_call/1,
         fp_use_onchain_contract/1
        ]).
```